Sistema Cliente Com Interface Gráfica Para Conexão FTP Exercício Computacional I - Redes De Computadores

Rafael Gonçalves de Oliveria Vianal[1]

[1]Sistemas de Informação – Universidade Federal do Mato Grosso do Sul (UFMS)
Caixa Postal 79400-000 – Coxim – MS – Brazil

rafael.viana@aluno.ufms.br

Resumo. Este relatório descreve como foi implementado um sistema cliente com interface gráfica para conexão FTP, utilizando JavaFX em conjunto do Apache Commons Net 3.6, uma biblioteca de conexão FTP.

1. JavaFX

Foi escolhido o JavaFX para criar uma interface gráfica onde o usuário terá um melhor desempenho, ao utilizar o sistema. Para criar um sistema elegante foi utilizado uma biblioteca com novos elementos CSS, a bibliteca utilizada para essa finalidade foi a JFoenix, essa bibliteca é open source. Para icones foi utilizado a biblioteca fontawesomefx-8.9 essa biblioteca é open source.

2. Apache Commons Net 3.6

Para melhor desempenho nas conexões ftp, foi utilizada a biblioteca de conexão FTP da Apache Commons, onde a mesma se encontra atualmente na versão 3.6.

3. Problematica

O trabalho proposto tem como objetivo criar um client FTP, no qual tenha como Adicionar, Renomear, realizar Download e Upload de arquivos e pastas. Tendo como restrição o sistema deve apenas deixar criar 5 pastas e 2 arquivos por diretório, sendo que no máximo deve ser criados 3 níveis de diretórios.

A implementação de restrições no software cliente e não no software servidor, coloca a aplicação em risco, um usuário mal intencionado poderia utilizar outro software cliente para acessar os serviços do software servidor, já que o mesmo não possui restrições, assim o usuário mal intencionado passaria a ter privilégios.

Assim podemos observar uma falha na segurança dessas restrições, porém como essa aplicação e para fins acadêmicos, não focaremos nessa questão.

4. Metodologia

Como a aplicação é pequena a estrutura da mesma, foi dividida em 3 pacotes sendo eles: Cliente, Icons e Socket, mostradas na Figura 1

1. A pasta cliente é responsável pela parte de Interface Gráfica do Usuário, envolvendo controles e Views.
2. A pasta Icons armazena icons utilizados na pasta cliente.

3. A pasta socket e responsável por toda comunicação da Lib da Apache com os controles do cliente.

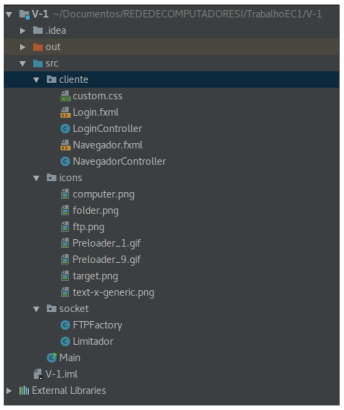

Figura 1. Imagem da Tela de Login.

Primeiramente a Class Main e invocada, chamando a Scene do Login.fxml, o controller do login e responsável por fornecer o necessario para o FTPClient poder fazer a conexão.

```
1
2  public class Main  extends Application{
3
4  @Override
5      public void start(Stage stage) throws Exception {
6      Parent root = FXMLLoader.load(getClasssingleton().
           getResource("/cliente/Login.fxml"));
7      Scene scene = new Scene(root);
8      stage.setScene(scene);
9      stage.show();
10 }
11
12
13 public static void main(String[] args) {
14     launch(args);
15
16 }
17
18 }
```

Figura 2. Imagem da Tela de Login.

Com a tela de login aberta o usuário entra com as informações login , senha, endereço do host, port do host mostrado no código abaixo e na Figura 2 .

```
1  private void login(ActionEvent event) {
2
3      btnLogin.setVisible(false);
4      imgProgress.setVisible(true);
5
6      PauseTransition pauseTransition = new PauseTransition();
7      pauseTransition.setDuration(Duration.seconds(3));
8      pauseTransition.setOnFinished(ev -> {
9
10 try {
11
12         int reply = FTPFactory.getInstance().FTPConecta(
               txtHostName.getText(),Integer.parseInt(txtHostPort.
               getText()),this.txtUsername.getText(),this.
               txtPassword.getText());
13         System.out.println("Igual:" + reply);
14
15         if (reply == 230) {
16
17             btnLogin.getScene().getWindow().hide();
18             completeLogin();
```

```
19
20          } else {
21
22              imgProgress.setVisible(false);
23              btnLogin.setVisible(true);
24              JOptionPane.showMessageDialog(null,"Erro Senha ou
                    Usuário incorreto !!", "Erro ao Logar",
                    JOptionPane.ERROR_MESSAGE);
25          }
26
27      } catch (IOException ex) {
28          Logger.getLogger(LoginController.class.getName()).log(
                Level.SEVERE, null, ex);
29      } catch (Exception ex) {
30          Logger.getLogger(LoginController.class.getName()).log(
                Level.SEVERE, null, ex);
31      }
32
33  });
34    pauseTransition.play();
35  }
36
37  private void completeLogin() throws IOException {
38
39      imgProgress.setVisible(false);
40      Stage dashboardStage = new Stage();
41      dashboardStage.setTitle("");
42      Parent root = FXMLLoader.load(getClass().getResource("
            Navegador.fxml"));
43      Scene scene = new Scene(root);
44      dashboardStage.setScene(scene);
45      dashboardStage.show();
46
47  }
```

Para poder realizar a comunicação entre as classes e o FTPClient da apache, foi criado uma classe Singleton chamada de FTPFactory onde a mesma cria uma getInstance de FTPClient, podendo ser chamada de qualquer classe sem ter que ser instanciada novamente, o que irá ocasionar a perda da conexão FTP.

```
1
2   public class FTPFactory {
3
4       private final FTPClient ftp;
5       private TreeItem<FTPFile> file;
6
7       private FTPFactory() {
8               this.ftp = new FTPClient();
9       }
10
```

```
11      public static FTPFactory getInstance() {
12
13          return FTPFactoryHolder.INSTANCE;
14      }
15
16
17      /**
18      * Classe privada que armazena a única instância de
           FTPFactory.
19      */
20
21      private static class FTPFactoryHolder {
22
23          private static final FTPFactory INSTANCE = new
               FTPFactory();
24      }
25
26
27      public FTPClient getFTP() {
28          return this.ftp;
29      }
30
31
32      public boolean Excluir(FTPFile file) {
33      try {
34              if (file.isDirectory()) {
35                  System.out.println(file.getLink());
36                  return ftp.removeDirectory(file.getLink());
37              } else {
38                  System.out.println(file.getLink());
39                  return ftp.deleteFile(file.getLink());
40              }
41      } catch (IOException e) {
42          e.printStackTrace();
43          }
44          return false;
45      }
46
47
48      public int FTPConecta(String host, int port, String user,
           String pwd) throws Exception {
49          int reply;
50          ftp.connect(host, port);
51          reply = ftp.getReplyCode();
52          if (!FTPReply.isPositiveCompletion(reply)) {
53              ftp.disconnect();
54              throw new Exception("Exception in connecting to FTP
                   Server");
55          }
```

```
56          ftp.login(user, pwd);
57          reply = ftp.getReplyCode();
58          ftp.setFileType(FTPClient.BINARY_FILE_TYPE);
59          ftp.enterLocalPassiveMode();
60          ftp.setAutodetectUTF8(true);
61          return reply;
62      }
63
64
65      public void disconnect() {
66          if (this.ftp.isConnected()) {
67          try {
68              this.ftp.logout();
69              this.ftp.disconnect();
70              } catch (IOException f) {
71
72              }
73          }
74      }
75  }
```

Após a Class login em conjunto com a Class FTPFactor, validar os dados de usuário a tela de navegação e aberta, essa tela nada mais é do que um conjunto de botões em uma Grid a esquerda e um TreeView do JavaFX no centro para poder navegar pela estutura dos diretórios.

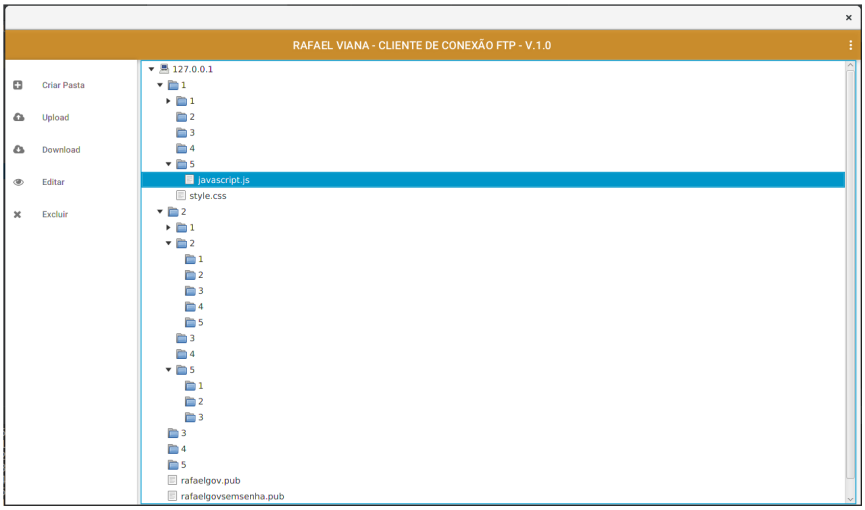

Figura 3. Imagem da Tela de Navegação.

Toda lógica de abastecimento da TreeView e a criação dos TreeItem utilizados na TreeView estão na Class navegação, onde inicia a criação dos TreeItem da toda a estrutura da arvore como ADICIONAR, REMOVER, LISTAR, EDITAR, DOWNLOAD e UPLOAD de arquivos/pastas, o método CarregarFiles(), cria

recursivamente Nodes/TreeItem apartir dos files no servidor FTP, posteriormente encadeando os TreeItems e lançando-os na TreeView, facilitando assim a navegação.

```
1
2  private void Navegacao() throws IOException {
3      FTPFile files[];
4      TreeItem<FTPFile> treeRoot;
5      files = FTPFactory.getInstance().getFTP().listFiles();
6      Tree.setEditable(true);
7      if (files != null && files.length > 0) {
8          files[0].setRawListing(FTPFactory.getInstance().getFTP()
               .getPassiveHost());
9          treeRoot = CarregarFiles(files[0], true);
10     } else {
11         FTPFile file = new FTPFile();
12         file.setType(FTPFile.DIRECTORY_TYPE);
13         file.setLink(FTPFactory.getInstance().getFTP().
               printWorkingDirectory());
14         file.setRawListing(FTPFactory.getInstance().getFTP().
               getPassiveHost());
15         treeRoot = new TreeItem<>(file, new ImageView(computador
               ));
16     }
17
18     Tree.getSelectionModel().select(treeRoot);
19     Tree.setRoot(treeRoot);
20
21     //Os Botões ADD,RENAME,DELETE,UPLOAD e DOWNLOAD estão nessa
           classe, foi retirada para um melhor o entendimento.
22
23 }
24
25
26 public TreeItem<FTPFile> CarregarFiles(FTPFile directory,
       boolean v) throws IOException {
27
28     TreeItem<FTPFile> root;
29
30     if (v) {
31         directory.setType(FTPFile.DIRECTORY_TYPE);
32         directory.setLink(FTPFactory.getInstance().getFTP().
               printWorkingDirectory());
33         root = new TreeItem<FTPFile>(directory, new ImageView(
               computador));
34     } else {
35         root = new TreeItem<FTPFile>(directory, new ImageView(
               pasta));
36     }
37     root.setExpanded(true);
38     FTPFile[] files = FTPFactory.getInstance().getFTP().
```

```
        listFiles();
39      for (FTPFile f : files) {
40          System.out.println("Carregando .. " + f.getName());
41          if (f.isDirectory()) {
42              FTPFactory.getInstance().getFTP().
                    changeWorkingDirectory(f.getName());
43              f.setLink(FTPFactory.getInstance().getFTP().
                    printWorkingDirectory());
44              root.getChildren().add(CarregarFiles(f, false));
45          } else {
46          f.setLink(FTPFactory.getInstance().getFTP().
                printWorkingDirectory() + separador + f.getName());
47          root.getChildren().add(new TreeItem<FTPFile>(f, new
                ImageView(this.arquivo)));
48          }
49      }
50      FTPFactory.getInstance().getFTP().changeToParentDirectory();
51      return root;
52  }
```

Um dos objetivos do trabalho era limitar o cliente FTP, fazendo com que usuários que utilizar o sistema só poderam criar 5 pastas e 2 arquivos por diretório, sendo que poderá criar no máximo 3 níveis de diretórios.

A classe Limitador do pacote socket e responsável por fazer a armazenagem da quantidade máxima de pastas e arquivos. Sendo utilizada na classe navegação do pacote cliente, em conjunto dos métodos limiteArquivo(), limitePasta() e limiteNivel().

```
1   public class Limitador {
2
3       private int p;
4       private int a;
5
6       public Limitador(int p, int a) {
7           this.p = p;
8           this.a = a;
9       }
10
11      public int getMP() {
12          return p;
13      }
14
15      public int getMA() {
16          return a;
17      }
18
19  }
20
21  // Os métodos abaixo pertencem Class NavegadorControoler
```

```
22
23 private boolean limiteArquivo() throws IOException {
24     int num = FTPFactory.getInstance().getFTP().listDirectories
       ().length;
25     int numa = FTPFactory.getInstance().getFTP().listFiles().
       length - num;
26     return numa < limite.getMA();
27 }
28
29 private boolean limitePasta() throws IOException {
30     int num_diretorios = FTPFactory.getInstance().getFTP().
       listDirectories().length;
31     return num_diretorios < limite.getMP();
32 }
33
34 private int limiteNivel() throws IOException {
35     int num = FTPFactory.getInstance().getFTP().
       printWorkingDirectory().split("/").length;
36     return num;
37 }
```

Para deletar uma pasta deve-se utilizar o método removeDirectory(Caminho), ou para deletar um arquivo utiliza-se o método deleteFile(Caminho) do FTPClient Apache. Porém para deletar uma pasta que contém N sub pastas deve-se percorrer recursivamente as pastas, até a folha mais baixa e vim apagando debaixo para cima cada pasta ou arquivo de cada pasta recursivamente pelo método DeletarRecursivo(TreeItem). Para renomear um arquivo ou pasta utiliza-se o método rename(caminho,novonome) da Apache. Porém para mudar os links das sub pastas de um diretório

```
1
2 public boolean DeletarRecursivo(TreeItem<FTPFile> a) {
3     boolean flag = false;
4
5     if (a.getChildren().isEmpty()) {
6         if (FTPFactory.getInstance().Excluir(a.getValue())) {
7             return true;
8         }
9     }else {
10
11         for (TreeItem<FTPFile> iterator: a.getChildren()){
12             DeletarRecursivo(iterator);
13         }
14
15         if (FTPFactory.getInstance().Excluir(a.getValue())) {
16             return true;
17         }
18     }
19     return false;
20 }
```

```
21
22 public void RenameRecursivao(TreeItem<FTPFile> a) {
23     String novolink;
24     for (Iterator<TreeItem<FTPFile>> iterator = a.getChildren().
          iterator(); iterator.hasNext(); ) {
25         TreeItem<FTPFile> c = iterator.next();
26         novolink = a.getValue().getLink() + separador + c.
              getValue().getName();
27         c.getValue().setLink(novolink);
28         if (!c.getChildren().isEmpty()) {
29             RenameRecursivao(c);
30         }
31     }
32 }
```