

Sistema Cliente Para Conexão FTP

Exercício Computacional I - Redes de Computadores

Rafael Gonçalves de Oliveria Viana¹

¹Sistemas 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.[1]

Para criar um sistema elegante foi utilizada uma biblioteca com novos elementos CSS, a biblioteca utilizada para essa finalidade foi a JFoenix, esta é uma biblioteca java de código aberto, que implementa o Google Material Design usando componentes java.[2]

Para ícones foi utilizada a biblioteca fontawesomefx-8.9, open source.[3]

2. Apache Commons Net 3.6

Para melhor desempenho nas conexões FTP, foi utilizada a biblioteca de conexão FTPClient da Apache Commons, onde a mesma se encontra atualmente na versão 3.6.[4]

3. VSFTPD

Para realização de testes de desenvolvimento foi utilizado o VSFTPD como servidor FTP, assim agilizando a produção do software cliente. O VSFTPD é um servidor de FTP licenciado pela GPL para sistemas UNIX, incluindo o Linux. É seguro e extremamente rápido.[5]

4. Problemática

O trabalho proposto tem como objetivo criar um cliente FTP, no qual tenha como Adicionar, Renomear, realizar Download e Upload de arquivos e pastas. Tendo como restrição, o sistema deve permitir apenas a criação de 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.[6]

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

5. 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. O pacote cliente é responsável pela interface gráfica do usuário, onde estão os controles e views.
2. O pacote icons armazena ícones utilizados no pacote cliente.
3. O pacote socket é responsável por toda comunicação da biblioteca Apache Commons Net 3.6 com os controles do pacote cliente.

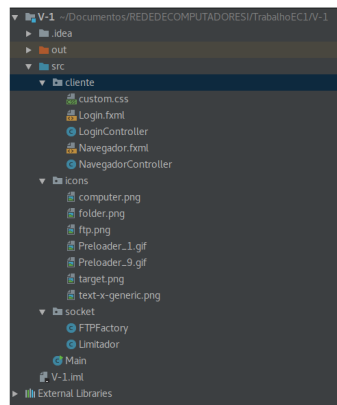


Figura 1. Imagem da Tela de Login.

Primeiramente, a classe Main é invocada, chamando a Scene do Login.fxml, o controle do login é responsável por fornecer o necessário para o FTPClient 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(getClass().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(
13            txtHostName.getText(), Integer.parseInt(txtHostPort.
14                getText()), this.txtUsername.getText(), this.
15                txtPassword.getText());
16        System.out.println("Igual:" + reply);
17
18        if (reply == 230) {
19
20            btnLogin.getScene().getWindow().hide();
21            completeLogin();
22
23        } else {
```

```

21         imgProgress.setVisible(false);
22         btnLogin.setVisible(true);
23         JOptionPane.showMessageDialog(null, "Erro Senha ou
24         Usuário incorreto !!", "Erro ao Logar",
25         JOptionPane.ERROR_MESSAGE);
26     }
27     } catch (IOException ex) {
28         Logger.getLogger(LoginController.class.getName()).log(
29         Level.SEVERE, null, ex);
30     } catch (Exception ex) {
31         Logger.getLogger(LoginController.class.getName()).log(
32         Level.SEVERE, null, ex);
33     }
34     });
35     pauseTransition.play();
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("
43     Navegador.fxml"));
44     Scene scene = new Scene(root);
45     dashboardStage.setScene(scene);
46     dashboardStage.show();
47 }

```

Para realizar a comunicação entre as classes e o FTPClient da apache, criou-se uma classe Singleton chamada de FTPFactory onde a mesma utiliza uma getInstance de FTPClient, que pode 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
19     * FTPFactory.
20     */
21     private static class FTPFactoryHolder {
22
23         private static final FTPFactory INSTANCE = new
24             FTPFactory();
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,
49         String pwd) throws Exception {
50         int reply;
51         ftp.connect(host, port);
52         reply = ftp.getReplyCode();
53         if (!FTPReply.isPositiveCompletion(reply)) {
54             ftp.disconnect();
55             throw new Exception("Exception in connecting to FTP
56                 Server");
57         }
58         ftp.login(user, pwd);
59         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 classe login em conjunto com a classe FTPFactory, validar os dados de usuário a tela de navegação é aberta, essa tela nada mais é do que um conjunto de botões em uma Grid à esquerda um ToolBar no top e um TreeView do JavaFX no centro para poder navegar pela estrutura dos diretórios.

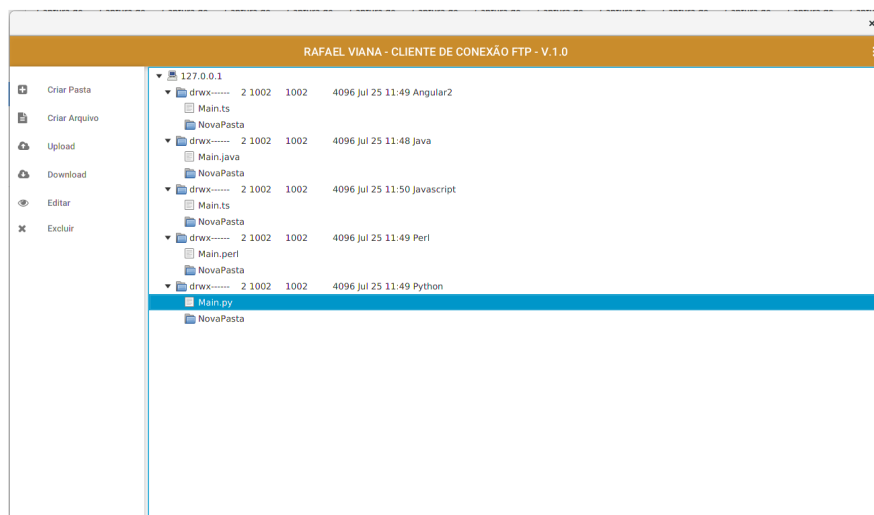


Figura 3. Imagem da Tela de Navegação.

Toda lógica de abastecimento da TreeView e a criação dos TreeItem utilizados na TreeView estão no método Navegacao(), pertencente a classe NavegadorController, o mesmo chama o método CarregarFiles(), que tem a obrigação de criar recursivamente a partir dos arquivos que estão no servidor FTP, os TreeItem<FTPFile> que são encadeando-os e lançando na TreeView, facilitando assim a navegação pelos diretórios.

Para melhor leitura os métodos implementados pelos botões foram removidos, porém podem ser verificados na classe NavegadorController no código fonte. As funções que os mesmos exercem são com base nos comandos implementados no FTPClient da Apache sendo estes:

1. Criar Pasta `makeDirectory(String caminho)`
Este método exige um caminho do diretório remoto para criar uma pasta no mesmo.
2. Criar Arquivo `makeDirectory()`
Este método exige um caminho do diretório remoto para criar um arquivo no mesmo.
3. Upload Arquivo `storeFile(String caminho, InputStream local)`
Este método exige um caminho do diretório remoto e um `InputStream` local, que indique o arquivo desejado.
4. Download `retrieveFile(String caminho, OutputStream local)`
Este método exige um caminho do diretório remoto que tenha o arquivo e `OutputStream` para o local de destino do arquivo.
5. Editar `rename(String nomeAtual, String novoNome)`
Este método exige o nome do diretório remoto que deseja alterar e o novo nome para o mesmo.
6. Excluir `removeDirectory(String diretorioRemoto)` ou `deleteFileString(String arquivoRemoto)`
Para excluir foram utilizados dois métodos, um para deletar as pastas `removeDirectory(String caminhoRemoto)` e outro para arquivos `deleteFileString(String caminhoRemoto)`.

```
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().
9             .getPassiveHost());
10        treeRoot = CarregarFiles(files[0], true);
11    } else {
12        FTPFile file = new FTPFile();
13        file.setType(FTPFile.DIRECTORY_TYPE);
14        file.setLink(FTPFactory.getInstance().getFTP().
15            printWorkingDirectory());
16        file.setRawListing(FTPFactory.getInstance().getFTP().
17            getPassiveHost());
18        treeRoot = new TreeItem<>(file, new ImageView(computador
19            ));
20    }
21
22    Tree.getSelectionModel().select(treeRoot);
23    Tree.setRoot(treeRoot);
24
25 }
```

```

21 //Os Botões ADD,RENAME,DELETE,UPLOAD e DOWNLOAD estão nessa
    classe , foram retirados para um melhor entendimento do
    código .
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 é limitar o cliente FTP, fazendo com que usuários que utilizam o sistema possam criar 5 pastas e 2 arquivos por diretório, sendo que poderão criar no máximo 3 níveis de diretórios.

A classe Limitador do pacote socket é responsável por fazer a armazenagem da quantidade máxima de pastas e arquivos. Sendo utilizada na classe Navega-

dorController 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 a classe NavegadorControoler
22
23 private boolean limiteArquivo() throws IOException {
24     int num = FTPFactory.getInstance().getFTP().listDirectories
25         ().length;
26     int numa = FTPFactory.getInstance().getFTP().listFiles().
27         length - num;
28     return numa < limite.getMA();
29 }
30
31 private boolean limitePasta() throws IOException {
32     int num_diretorios = FTPFactory.getInstance().getFTP().
33         listDirectories().length;
34     return num_diretorios < limite.getMP();
35 }
36
37 private int limiteNivel() throws IOException {
38     int num = FTPFactory.getInstance().getFTP().
39         printWorkingDirectory().split("/").length;
40     return num;
41 }
```

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 seguir apagando debaixo para cima cada pasta ou arquivo, recursivamente pelo método DeletarRecursivo(TreeItem),

implementado na classe NavegadorController.

```
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 }
```

Após a renomeação de um `TreeItem<FTPFile>` da `TreeViw`, foi preciso modificar recursivamente os `TreeItem<FTPFile>` filhos, pelo método `RenameRecursivao(TreeItem<FTPFile> a)`, esse método notifica os filhos mudando assim o link dos mesmos.

```
1 public void RenameRecursivao(TreeItem<FTPFile> a) {
2     String novolink;
3     for (Iterator<TreeItem<FTPFile>> iterator = a.getChildren().
4         iterator(); iterator.hasNext(); ) {
5         TreeItem<FTPFile> c = iterator.next();
6         novolink = a.getValue().getLink() + separador + c.
7             getValue().getName();
8         c.getValue().setLink(novolink);
9         if (!c.getChildren().isEmpty()) {
10             RenameRecursivao(c);
11         }
12     }
13 }
```

6. Testes

Foram realizados testes funcionais para correções de erros, na implementação. Para uma maior segurança foram realizados testes nos métodos de criação de pastas e arquivos, exclusão de pastas e arquivos, adicionar arquivos e pastas, upload de arquivos, renomeação arquivos e pastas, download de arquivos e exclusão de diretório inteiro.

Referências

- [1] K. Copyright, JavaFX Developer's Guide. Pearson Education, 2010.
- [2] J. Copyright, JFoenix Developer's Guide.
- [3] J. DELIGHT, Introducing: FontAwesomeFX 8.1.
- [4] Apache, Apache Commons Net 3.6.
- [5] M. Bauer, "Paranoid penguin—secure anonymous ftp with vsftpd," LINUX® Journal, Jul, vol. 1, 2004.
- [6] A. R. C. d. Rocha, J. C. Maldonado, and K. C. Weber, "Qualidade de software," São Paulo: a Pretince Hall, 2001.