Segurança e Confiabilidade

2016/2017

Relatório e Discussão – Etapa 1

Grupo 005

Autores

Francisco João Guimarães Coimbra de Almeida Araújo, n.º 45701

João Marques de Barros Mendes Leal, n.º 46394

Joana Correia Magalhães Sousa, n.º 47084

Nota Introdutória:

Todos os objetivos foram compridos com sucesso.

Configuração da Sandbox

Na pasta do projeto, onde se encontram as pastas src e bin, colocar os ficheiros server.policy e cliente.policy. Para configurar as sanboxes devem ser seguidos os seguintes passos:

1. No Eclipse, selecionar a opção “Run Configurations…”;
2. Clicar duas vezes na opção “Java Aplication”;
3. Para configurar o cliente escrever “myGit” no campo “Main class”;
4. Selecionar a opção “Arguments”;
5. No campo “VM arguments” escrever:
   “-Djava.security.manager -Djava.security.policy=client.policy”;
6. No campo “Name”, no campo superior do ecrã, escrever “myGit”;
7. Selecionar a opção “Apply”.

Para configurar a sandbox do servidor devem ser seguidos os mesmo passos, mas substituindo “myGit” por “myGitServer” e em vez de “(…) cliente.policy” escrever “(…) server.policy”.

Funcionamento da Sandbox

Sanbox do cliente:

O cliente faz o pedido de ligação usando o porto 2345. O cliente vai ter permissão para ler e escrever todos os ficheiros.

Sandbox do servidor:

O servidor fica à espera dos pedidos de ligação através do porto 23456 e aceita qualquer ligação de um porto com número maior ou igual a 1024. O servidor pode ler escrever e apagar qualquer ficheiro.

Tanto para o cliente como para o servidor o endereço IP das ligações ou pedidos não são relevantes.

Organização do Software

myGit.java:

Esta classe é o cliente e tem como funções iniciar e correr o cliente através da criação de uma nova thread e estabelecer a ligação entre o cliente e o servidor. Para além disso esta classe faz os pedidos ao cliente. Os pedidos são feitos por intermédio de uma outra classe.

myGitServer.java:

Esta classe é o servidor e tem como função iniciar e criar o servidor, criando uma thread e verificando se os dados provenientes do cliente se referem a um cliente que já existe, caso não exista esse cliente, ele será criado. Esta ultima parte de verificar a existência do cliente com os dados fornecidos e adicioná-lo caso ele não exista é tratada por outra classe. Esta classe também processa os pedidos do cliente e chama uma classe para tratar deles.

ClientServerHandle.java:

Esta classe trata das interações entre o cliente e o servidor. Recebe os pedidos do cliente e encaminha-os para o servidor. Estes pedidos são aqueles que foram pedidos no enunciado, adicionar um cliente, fazer o push dos ficheiros e dos repositórios, fazer pull, indica com quem é partilhado o repositório, remove um ficheiro do repositório e indica quando foi a última vez que o ficheiro foi modificado.

ServerClientHandle.java:

Esta classe trata dos pedidos do cliente que são encaminhados através do servidor, para além de verificar se o cliente existe e de o adicionar caso isso não se verifique. Para além desses pedidos esta classe também adiciona um ficheiro ao histórico do repositório, devolve o nome dos ficheiros e dos repositórios e verifica se um cliente tem acesso a um repositório, esta verificação é feita com recurso a uma outra classe.

userCatalog.java:

Esta classe gere os clientes, adicionando-os, caso sejam novos, escrevendo-os num ficheiro e consultando esse ficheiro quando é necessário saber se um cliente já existe e para fazer o login de um cliente.

repCatalog.java:

Esta classe gere os repositórios, criando novos, adicionado um novo utilizador ao repositório, indica quem foi o criador de um repositório, remove um utilizador do repositório, indica quando é que um repositório foi modificado pela última vez e também dá informação sobre se um utilizador tem acesso a um repositório.

Messager.java:

Esta classe cria e trata das mensagens trocadas entre o cliente e o servidor.

Mensagens:

O nosso sistema envia cinco tipos diferentes de mensagens, uma para o envio de um ficheiro, uma para a receção de um ficheiro, uma para fazer push para um repositório e também para o confirmar e rejeitar uma operação. As mensagens de envio, confirmação e rejeição são do tipo void, a receção de um ficheiro é do tipo File e a de push tem tipo boolean.

Requisitos de Segurança:

O nosso sistema no seu atual estado tem bastantes vulnerabilidades. Para as resolver o sistema tem vários requisitos, nomeadamente:

- Confidencialidade dos dados;

- Privacidade, por exemplo, na troca de mensagens entre o cliente e o servidor;
- Integridade dos ficheiros, ou seja, proteger os ficheiros de acessos e alterações indevidas;
- Integridade do sistema;
- Disponibilidade, assegurar que o sistema não é desativado por ataques externos;
- Autenticidade, garantir que um cliente é de facto real e não intruso;
- Prestação de contas.

Mecanismos de Segurança:

De modo a garantir estes requisitos, o nosso sistema deveria implementar:

- Criptografia de modo a cifrar as comunicações entre o cliente e o servidor;
- Firewall, para filtrar as mensagens que são recebidas, excluindo as que originam de um intruso, ou utilizador não autorizado;
- Mecanismos de controlo de acesso, para se certificar que os ficheiros apenas são acedidos pelos utilizadores com permissão para tal.

Código Fonte:

ClientServerHandler:

```
/**Grupo sc005
 * Francisco João Guimarães Coimbra de Almeida Araújo nº45701
 * Joana Correia Magalhães Sousa nº47084
 * João Marques de Barros Mendes Leal nº46394
 */



import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Scanner;
/**
 * A ideia desta classe eh tratar das interaccoes do myGit com o myGitServer
 */
public class ClientServerHandler {

	private static final int PUSH_REP = 20;
	private static final int PUSH_FILE = 30;
	private static final int PULL = 40;
	private static final int SHARE = 50;
	private static final int REMOVE = 60;

	private String username;
```

```
private String passwd;
private Messager msg;



/**
 * Construtor de ClientServerHandler
 * @param username - nome do cliente
 * @param passwd - password do cliente
 */
public ClientServerHandler(String username,String passwd){
	this.username = username;
	this.passwd = passwd;
	msg = new Messager();
}

/**
 * Envia para o servidor os dados relativamente sobre o cliente em questao
 * @param outStream - objeto por onde escreve ao servidor
 */
public void sendInitInfo(ObjectOutputStream outStream){
	try {
		outStream.writeObject(username);
		outStream.writeObject(passwd);
	} catch (IOException e) {
		e.printStackTrace();
	}
}

/**
 * Metodo que cria um novo utilizador caso este nao exista
 * @param outStream - objeto por onde escreve ao servidor
 * @param inStream - objeto por onde le ao servidor
```

```
	 */
	public int addUser(ObjectOutputStream outStream, ObjectInputStream inStream){
		try {
			outStream.flush();
			if(inStream.readShort() == 1){ //se nao existir
				System.out.println("--O utilizador " + username + "vai ser
criado");
				System.out.println("Confirmar password do utilizador " +
username);
				Scanner reader = new Scanner(System.in);
				String comp = reader.nextLine();
				reader.close();
				if( passwd.equals(comp)){
					msg.confirm(outStream);
					if(inStream.readShort() == 1){ //ler
						System.out.println("--O utilizador "
+username+ " foi"
								+ " criado com sucesso");
						return 1;
					}
					else{
						System.out.println("--Ouve um erro na criacao"
								+ " do utilizador");
						return -1;
					}
				}
				else{
					msg.reject(outStream);
					System.out.println("Erro:Ocorreu um erro a criar o
utilizador");
					return -1;
				}
			}
			else{
```

```
                        if(inStream.readShort() == -1){
                                System.out.println("Erro:Password errada!");
                                return -1;
                        }
                        return 0;
                }
        } catch (IOException e) {
                e.printStackTrace();
        }
        return -1;
}


/**
 * Funcao que faz push de um ficheiro para o repositorio que se encontra no servidor
 * que tem o mesmo nome que o repositorio que se encontra localmente
 * @param file - ficheiro ao qual fazemos push
 * @param filename - nome do ficheiro ao qual vamos fazer push
 * @param outStream - objeto por onde escreve ao servidor
 * @param inStream - objeto por onde le ao servidor
 */
public void push_file(File file,String filename,
                ObjectOutputStream outStream, ObjectInputStream inStream){
        try {
                outStream.writeInt(PUSH_FILE);
                outStream.writeObject(filename);
                outStream.flush();
                if(inStream.readShort() == -1){
                        System.out.println("Erro:Nao tem permissao para entrar nesse
ficheiro");
                        return;
                }
                if(msg.basic_push(file,outStream, inStream))
```

```
                        System.out.println("-- O ficheiro " + file.getName() + " foi"
                                        + " enviado para o servidor");
                    else
                        System.out.println("-- O ficheiro " + file.getName() + "ja "
                                        + "se encontra actualizado no servidor");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }


            /**
             * Funcao que faz push de todos os ficheiros que se encontram no repositorio para o
servidor
             * com o repositorio com o mesmo nome
             * Caso exista um ficheiro no servidor mas nao localmente, apaga-se o ficheiro do
servidor
             * @param rep - repositorio ao qual vamos fazer push
             * @param repName - nome do repositorio ao qual vamos fazer push
             * @param outStream - objeto por onde escreve ao servidor
             * @param inStream - objeto por onde le ao servidor
             */
            public void push_rep(File rep,String repName,ObjectOutputStream outStream,
                    ObjectInputStream inStream){
                try {
                    outStream.writeInt(PUSH_REP);
                    int total = 0;
                    outStream.writeObject(repName);
                    short resp;
                    if((resp = inStream.readShort()) == 1){
                        total ++;
                        System.out.println("-- O repositorio " +repName+" foi criado
no"
```

```
                        + " servidor");
}
else if(resp == 0){
        System.out.println("Erro: nao pode criar repositorios "
                        + "nas paginas de outros utilizadores");
        return;
}

File[] files = rep.listFiles( new FileFilter(){
        @Override
   public boolean accept(File pathname) {
      return pathname.isFile();
   }
});

outStream.flush();
if(inStream.readShort() == -1){
        System.out.println("Erro: Nao tem acesso a esse repositorio");
        return;
}
int size = files.length;
outStream.writeInt(size); //numero de ficheiros que vamos enviar
outStream.flush();
for(File fl: files){
        outStream.writeObject(rep.getName() + "/" +fl.getName());
        outStream.flush();
        if(msg.basic_push(fl,outStream, inStream)){
                total ++;
                System.out.println("-- O ficheiro " + fl.getName() +
                                " vai ser adicionado ao servidor");
        }
}
```

```
                        short delSize = inStream.readShort();
                        for(int i = 0; i < delSize; i++){
                                try {
                                        total ++;
                                        System.out.printf("-- O ficheiro %s foi apagado do
servidor\n"
                                                        ,(String)inStream.readObject());
                                } catch (ClassNotFoundException e) {
                                        e.printStackTrace();
                                }
                        }
                        if (total == 0)
                                System.out.println("-- Nao foi mudificado nada no lado do"
                                                + " servidor");
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }


        /**
         * Funcao que faz pull de um ficheiro de um servidor para o repositorio local
         * @param filename - nome do ficheiro ao qual vamos fazer pull
         * @param outStream - objeto por onde escreve ao servidor
         * @param inStream - objeto por onde le ao servidor
         */
        public void pull(String filename,
                        ObjectOutputStream outStream, ObjectInputStream inStream){
                try {
                        outStream.writeInt(PULL);

                        outStream.writeObject(filename);
                        outStream.flush();
```

```
                int answ;
                long date;
                if((answ = inStream.readShort()) == 1){ //DIRETORIO
                        pull_rep(filename, inStream, outStream);

                }
                else if(answ == 0){
                        if(inStream.readShort() == -1){
                                System.out.println("Erro: Nao tem acesso a esse
diretorio/ficheiro");
                                return;
                        }
                        date = inStream.readLong();
                        if(lastModified(filename) < date){
                                msg.confirm(outStream);
                                File file = msg.receiveFile(filename,inStream);
                                file.setLastModified(date);
                                System.out.println("-- O ficheiro " +
filename.split("/")[1] +
                                                " foi copiado do servidor");
                        }
                        else{
                                System.out.println("-- O ficheiro " +
                                                filename.split("/")[1] + " encountra se
actualizado");
                                outStream.writeShort(-1);
                        }
                }
                else
                        System.out.println("Erro: Ficheiro nao encontrado");
        } catch (IOException e) {
                e.printStackTrace();
        } catch (ClassNotFoundException e) {
```

```
                e.printStackTrace();
            }
        }

        /**
         * Funcao que faz pull de todos os ficheiros que se encontram num repositorio do servidor
         * para o repositorio local com o mesmo nome
         * @param filename - nome no repositorio ao qual vamos fazer pull
         * @param inStream - objeto por onde le ao servidor
         * @param outStream - objeto por onde escreve ao servidor
         * @throws IOException
         * @throws ClassNotFoundException
         */
        public void pull_rep(String filename,ObjectInputStream inStream,
                ObjectOutputStream outStream) throws IOException, ClassNotFoundException{
            if(inStream.readShort() == -1){
                System.out.println("Erro: nao tem acesso a esse diretorio");
                return;
            }

            boolean ourRep = inStream.readBoolean();
            if(!ourRep && !new File(filename).exists())
                System.out.printf("-- O repositorio %s do utilizador %s foi"
                                + " copiado do servidor\n",filename.split("/")[0],username);
            long date;
            int total = 0,size = inStream.readInt();
            String fl,totalName;
            File file;
            final ArrayList<String> recNames = new ArrayList<String>();
            for(int i = 0; i < size; i++){
```

```
                    fl = (String) inStream.readObject();
                    totalName = getTotalName(filename,fl);
                    recNames.add(totalName.split("/")[totalName.split("/").length -1]);
                    date = inStream.readLong();
                    if(lastModified(totalName) < date){
                            msg.confirm(outStream);
                            file = msg.receiveFile(totalName, inStream);
                            file.setLastModified(date);
                            if(ourRep)
                                    System.out.println("-- Copiamos o ficheiro " + fl + " do
servidor");
                            total ++;
                    }
                    else{
                            msg.reject(outStream);
                    }
            }
            size = inStream.readInt();
            for(int i = 0; i < size; i ++){
                    String name = (String) inStream.readObject();
                    totalName = getTotalName(filename,name);
                    if(new File(totalName).exists()){
                            System.out.println("-- O ficheiro " + name + " existe"
                                            + " localmente mas foi eliminado do servidor");
                            total++;
                    }
            }
            if(total == 0)
                    System.out.println("-- Nenhuma alteracao a informar");
    }

    /**
     * Funcao que devolve o nome completo, ou seja, o diretorio de um ficheiro
```

```
	 * @param filename - nome do ficheiro ao qual vamos depois devolver o diretorio completo
	 * @param fl
	 * @return o diretorio do ficheiro com nome filename
	 */
	private String getTotalName(String filename,String fl){
		if(filename.split("/").length == 1){
			new File(filename).mkdir();
			return filename + "/" + fl;
		}
		if(filename.split("/").length == 2){
			String[] folderNames = filename.split("/");
			if(folderNames[0].equals(username)){
				new File(filename).mkdir();
				return folderNames[1] +"/"+fl;
			}

			new File(folderNames[0]).mkdir();
			if(!new File(filename).exists()){
				System.out.printf("-- Vamos copiar o diretorio %s "
						+ "do utilizador %s.\n",folderNames[1],folderNames[0]);
			}
			new File(filename).mkdirs();
			return filename+"/"+fl;

		}
		else //tamanho 3
				return filename;
	}

	/**
	 * Funcao que vai permitir um utilizador partilhar o seu repositorio com um outro
```

```
	 * utilizador
	 * @param outStream - objeto por onde escreve ao servidor
	 * @param myRep - nome do repositorio que vai ser partilhado
	 * @param userTo - nome do utilizador nao criador do repositorio que vai ganhar acesso ao
	 * dito repositorio
	 * @param inStream - objeto por onde le ao servidor
	 */
	public void share(ObjectOutputStream outStream,String myRep,String userTo,
			ObjectInputStream inStream){
		try {
			outStream.writeInt(SHARE);

			outStream.writeObject(myRep);
			outStream.writeObject(userTo);
			if(inStream.readShort() != 1){
				System.out.println("Erro: Nao pode partilhar com o proprio utilizador");
				return;
			}
			outStream.flush();
			if(inStream.readInt() == 1){
				int ans;
				if((ans = inStream.readInt()) == 1)
					System.out.println("--  O repositorio "+myRep+" foi "
							+ "partilhado com o utilizador " + userTo);
				else if(ans == 0)
					System.out.printf("Erro: Utilizador %s ja tinha acesso"
							+ " ao repositorio %s\n",userTo,myRep);
				else
					System.out.println("Erro: Ocorreu um erro");
			}
```

```
                        else
                                System.out.println("Erro: O user "+userTo+" nao existe");
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        /**
         * Funcao que permite retirar as permissoes de acesso de um utilizador sobre um
         * repositorio
         * @param outStream - obejto por onde escreve ao servidor
         * @param myRep - nome do repositorio ao qual vai retirar as permissoes de acesso
         * @param userTo - nome do utilizador nao criador que vai perder as permissoes de
acesso
         * @param inStream - objeto por onde le ao servidor
         */
        public void remove(ObjectOutputStream outStream,String myRep,
                        String userTo,ObjectInputStream inStream){
                try {
                        outStream.writeInt(REMOVE);
                        outStream.writeObject(myRep);
                        outStream.writeObject(userTo);
                        outStream.flush();
                        if(inStream.readInt() == 1){
                                int ans;
                                if((ans = inStream.readInt()) == 1)
                                        System.out.println("--  O utilizador "+userTo+" foi "
                                                        + "removido doo repositorio " +
myRep);
                                else if(ans == 0)
                                        System.out.printf("Erro: Utilizador %s nao tinha
acesso"
                                                        + " ao repositorio
%s\n",userTo,myRep);
```

```
                                else
                                        System.out.println("Erro: Ocorreu um erro ao fazer o
remove");
                        }
                        else
                                System.out.println("Erro: O user " +userTo + " nao existe");
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        /**
         * Funcao que indica a data da ultima vez que um ficheiro foi alterado
         * @param fileName - nome do ficheiro ao qual vamos avaliar
         * @return qual foi a ultima vez que um ficheiro foi alterado, -1 caso nao exista
         */
        private long lastModified(String fileName){
                File file = new File(fileName);
                return file.exists() ? file.lastModified() : -1;
        }



}
```

Messager:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.nio.ByteBuffer;
```

```
public class Messager {

	private static final int SIZE = 1024;

	/**
	 * Funcao que trata do envio do ficheiro
	 * @param file - ficheiro que vai ser enviado
	 * @param outStream - objeto por onde escreve ao recetor
	 * @throws IOException
	 */
	public void sendFile(File file, ObjectOutputStream outStream) throws IOException{
		FileInputStream fp = new FileInputStream(file.getPath());
		byte[] aEnviar = new byte[SIZE];
		outStream.write(ByteBuffer.allocate(4).putInt((int)file.length()).array(),0,4);
		//passar o tamanho total
		int n;
		while((n=fp.read(aEnviar,0,SIZE))>=0){
			outStream.write(aEnviar,0,n);
		}
		outStream.flush();
		fp.close();
	}

	/**
	 * Funcao que trata de rececao de um ficheiro
	 * Recebemos fileSize, e os bytes do ficheiro, 1024 de cada vez
	 * @param: fileName - Nome do ficheiro que vai ser recebido
	 * @throws IOException
	 */
	public File receiveFile(String fileName,ObjectInputStream inStream) throws
IOException{
		byte[] by = new byte[4];
```

```
        byte[] fileBytes = new byte[SIZE];
        File file = new File(fileName);
        FileOutputStream fos = new FileOutputStream(fileName);
        inStream.read(by,0,4);
        int fileSize = ByteBuffer.wrap(by).getInt();
        int n;
        while(file.length()< fileSize){
                n = inStream.read(fileBytes, 0, 1024);
                fos.write(fileBytes, 0, n);
        }
        fos.close();
        return file;
}

/**
 * Funcao que trata do push do ficheiro
 * @param file - ficheiro ao qual vamos fazer o push
 * @param outStream - objeto por onde escreve ao recetor
 * @param inStream - objeto por onde le ao recetor
 * @return true se o push foi feito com sucesso; false caso contrario
 * @throws IOException
 */
public boolean basic_push(File file,
                ObjectOutputStream outStream,
                ObjectInputStream inStream) throws IOException{
        outStream.writeLong(file.lastModified());
        outStream.flush();
        if(inStream.readShort() == 1){
                sendFile(file, outStream);
                return true;
        }
```

```
                else
                        return false;
        }

        /**
         * Funcao que confirma a rececao do ficheiro
         * @param outStream - obejto por onde escreve ao recetor
         */
        public void confirm(ObjectOutputStream outStream){
                try {
                        outStream.writeShort(1);
                        outStream.flush();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        /**
         * Funcao que rejeita a rececao do ficheiro
         * @param outStream - objeto por onde escreve ao recetor
         */
        public void reject(ObjectOutputStream outStream){
                try {
                        outStream.writeShort(-1);
                        outStream.flush();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

myGit:

```
import java.io.File;
```

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Scanner;



public class myGit{

	private static int ped;
	private static boolean missingPass = false;
	private static String username;
	private static String serverAddress;
	private static String password;
	private static String filename;
	private static String userTo;

	private final static int INIT = 0;
	private final static int ADD_USER = 10;
	private final static int PUSH = 20;
	private final static int PULL = 30;
	private final static int SHARE = 40;
	private final static int REMOVE = 50;

	public static void main(String[] args) throws FileNotFoundException {
		ped = isValid(args);

		myGit client = new myGit();
		client.startClient();
	}
```

```
/**
 * Funcao que inicia o cliente
 */
public void startClient (){
	ClientThread newClientThread = new ClientThread();
	newClientThread.start();
}

/**
 * Funcao que trata de ligacao do cliente com o servidor
 * @param ServerAddress - o endereco do servidor ao qual vai fazer a ligacao
 * @return a socket da ligacao estabelecida
 */
public Socket startConnection(String ServerAddress){
	Socket sSoc = null;
	String srvAdrs[] = ServerAddress.split(":");
	try {
		sSoc = new Socket(srvAdrs[0],Integer.parseInt(srvAdrs[1]));
	} catch (IOException e) {
		System.out.println("Erro: Servidor nao encontrado!");
		System.exit(-1);
	}
	return sSoc;
}

//Threads utilizadas para comunicacao com o servidor
class ClientThread extends Thread {

	private Socket socket = null;

	/**
```

```
     * Trata da iteracao entre o cliente e o servidor
     */
    public void run(){
        try {

            if(ped == INIT)
                init(filename);
            else if(ped == -1)
                System.out.println("Erro: comando nao reconhecido");
            else{

                socket = startConnection(serverAddress);

                ObjectOutputStream outStream = new
ObjectOutputStream(socket.getOutputStream());
                ObjectInputStream inStream = new
ObjectInputStream(socket.getInputStream());
                Scanner reader = new Scanner(System.in);
                if(missingPass){
                    System.out.println("-- Porfavor diga a
password");
                    password = reader.nextLine();
                }
                ClientServerHandler csh = new
ClientServerHandler(username,password);

                //enviar logo aquilo que vai enviar obrigatoriamente
                csh.sendInitInfo(outStream);
                //preciso de fazer o log in
                int addUs;
                if((addUs = csh.addUser(outStream, inStream)) == -1)
                    return;

                switch(ped){
```

```
                                        case ADD_USER:
                                                if(addUs == 0)
                                                        System.out.println("-- O utilizador " +
username
                                                                        + " ja existe");
                                                outStream.writeInt(-1);
                                                outStream.flush();
                                                break;
                                        case PUSH:
                                                File file = new File(filename);
                                                if(file.exists()){
                                                        if(file.isDirectory())

        csh.push_rep(file,filename,outStream, inStream);
                                                        else{
                                                                if(filename.split("/").length ==
1){
                                                                        outStream.writeInt(-1);

        System.out.println("ERRO: Tem de especificar o diretorio");
                                                                        break;
                                                                }
                                                                csh.push_file(file,filename,
outStream, inStream);
                                                        }
                                                }else{
                                                        System.out.println("Eroo: Esse ficheiro
nao existe");
                                                        outStream.writeInt(-1);
                                                }

                                                break;
                                        case PULL:
                                                csh.pull(filename, outStream, inStream);
```

```
                                        break;
                                    case SHARE:
                                        csh.share(outStream, filename, userTo,
inStream);
                                        break;
                                    case REMOVE:
                                        csh.remove(outStream, filename, userTo,
inStream);
                                        break;
                                    }

                                    reader.close();
                                    outStream.close();
                                    inStream.close();
                                    socket.close();
                                }

                            } catch (IOException e) {
                                System.out.println("Erro:nao se conseguiu conectar ao
servidor");
                            }
                        }
                    }

                    /**
                     * Funcao que verifica se os argumentos inseridos sao validos
                     * @param arg - os argumentos que vao ser avaliados
                     * @return
                     */
                    public static int isValid(String[] arg){
                        int min = 0,tam = arg.length;
                        if(tam < 2 || tam > 8)
                            return -1;
```

```
if(arg[0].equals("-init") && tam == 2){
        filename = arg[1];
        return INIT;
}
username = arg[0];
serverAddress = arg[1];
if(tam == 2){
        missingPass = true;
        return ADD_USER;
}
else if (tam >=4)
        password = arg[3];
else
        return -1;
if(tam >= 4){
        if(!arg[2].equals("-p")){
                missingPass = true;
                min = 2;
        }
        if(tam + min == 4)
                return ADD_USER;
        if(tam + min == 6){
                filename = arg[5 - min];
                if(arg[4 - min].equals("-push"))
                        return PUSH;
                else if(arg[4 - min].equals("-pull"))
                        return PULL;
        }
        else if(tam + min == 7){
                filename = arg[5 - min];
                userTo = arg[6 - min];
                if(arg[4 - min].equals("-share"))
```

```
                                        return SHARE;
                                else if(arg[4 - min].equals("-remove"))
                                        return REMOVE;
                        }
                }
                return -1;
        }

        /**
         * Funcao que trata da criacao de um repositorio local
         * @param folderName - nome do repositorio a ser criado
         */
        public static void init(String folderName){
                if(new File(folderName).mkdirs())
                        System.out.println("-- O repositorio " + folderName + " foi criado
localmente");
                else
                        System.out.println("-- Erro na criacao do " + folderName);
        }
}
```

ServerClientHandler:

```
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Arrays;

public class ServerClientHandler {

        private String username;
```

```
private String passwd;
private Messager msg;

public ServerClientHandler(String username,String passwd){
	this.username = username;
	this.passwd = passwd;
	msg = new Messager();
}

/**
 * Adiciona um utilizador a uma lista de utilizadores que se encontra no servidor
 * @param users - lista de utilizadores onde se vai adicionar o novo
 * @param outStream - objeto por onde escreve ao servidor
 * @param inStream  - objeto por onde le ao servidor
 */
public boolean AddUser(userCatalog users,ObjectOutputStream outStream,
		ObjectInputStream inStream){
	try {
		if(!users.existsUser(username)){
			msg.confirm(outStream);
			if(inStream.readShort() == 1){ //LER, confirmado pelo util
				users.addUser(username, passwd);
				msg.confirm(outStream);
				return true;
			}else{
				return false;
			}
		//falta me confirmar
		}else{
			msg.reject(outStream);
			if(!users.login(username, passwd)){//nao consegui fazer login
				msg.reject(outStream);
```

```
                        return false;
                    }
                    msg.confirm(outStream);
                    return true;
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
            return false;
            //ou dizer que deu erro
}

/**
 * Funcao que apos receber o pedido de push de um repositorio, faz o push de todos os
 * ficheiros que se encontram do repositorio local para o repositorio do servidor com
 * o mesmo nome
 * @param inStream - objeto por onde le ao servidor
 * @param outStream - objeto por onde escreve ao servidor
 * @param reps - lista dos repositorios que se encontram no servidor
 */
public void push_rep(ObjectInputStream inStream,ObjectOutputStream outStream,
                repCatalog reps){
        try {
                String repname = fullNameRep((String) inStream.readObject());
                System.out.println("repname: " + repname);

                if(!new File(repname).exists() && isCreator(repname)){
                    reps.addRep(repname, username);
                    msg.confirm(outStream);
                }
                else if(!new File(repname).exists() && !isCreator(repname)){
                    outStream.writeShort(0);
```

```
                return;
        }
        else
                msg.reject(outStream); //ja exisita

        if(!hasAccess(reps, repname, username, outStream))
                        return;
        int size = inStream.readInt(); //receber o num dos ficheiros

        final ArrayList<File> allAddedFiles = new ArrayList<File>();
        for(int i = 0; i < size; i++){
                outStream.flush();
                String filename = (String) inStream.readObject();
                long date = inStream.readLong();
                String totalName = repname + "/" +filename.split("/")[1];
                File file = new File(totalName);
                allAddedFiles.add(file);
                if(file.lastModified() < date){
                        msg.confirm(outStream);
                        addHist(totalName);
                        file = msg.receiveFile(totalName, inStream);
                        file.setLastModified(date);
                }
                else
                        outStream.writeShort(-1);
        }
        //so falta avizar
        File[] files = new File(repname).listFiles( new FileFilter(){
                @Override
           public boolean accept(File pathname) {
                        char lastChar = pathname.getName().charAt(
                                        (int) (pathname.getName().length() -
1));
```

```
                        return !allAddedFiles.contains(pathname) &&
                                !Character.isDigit(lastChar);
                    }
                });
                outStream.writeShort(files.length);
                String totalName;
                for(File fl:files){
                        outStream.writeObject(fl.getName());
                        totalName = repname +"/"+fl.getName();
                        addHist(totalName); //tenho que testar
                        fl.delete();
                }
        } catch (ClassNotFoundException | IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}


/**
 * Funcao que trata de fazer o push de um ficheiro para um repositorio especifico
 * no servidor
 * @param reps - lista de repositorios
 * @param inStream - objeto por onde escreve ao servidor
 * @param outStream - objeto por onde le ao servidor
 */
public void push_file(repCatalog reps,ObjectInputStream inStream,
                ObjectOutputStream outStream){
        try {
                String filename = (String) inStream.readObject();
                System.out.println("FileName: " + filename);
                String fullName = fullNameFile(filename);
```

```
                    System.out.println("FullName: " +fullName);
                    File file = new File(fullName);



                    if(!hasAccess(reps, fullName, username, outStream))
                            return;

                    long date = inStream.readLong();
                    if(file.lastModified() < date){
                            msg.confirm(outStream);
                            addHist(fullName);
                            file = msg.receiveFile(fullName,inStream);
                            file.setLastModified(date);
                    }
                    else
                            msg.reject(outStream);
            } catch (ClassNotFoundException | IOException e) {
                    e.printStackTrace();
            }
    }

    /**
     * Funcao de trata de fazer pull de um ficheiro que se encontra num repositorio no
servidor
     * para o repositorio local com o mesmo nome
     * @param rep - nome do repositorio de onde se vai fazer o pull dos seu ficheiros
     * @param outStream - objeto por onde escreve ao servidor
     * @param inStream - objeto por onde le ao servidor
     */
    public void pull(repCatalog rep,ObjectOutputStream outStream,
                    ObjectInputStream inStream){
            String filename;
            File file;
```

```
try {
        filename = (String) inStream.readObject();//nome

        String totalName;
        if(filename.split("/").length == 1)
                totalName = username + "/" + filename;
        else if(filename.split("/").length == 2){
                File test= new File(username +"/"+filename);
                if(test.exists() && test.isFile())
                        totalName = username + "/" + filename;
                else
                        totalName = filename;
        }
        else
                totalName = filename;

        file = new File(totalName);
        //ver se esta em formato folder
        if(file.isDirectory()){ //ver se eh diretoria
                System.out.println("EH DIRETORIA");
                outStream.writeShort(1);//dir
                pull_rep(file, totalName, rep, outStream, inStream);
        }
        //ver se esta em formato file
        else if(file.isFile()){
                outStream.writeShort(0); //confirmar
                if(!hasAccess(rep, totalName, username, outStream)){
                        return;
                }
                msg.basic_push(file,outStream, inStream);
        }
        else
```

```
                        outStream.writeShort(-1);
            } catch (ClassNotFoundException e) {
                        e.printStackTrace();
            } catch (IOException e) {
                        e.printStackTrace();
            }
}

/**
 *
 * Funcao de trata de fazer pull de todos os ficheiros que se encontram num repositorio
 * especifico no servidor para o repositorio local com o mesmo nome
 * @param file - o repositorio onde vamos buscar os ficheiros
 * @param totalName - nome do diretorio do repositorio
 * @param rep - lista de repositorios que se encontram no servidor
 * @param outStream - objeto por onde escreve ao servidor
 * @param inStream - objeto por onde le ao servidor
 */
public void pull_rep(File file,String totalName,repCatalog rep,
                        ObjectOutputStream outStream,ObjectInputStream inStream){
            if(!hasAccess(rep, totalName, username, outStream)){
                        return;
            }
            File[] files = file.listFiles( new FileFilter(){
                        @Override
                  public boolean accept(File pathname) {
                                    char lastChar = pathname.getName().charAt(
                                                pathname.getName().length() - 1);
                        return !Character.isDigit(lastChar);
                  }
            }); //nao ha subdir
            try {
```

```
if(totalName.split("/")[0].equals(username))
        outStream.writeBoolean(true); //eh o nosso util a fazer o push?
else
        outStream.writeBoolean(false);
outStream.writeInt(files.length);
final ArrayList<String> sendFiles = new ArrayList<>();
for(File fl:files){
        outStream.writeObject(fl.getName()); //enviar o nome
        msg.basic_push(fl,outStream, inStream);
        sendFiles.add(fl.getName());
}
//Envio o nome do primeiro historico dos que nao foram enviados

File[] histFiles = file.listFiles( new FileFilter(){
        @Override
   public boolean accept(File pathname) {
                char lastChar = pathname.getName().charAt(
                        (int) (pathname.getName().length() -
1));
                String[] allDots = pathname.getName().split("\\.");
                System.out.println(pathname.getName());
                String fileActualName = allDots[0] + "." + allDots[1];
      return lastChar == '1' && !sendFiles.contains(fileActualName);
   }
});

outStream.writeInt(histFiles.length);
for(File fl : histFiles){
        String[] allDots = fl.getName().split("\\.");
        String fileActualName = allDots[0] + "." + allDots[1];
        outStream.writeObject(fileActualName);
}
```

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

/**
 * Funcao que trata da organizacao do acesso de um utilizador a um repositorio
 * @param outStream - objeto por onde escreve ao servidor
 * @param inStream - objeto por onde le ao servidor
 * @param reps - lista de repositorios que se encontram no servidor
 * @param users - lista de utilizadores que se econtram no servidor
 */
public void share(ObjectOutputStream outStream,
			ObjectInputStream inStream,repCatalog reps,userCatalog users){
		try {
			String myRep = (String) inStream.readObject();
			String userTo = (String) inStream.readObject();
			if(userTo.equals(username)){
				msg.reject(outStream);
				return;
			}
			else
				msg.confirm(outStream);

			if(users.existsUser(userTo)){
				outStream.writeInt(1); //first confirm
				if(new File(username + "/" + myRep).exists() &&
						reps.isCreator(username, myRep)){
					if(reps.addUser(username, myRep, userTo))
						outStream.writeInt(0);
					else
						outStream.writeInt(1);
```

```
                }
                else
                        outStream.writeInt(-1);
        }else
                outStream.writeInt(-1);

    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Funcao que trata de remover a permissao de acesso de um utilizador a um repositorio
 * especifico
 * @param outStream - objeto por onde escreve ao servidor
 * @param inStream - objeto por onde le ao servidor
 * @param reps - lista de repositorios que se encontram no servidor
 * @param users - lista de utilizadores que se encontram no servidor
 */
public void remove(ObjectOutputStream outStream,
        ObjectInputStream inStream,repCatalog reps,userCatalog users){
    System.out.println("Vamos fazer remove");
    try {
        String myRep = (String) inStream.readObject();
        String userTo = (String) inStream.readObject();
        if(users.existsUser(userTo)){
            outStream.writeInt(1);
            if(new File(username + "/" + myRep).exists() &&
                    reps.isCreator(username, myRep)){
                if(reps.removeUser(username, userTo, myRep))
```

```
                                        outStream.writeInt(1);
                                else
                                        outStream.writeInt(0);
                        }
                        else
                                outStream.writeInt(-1);
                }else
                        outStream.writeInt(-1);

        } catch (ClassNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
}

/**
 * Funcao que trata do historico de um ficheiro
 * @param filename - nome do ficheiro onde vamos tratar do historico
 */
public void addHist(String filename){
        if(new File(filename).exists()){
                boolean found = false;
                for(int i = 1 ;!found;i++ ){
                        File file = new File(filename + "." + i);
                        if(!file.exists()){
                                new File(filename).renameTo(file);
                                found = true;
                        }
                }
        }
```

```
}

/**
 * Funcao que devolve a diretoria de um ficheiro
 * @param filename - nome do ficheiro onde vamos descobrir o historico
 * @return o diretorio do ficheiro com nome filename
 */
private String fullNameFile(String filename){
	String[] allFiles = filename.split("/");
	if(allFiles.length == 1)
		return filename;
	else if(allFiles.length == 2){

		return username + "/" +filename;
	}
	else
		return filename;
}

//Se o ficheiro for um folder
/**
 * Funcao que devolve o diretorio de um repositorio
 * @param filename - nome do repositorio
 * @return o diretorio do repositorio
 */
private String fullNameRep(String filename){
	String[] allFiles = filename.split("/");

	System.out.println("InFullNameRepFunc arg0 = " + filename);

	if( allFiles.length == 1 ){
		return username + "/" + filename;
```

```
                }
                else if(allFiles.length == 2){
                        return filename;
                }
                else
                        return filename;
        }

        /**
         * Funcao que indica se um utilizador tem acesso a um repositorio especifico
         * @param reps - lista de repositorios que se encontram no servidor
         * @param fullName - diretorio do repositorio
         * @param username - nome do utilizador que vamos avaliar
         * @param outStream - objeto por onde escreve ao servidor
         * @return true se o utilizador username tem acesso ao repositorio com o diretorio
fullName;
         * false caso contrario
         */
        private boolean hasAccess(repCatalog reps,String fullName,
                        String username,ObjectOutputStream outStream){
                String[] folderNames = fullName.split("/");
                try {

                        if(fullName.split("/")[1].equals("users.txt") ||
                                        fullName.split("/")[1].equals("..")){
                                msg.reject(outStream);
                                return false;
                        }
                        if(isCreator(fullName)){
                                msg.confirm(outStream);
                                return true;
                        }
                        if(!reps.hasAccess(folderNames[0] +"/"+ folderNames[1], username)){
```

```
                        msg.reject(outStream);
                        return false;
                }
                msg.confirm(outStream);
                return true;
        } catch (IOException e) {
                e.printStackTrace();
        }
        return false;
    }

    /**
     * Funcao que indica se um utilizador eh criador de um repositorio
     * @param fullNameFile - diretorio do repositorio
     * @return true se o utilizador username eh criador do repositorio com o diretorio
     * fullNameFile; false caso contrario
     */
    private boolean isCreator(String fullNameFile){
        return username.equals(fullNameFile.split("/")[0]);
    }
}
```

repCatalog:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Cada utilizador vai ter no seu dir de util um ficheiro, que nao pode ser acedido, que
 * dita quem pode ver que repositorios no seu dir do genero
```

```
 *
 * <repositorio>:<util1;util2;util3>
 *
 */
public class repCatalog{

	private String fileName;

	public repCatalog(String fileName){
		this.fileName = fileName;
	}

	/**
	 * Cria um novo repositorio
	 * @param repName - nome do repositorio a ser criado
	 * @param Creator - nome do criador do repositorio a ser criado
	 * @throws IOException
	 */
	public void addRep(String repName,String Creator) throws IOException{
		new File(repName).mkdir();
		new File(Creator + "/" + fileName).createNewFile();

	}

	/**
	 * Adiciona um utilizador ah lista daqueles que o criador partilha o seu repositorio
	 * @param sharer - Nome do criador quer partilhar o seu repositorio
	 * @param repName - Nome do repositorio que o criador quer partilhar
	 * @param userToShare - O nome do utilizador a partilhar com
	 * @throws IOException
	 */
	public boolean addUser(String sharer,String repName,
```

```
                String userToShare) throws IOException{
        return addUserToFile(sharer,userToShare,repName);
    }

    /**
     * Funcao auxiliar do addUser
     * @param sharer - Nome do criador quer partilhar o seu repositorio
     * @param userToShare - O nome do utilizador a partilhar com
     * @param repName - Nome do repositorio que o criador quer partilhar
     * @return true se a funcao foi feit com sucesso; false caso contrario
     */
    private boolean addUserToFile(String sharer,String userToShare,String repName){
        File file = new File(sharer + "/"  +fileName);
        File tempFile = new File(sharer + "/myTempFile.txt");
        try {
            BufferedReader reader = new BufferedReader(new FileReader(file));
            BufferedWriter writer = new BufferedWriter(new
FileWriter(tempFile));
            boolean isArleadyThere = false,foundRep = false;
            String currentLine;
            String[] line,users;
            StringBuilder userLine = new StringBuilder();
            while((currentLine = reader.readLine()) != null) {
                line = currentLine.split(":");
                if(line[0].equals(repName)){
                    foundRep = true;
                    users = line[1].split(";");
                    for(String user:users){
                        if(user.equals(userToShare))
                            isArleadyThere = true;
                        userLine.append(user + ";");
                    }
                    if(!isArleadyThere)
```

```
                                        userLine.append(userToShare+";");
                                    userLine.deleteCharAt(userLine.length() - 1);
                                    writer.write(repName +":"+userLine.toString()+
                                                System.getProperty("line.separator"));
                                }
                                else
                                    writer.write(currentLine +
System.getProperty("line.separator"));
                            }
                            if(!foundRep)
                                writer.write(repName +":"+userToShare +
System.getProperty("line.separator"));
                            writer.close();
                            reader.close();
                            if(!file.delete())
                              System.out.println("Could not delete file");
                             //Rename the new file to the filename the original file had.
                            if (!tempFile.renameTo(file))
                                System.out.println("Could not rename file");

                            return isArleadyThere;

                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        return false;
                    }

                    /**
                     * Funcao que indica se um utilizador eh criador de um dado repositorio
                     * @param sharer - nome de utilizador a avaliar
                     * @param repName - nome do repositorio a ser utilizado para fazer a verificacao
                     * @return true se sharer eh criador do repositorio repName; false caso contario
```

```
     */
    public boolean isCreator(String sharer,String repName){
            return new File(sharer + "/" + repName).exists();
    }

    /**
     * Funcao que remove um utilizador da lista daqueles que o criador quer
     * partilhar o seu repositorio
     * @param repName - nome do repositorio ao qual o utilizador vai ser removido
     * @param remover - nome do criador do repositorio
     * @param user - nome do utilizador a ser removido
     */
     public boolean  removeUser(String sharer,String userToRemove,
            String repName) throws IOException{
              File file = new File(sharer + "/"  +fileName);
                    File tempFile = new File(sharer + "/myTempFile.txt");
                    try {
                            boolean wasThere = false;
                            BufferedReader reader = new BufferedReader(new
FileReader(file));
                            BufferedWriter writer = new BufferedWriter(new
FileWriter(tempFile));
                            String currentLine;
                            String[] line,users;
                            StringBuilder userLine = new StringBuilder();
                            while((currentLine = reader.readLine()) != null) {
                                    line = currentLine.split(":");
                                    if(line[0].equals(repName)){
                                            users = line[1].split(";");
                                            for(String user:users){
                                                    if(user.equals(userToRemove)) {
                                                            wasThere = true;
                                                    } else
```

```
                                                    userLine.append(user + ";");
                                                }
                                                if(userLine.length() > 0){

        userLine.deleteCharAt(userLine.length() - 1);
                                                    writer.write(repName
+":"+userLine.toString()+

        System.getProperty("line.separator"));
                                                }
                                            }
                                            else
                                                writer.write(currentLine +
System.getProperty("line.separator"));
                                        }
                                        writer.close();
                                        reader.close();
                                        if(!file.delete())
                                      System.out.println("Could not delete file");

                                    //Rename the new file to the filename the original file had.
                                    if (!tempFile.renameTo(file))
                                      System.out.println("Could not rename file");

                                    return wasThere;

                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                                return false;
                    }

                /**
```

```
 * Funcao que indica se um certo utilizador tem acesso a um dado repositorio
 * @param repName - nome do repositorio
 * @param userToLook - nome do utilizador que vamos avaliar
 * @return true se userToLook tem acesso ao repName; false caso contrario
 * @throws IOException
 * @requires File repName exists
 */
public boolean hasAccess(String repName, String userToLook) throws IOException{
        System.out.println("Entrou no hasAcess");
        String username = repName.split("/")[0];
        System.out.println("repName: " + repName);
        File file = new File(username+"/"+fileName);
        return hasAccess(file,repName,userToLook);
}

/**
 * Funcao auxiliar de hasAccess
 * @param file - nome do ficheiro que se encontra no repositorio
 * @param repName - nome do repositorio
 * @param user - nome do utilizador a ser avaliado
 * @return true se user tem acesso a repName; false caso contrario
 */
private boolean hasAccess(File file,String repName,String user){
        try {
                System.out.println("repName: " + repName);
                FileReader fr = new FileReader(file);
                BufferedReader br = new BufferedReader(fr);
                String line;
                while((line = br.readLine()) != null){
                        System.out.println(line);
                        String[] folderNames = repName.split("/");
                        if(line.split(":")[0].equals(folderNames[1]) ){
```

```
                                        String[] users = line.split(":")[1].split(";");
                                        for(String fileUser:users){
                                                if(fileUser.equals(user)){
                                                        fr.close();
                                                        br.close();
                                                        return true;
                                                }
                                        }
                                }
                        }
                        fr.close();
                        br.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return false;
        }

        /**
         * Funcao que indica a data da ultima vez que o ficheiro foi alterado
         * @param user - nome do criador do repositorio
         * @param myRep - nome do repositorio onde se encontra o ficheiro
         * @param file - nome do ficheiro que queremos verificar a data
         * @return a data em que o ficheiro foi modificado, -1 caso nao exista
         */
        public long lastModified(String user,String myRep,String file){
                File fl = new File(user+"/" +myRep + "/" + file);
                return fl.exists() ? fl.lastModified() : -1;
        }

}
```

userCatalog:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.IOException;
/**
 * Ideia, usar o ficheiro como o catalogo de utilizadores
 * Podemos usar User U ou
 * String username && String password
 * @author Utilizador
 */
public class userCatalog {
	private static File passwords;

	public userCatalog(){
		passwords = new File("passwords.txt");
		try {
			passwords.createNewFile();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	/**
	 * Adiciona um utilizador ao ficheiro passwords
	 * @param username - nome do utilizador a adicionar
	 * @param password - password do utililizador a adicionar
	 * @throws IOException
	 */
```

```
public void addUser(String username, String password){
	StringBuilder Stb = new StringBuilder();
	Stb.append(username + ":" + password + System.getProperty("line.separator"));
	if(existsUser(username))
		return;
	try {
		FileWriter fw = new FileWriter(passwords,true);
		fw.write(Stb.toString());
		new File(username).mkdir(); //create user folder
		fw.close();
	} catch (IOException e1) {
		e1.printStackTrace();
	}
}

/**
 * Funcao que indica se existe um utilizador
 * @param username - nome do utilizador que vamos avaliar
 * @return true se o utilizador username existe; false caso contrario
 * @throws IOException
 */
public boolean existsUser(String username){
	BufferedReader reader;
	try {
		reader = new BufferedReader(new FileReader(passwords));
		String line;
		while((line = reader.readLine()) != null){
			if(line.split(":")[0].equals(username)){
				reader.close();
				return true;
			}
		}
```

```
                reader.close();
        } catch (IOException e) {
                e.printStackTrace();
        }

        return false;
}

/**
 * Funcao que trata do login do utilizador
 * @param username - nome do utilizador
 * @param passwordGiven - password do utilizador
 * @return true se o login foi feito com sucesso; false caso contrario
 */
public boolean login(String username,String passwordGiven){
        BufferedReader reader;
        try {
                reader = new BufferedReader(new FileReader((passwords)));
                String line;
                while((line = reader.readLine()) != null){
                        if(line.split(":")[0].equals(username)){
                                if(line.split(":")[1].equals(passwordGiven)){
                                        reader.close();
                                        return true;
                                }
                                return false;
                        }
                }
                reader.close();
        } catch (IOException e) {
                        e.printStackTrace();
        }
```

```
                return false;
        }
}
```

myGitServer:

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

//Servidor myServer

public class myGitServer{

        private repCatalog reps = new repCatalog("users.txt");
        private userCatalog users = new userCatalog();

        private final static int ADD_USER = 10;
        private final static int PUSH_REP = 20;
        private final static int PUSH_FILE = 30;
        private static final int PULL = 40;
        private final static int SHARE = 50;
        private final static int REMOVE = 60;

        public static void main(String[] args) {
                System.out.println("servidor: main");
                myGitServer server = new myGitServer();
                if(args.length < 1 || args.length > 1){
                        System.out.println("Erro: Criacao do servidor so recebe o porto");
                        return;
                }
```

```
		int serverPort = Integer.parseInt(args[0]);
		server.startServer(serverPort);
	}

	/**
	 * Funcao que inicia o servidor
	 * @param serverSocket - socket pelo qual e feito a ligacao com o servidor
	 */
	public void startServer (int serverSocket){
		ServerSocket sSoc = null;

		try {
			sSoc = new ServerSocket(serverSocket);
		} catch (IOException e) {
			System.err.println(e.getMessage());
			System.exit(-1);
		}

		while(true) {
			try {
				Socket inSoc = sSoc.accept();
				ServerThread newServerThread = new ServerThread(inSoc);
				newServerThread.start();
		    }
		    catch (IOException e) {
		        e.printStackTrace();
		    }

		}
		//sSoc.close();
	}
```

```
class ServerThread extends Thread {

	private Socket socket = null;

	ServerThread(Socket inSoc) {
		socket = inSoc;
	}

	/**
	 * Trata da iteracao entre o servidor e cliente
	 */
	public void run(){

		try {
			ObjectOutputStream outStream = new
ObjectOutputStream(socket.getOutputStream());
			ObjectInputStream inStream = new
ObjectInputStream(socket.getInputStream());

			String user = (String)inStream.readObject();
			String passwd = (String)inStream.readObject();

			ServerClientHandler sch = new
ServerClientHandler(user,passwd);

			if(sch.AddUser(users, outStream, inStream)){
				System.out.println("Saiu do addUser");
				switch(inStream.readInt()){
				case ADD_USER:
					break;
				case PUSH_REP:
					sch.push_rep(inStream,outStream, reps);
					break;
```

```
                                        case PUSH_FILE:
                                                sch.push_file(reps, inStream, outStream);
                                                break;
                                        case PULL:
                                                sch.pull(reps, outStream, inStream);
                                                break;
                                        case SHARE:
                                                sch.share(outStream, inStream,reps,users);
                                                break;
                                        case REMOVE:
                                                sch.remove(outStream, inStream, reps, users);
                                                break;
                                        default:
                                                System.out.println("Comando nao
reconhecido");
                                        }
                                }

                                outStream.close();
                                inStream.close();
                                socket.close();

                        } catch (IOException e) {
                                e.printStackTrace();
                        } catch (ClassNotFoundException e) {
                                e.printStackTrace();
                        }
                }
        }
}
```