Segurança e Confiabilidade 2015-16

LISBOA

UNIVERSIDADE DE LISBOA

Ciências ULisboa

Grupo XXIII

fc41941 – Rodrigo Reis
fc41964 – Tito Oliveira
fc45773 – Jonas Ferreira

No trabalho, todos os objectivos requisitados foram concretizados sendo que a única limitação do projecto é ser permitido apenas enviar ficheiros para o servidor, que estejam na raiz do projecto e não devem ser enviados ficheiros com a extensão .msg.

O projecto está dividido em 3 pacotes client, functionality e server.

No pacote client está a classe myWhats.java que é a concretização do cliente.

No pacote server está a classe myWhatsServer.java que é a concretização do servidor.

No pacote functionality encontramos várias classes que contêm a concretização dos métodos necessários ao funcionamento do cliente e do servidor, sendo estas:

- Authentication.java – Trata a autenticação de um utilizador server sided e client sided

- Communication.java – Trata da abertura de sockets e envio de comandos para o servidor

- Configurations.java – Contém a definição de vários paths e valores necessários a troca de blocos de dados entre cliente e servidor

- Files.java – Trata das funcionalidades relacionada com o envio,recepção de ficheiros e operações com ficheiros e pastas

- Group.java – Trata das funcionalidade relacionadas com os grupos

-Message.java – Trata das funcionalidades relacionadas com as mensagens, escrita, leitura, recepção, envio e parsing das mesmas para um formato adequado

- Usage.java – Apresenta a utilização correcta do cliente

- User.java – Trata de todas as operações relacionadas com os utilizadores

A sandbox do Servidor permite que este abra uma socket que está a escuta, aceita pedidos e resolve urls para IP.
Em termos de permissões de ficheiros, é permitido a leitura, escrita e deleção dos ficheiros de credenciais e listagem de grupos. É também permitida a leitura e escrita das pastas users, groups e messages e ficheiros no seu interior.

A sandbox do Cliente permite que uma socket faça conecções e resolva urls para endereços de ip.
Em termos de permissões de ficheiros, é permitida a leitura e escrita de todos os ficheiros na raiz do projecto.

As mensagens trocadas consiste maioritariamente no uso de booleanos, para haver coordenação entre os métodos do servidor e do cliente, e em determinados casos, como na recepção das conversas, mensagens de erro e envio de flags e parâmetros, strings.

Os requisitos que esta aplicação deve conter são a Autenticação, sendo usado um sistema de username/password para garantir esta característica; Privacidade, sendo que cada utilizador só consegue ler as suas conversas, as dos grupos a que pertence e é apenas possível que este transfira ficheiros pertencentes às mesmas; Disponibilidade, que é garantida através do uso de um servidor multithread que atende vários clientes simultaneamente; Integridade, sendo que um utilizador só pode enviar mensagens em seu nome e não pode substituir ficheiros já existentes no servidor.

```java
package client;

import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

import functionality.*;

/**
 * This class represents the client myWhats
 */
public class myWhats {

    public static void main(String[] args) throws UnknownHostException,
            IOException, ClassNotFoundException {

        // Verify the giver argument number
        if (args.length < 3) {

            System.out.println("Incorrect parameters!\n");

            // Print correct usage
            Usage.printUsage();

        } else {

            Socket sock = null;

            String user = "";
            String pass = "";

            // Automatic password handling
            if (args.length >= 4) {
                user = args[0];
                pass = args[3];
            }

            // NO PASSWORD PROVIDED
            if (args.length == 3) {

                // Check if -p
                if (args[2].equals("-p")) {

                    Scanner sc = new Scanner(System.in);

                    user = args[0];

                    while (pass.length() < 2) {

                        System.out.print("Password: ");
                        pass = sc.nextLine();

                    }

                    // Close scanner
                    //sc.close();
                    //Warn user only registering or login will happen
                    System.out
                            .println("\nNOTICE: Your only arguments are your
username, "
```

```
                                + "server and password!\n"
                                + "This will only register you or try to log 
you in.\n");

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());
                        Authentication.login(in, out, user, pass);
                        out.close();
                        in.close();
                        sock.close();
                    }


                } else {

                    System.out.println("Incorrect parameters!\n");
                    Usage.printUsage();
                }

            // JUST THE PASSWORD
            } else if (args.length == 4 && args[2].equals("-p")) {
                //Warn user only registering or login will happen
                System.out
                        .println("\nNOTICE: Your only arguments are your username, 
"
                                + "server and password!\n"
                                + "This will only register you or try to log you 
in.\n");

                sock = Communication.connect(args[1]);
                if(sock != null){
                    ObjectOutputStream out = new ObjectOutputStream(
                            sock.getOutputStream());
                    ObjectInputStream in = new ObjectInputStream(
                            sock.getInputStream());
                    Authentication.login(in, out, user, pass);
                    out.close();
                    in.close();
                    sock.close();
                }


            // PASWORD AND -r (no args)
            } else if (args.length == 5) {

                if (args[4].equals("-r")) {

                    System.out.println("-r | Send all the lastest comms!");

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());
                        Authentication.login(in, out, user, pass);
                        String [] snd = {};
                        Communication.sendCommand(out, "-r", snd);

                        //Receive result
```

```java
                        String convo = (String) in.readObject();
                        System.out.println(convo);

                        out.close();
                        in.close();
                        sock.close();
                    }

                } else {

                    System.out.println("Incorrect parameters!\n");
                    Usage.printUsage();

                }

            // PASWORD AND -r ARG1
            } else if (args.length == 6) {

                if (args[4].equals("-r")) {

                    System.out.println("-r | Send all the comms with " + args[5]);

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());
                        Authentication.login(in, out, user, pass);

                        String [] snd = {args[5]};
                        Communication.sendCommand(out, "-r", snd);

                        //Receive the answer
                        String convo = (String) in.readObject();
                        System.out.println(convo);

                        out.close();
                        in.close();
                        sock.close();
                    }

                } else {

                    System.out.println("Incorrect parameters!\n");
                    Usage.printUsage();

                }

            // ALL OTHER FLAGS
            } else if (args.length == 7) {

                switch (args[4]) {

                //MESSAGE OPERATION
                case "-m":
                    System.out.println("-m | Send message \"" + args[6] + "\" to " + args[5]);

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());
                        Authentication.login(in, out, user, pass);
                        String [] snd = {args[5], args[6]};
                        Communication.sendCommand(out, "-m", snd);
                        Message.sendMessage(out, in);
                        out.close();
                        in.close();
                        sock.close();
                    }
                    break;

                //FILE OPERATION
                case "-f":

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());

                        Authentication.login(in, out, user, pass);

                        //Check if file exists
                        File file = new File(args[6]);

                        //Don't bother server if file doesn't exist localy
                        if(file.exists() && !file.isDirectory()){
                            String [] snd = {args[5], args[6]};
                            System.out.println("-f " + args[5] + " " + args[6]);
                            Communication.sendCommand(out, "-f", snd);
                            Files.sendFile(in, out, "", args[6]);

                        }else{
                            System.out.println("File " + args[6] + " not found!");
                        }

                        out.close();
                        in.close();
                        sock.close();
                    }
                    break;

                //REVIEW OPERATION
                case "-r":
                    System.out.println("-r " + args[5] + " " + args[6]);

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());
                        Authentication.login(in, out, user, pass);
                        String [] snd = {args[5], args[6]};
                        System.out.println("-r " + args[5] + " " + args[6]);
                        Communication.sendCommand(out, "-r", snd);

                        // Check if DOWNLOADS FOLDER exist
                        File downloadsFolder = new
File(Configurations.DOWNLOAD_FOLDER);
                        if (!downloadsFolder.exists()) {

                            downloadsFolder.mkdir();
                        }

                        boolean fileExists = in.readBoolean();

                        if (fileExists) {
                            Files.receiveFile(in, out, Configurations.DOWNLOAD_FOLDER, snd[1]);
                        }else{
                            System.out.println("The file you requested does not exist!");
                        }

                        out.close();
                        in.close();
                        sock.close();
                    }
                break;

                //GROUP ADD OPERATION
                case "-a":

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());
                        Authentication.login(in, out, user, pass);
                        System.out.println("-a " + args[5] + " " + args[6]);
                        String [] snd = {args[5], args[6]};
                        Communication.sendCommand(out, "-a", snd);

                        //receive response
                        String rsp = (String) in.readObject();
                        System.out.println(rsp);

                        out.close();
                        in.close();
                        sock.close();
                    }
                    break;

                //GROUP REMOVE OPERATION
                case "-d":

                    sock = Communication.connect(args[1]);
                    if(sock != null){
                        ObjectOutputStream out = new ObjectOutputStream(
                                sock.getOutputStream());
                        ObjectInputStream in = new ObjectInputStream(
                                sock.getInputStream());
                        Authentication.login(in, out, user, pass);

                        String [] snd = {args[5], args[6]};
                        System.out.println("-d " + args[5] + " " + args[6]);
                        Communication.sendCommand(out, "-d", snd);

                        out.close();
                        in.close();
                        sock.close();
                    }
                    break;

                //INVALID PARAMETERS
```

```
                    default:
                        System.out.println("Incorrect parameters!\n");
                        Usage.printUsage();


                    }
                }
            }
        }
}

```

```java
package functionality;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Scanner;

/**
 * This class handles the authentication process
 */
public class Authentication {

    /*************************************************************
     * CLIENT SIDE
     *************************************************************/

    /**
     * Authenticates with the server
     *
     * @param in
     *            - ObjectInputStream
     * @param out
     *            - ObjectOutputStream
     * @param username
     *            - User's username
     * @param password
     *            - User's password
     * @return true if user was successfully logged in
     */
    public static boolean login(ObjectInputStream in, ObjectOutputStream out,
String username, String password) {

        boolean userExists = false;
        boolean usernameAvailable = false;
        boolean successLogin = false;

        try {
            System.out.println("Trying to authenticate...");

            out.writeObject(username);
            out.writeObject(password);
            out.flush();

            // Receive if user exists
            userExists = in.readBoolean();

            // If user is doesn't exist
            if (!userExists) {

                // Check if available
                usernameAvailable = in.readBoolean();

                if (usernameAvailable) {

                    Scanner read = new Scanner(System.in);
                    String ans = "";

                    // Keep prompting for a valid answer to register or not
                    while (!ans.equals("n") && !ans.equals("y")) {

                        System.out.println("This username is not registered. Do you
want to register it? (Y/N)");
                        ans = read.nextLine().toLowerCase();
                        System.out.println(ans);
                    }

                    //read.close();

                    if (ans.equals("y")) {
                        out.writeBoolean(true);
                        out.flush();
                    } else {
                        out.writeBoolean(false);
                        out.flush();
                    }
                }

                successLogin = in.readBoolean();

                if (successLogin) {
                    System.out.println("User registered successfully!");
                } else {
                    System.out.println("User not registered!");
                }

                // Se o user existir verificar se foi logado com sucesso
            } else {

                successLogin = in.readBoolean();

                if (successLogin) {
                    System.out.println("Successfull Authentication!");
                    return true;
                } else {
                    System.out.println("Failed Authentication! Incorrect 
Credentials!");
                    return false;
                }
            }

        } catch (IOException e) {

            System.err.println("Error during autentication!");
            // e.printStackTrace();
        }

        return false;
    }

    /*************************************************************
     * SERVER SIDE
     *************************************************************/

    /**
     * Authenticates a user - If user exists checks for password, if not,
     * registers
     * 
     * @param in
     *            - ObjectInputStream
     * @param out
     *            - ObjectOutputStream
     * @param login
     *            - Credentials supplied during login in format user:password
     * @return True if password correct or new user was registered, False
     *         otherwise
     */
    public static boolean authenticateUser(ObjectInputStream in, ObjectOutputStream 
out, String login) {

        String creds;
        String[] parsedCreds = login.split(":");
        boolean register = false;

        try {
            // Verify if user exists
            if ((creds = User.userExists(parsedCreds[0])) != "") {

                // Send that the user exists
                out.writeBoolean(true);
                out.flush();

                String[] registeredCredentials = creds.split(":");

                // If password matches the one registered
                if (registeredCredentials[1].equals(parsedCreds[1])) {
                    System.out.println(registeredCredentials[0] + " logged in with success");
                    out.writeBoolean(true);
                    out.flush();
                    return true;

                } else {

                    System.out.println(registeredCredentials[0] + "'s password doesn't match");
                    out.writeBoolean(false);
                    out.flush();
                    return false;
                }

                // User doesn't exist, register it
            } else {

                // Send that the user doesn't exist
                out.writeBoolean(false);
                out.flush();

                // Verify if a group exists with the same name
                if (!Group.groupExists(parsedCreds[0])) {

                    // Tell username is available
                    out.writeBoolean(true);
                    out.flush();

                    register = in.readBoolean();

                    if (register) {
                        boolean result = User.createUser(login);
                        out.writeBoolean(result);
                        out.flush();
                        return result;

                    } else {
                        System.out.println("User not registered!");
                        out.writeBoolean(false);
                        out.flush();
                        return false;
                    }

                } else {

                    // Send username is not available
                    out.writeBoolean(false);
                    out.flush();
                    return false;
                }
            }

        } catch (IOException e) {

            System.err.println("Error during authentication!");
            // e.printStackTrace();
        }

        return false;
    }
}

```

```java
package functionality;

import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;

/**
 * This class handles communication between client/server
 */
public class Communication {

    /**
     * Connects to a server in the address and port provided
     *
     * @param addressAndPort
     *             - Server's address and port in X.X.X.X:YYYY FORMAT
     * @return A socket with the connection to the server, null in case of fail
     */
    public static Socket connect(String addressAndPort) {

        Socket sock = null;

        try {
            if (addressAndPort.matches("\\d+\\.\\d+\\.\\d+\\.\\d+\\:\\d+")) {
                String[] addPort = addressAndPort.split(":");
                sock = new Socket(addPort[0], Integer.parseInt(addPort[1]));
            }
        } catch (IOException e) {

            System.err.println("Error connecting to server!\nCheck if server or
connection are down!");
            // e.printStackTrace();
        }
        return sock;
    }

    /**
     * Sends a command flag and it's arguments to the server
     *
     * @param out
     *             - ObjectOutputStream
     * @param flag
     *             - The command flag for the operation
     * @param args
     *             - The arguments for the operation
     */
    public static void sendCommand(ObjectOutputStream out, String flag, String[]
args) {

        try {
            out.writeObject(flag);
            out.writeObject(args);
            out.flush();

        } catch (IOException e) {
            System.err.println("Error sending flag and arguments to server!");
            // e.printStackTrace();
        }
    }
}

```

```java
package functionality;

/**
 * The configurations class contains the definition of various paths and data sizes
 */
public class Configurations {

    //Credential file and group list file
    public final static String CREDENTIALS_FILENAME = "!credentialsFile";
    public final static String GROUPS_FILENAME = "!groupsFile";

    //User and group data saving destination
    public final static String USERS_FOLDER = "users";
    public final static String GROUPS_FOLDER = "groups";

    //Messages saving destination
    public final static String MESSAGES_FOLDER = "messages";

    //Path to files received from -r DEST FILENAME
    public final static String DOWNLOAD_FOLDER = "downloads";

    //Data block size to send and receive files
    public final static int DATA_BLOCK = 1024;
}

```

```java
package functionality;

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.util.ArrayList;
import java.util.List;

/**
 * This class handles files and folder operations
 */
public class Files {

    /**
     * Returns a String array with a list of folder names in path
     *
     * @param path
     *            - The path to map
     * @return - String array with the folders in path
     */
    public static String[] listFolders(String path) {

        File directory = new File(path);

        File[] folderList = directory.listFiles();
        List<String> list = new ArrayList<String>();

        for (File folder : folderList) {
            if (folder.isDirectory()) {
                list.add(folder.getName());
            }
        }

        return list.toArray(new String[list.size()]);
    }

    /**
     * Returns a String array with a list of file names in path
     *
     * @param path
     *            - The path to map
     * @return - String array with the file names in path
     */
    public static String[] listFiles(String path) {

        File directory = new File(path);

        File[] fileList = directory.listFiles();
        List<String> list = new ArrayList<String>();

        if (fileList != null) {
            for (File file : fileList) {
                if (file.isFile()) {
                    list.add(file.getName());
                }
            }
        }
        return list.toArray(new String[list.size()]);
    }

    /**
```

```
     * Creates the correct path according if destinatary is a group or a user
     *
     * @param destinatary
     *            - The destinatary
     * @param callingUser
     *            - The user who calls the method (remetent)
     * @return A#B destinatary is a user, B if destinatary is group, or "" if
     *         error
     */
    public static String getDestination(String destinatary, String callingUser) {

        int val = Group.isUserOrGroup(destinatary);
        String destination = "";

        // Destinatary is user
        if (val == 1) {

            // WRITE A MESSAGE SHOWING A FILE WAS SENT
            StringBuilder dst = new StringBuilder();

            if (destinatary.compareTo(callingUser) <= 0) {

                dst.append(destinatary);
                dst.append("#");
                dst.append(callingUser);

            } else {

                dst.append(callingUser);
                dst.append("#");
                dst.append(destinatary);

            }

            destination = dst.toString();

            // Destinatary is group
        } else if (val == 0) {

            destination = destinatary;

        }

        return destination;
    }

    /**
     * Deletes a folder and it's contents
     *
     * @param folder
     *            - A File object with the folder's path
     */
    static void deleteFolder(File folder) {

        // List all files
        File[] files = folder.listFiles();

        if (files != null) {
            // Delete one by one
            for (File f : files) {

                if (f.isDirectory()) {
                    deleteFolder(f);
                } else {
```

```java
                    f.delete();
                }
            }
        }
        // Delete folder
        folder.delete();
    }

    /**
     * Sends a file
     *
     * @param in
     *            - ObjectInputStream
     * @param out
     *            - ObjectOutputStream
     * @param filePath
     *            - The path of the file
     * @param fileName
     *            - The file name
     * @return True if sending was successful, False otherwise
     */
    public static boolean sendFile(ObjectInputStream in, ObjectOutputStream out,
 String filePath, String fileName) {

        byte[] buffer = new byte[Configurations.DATA_BLOCK];
        FileInputStream file = null;

        try {

            file = new FileInputStream(filePath + fileName);

            // Asks if destination is valid
            boolean validDestination = in.readBoolean();

            if (validDestination) {

                // Asks if file exists remotely
                boolean fileExistsAtDestination = in.readBoolean();

                // If file doesn't exist send
                if (!fileExistsAtDestination) {
                    // Send file size
                    long fileSize = file.getChannel().size();
                    out.writeLong(fileSize);
                    out.flush();

                    // Send file
                    int count;
                    while ((count = file.read(buffer)) > 0) {
                        out.write(buffer, 0, count);
                        out.flush();
                    }
                    file.close();

                    // Receive received byte amount
                    long rcvdBytes = in.readLong();

                    if (rcvdBytes == fileSize) {
                        System.out.println("File sent with success!");
                        return true;
                    } else {
                        System.out.println("Failed to send file!");
                        return false;
                    }

                } else {
                    file.close();
                    System.out.println("File already exists!");
                    return false;
                }

            } else {
                file.close();
                System.out.println("User or group doesn't exist!");
            }

        } catch (IOException e) {
            System.err.println("Error sending file!");
            e.printStackTrace();
        }

        try {
            file.close();
        } catch (IOException e) {
            // e.printStackTrace();
            System.err.println("File not found!");
        }
        return false;
    }

    /**
     * Receives a file
     *
     * @param in
     *            - ObjectInputStream
     * @param out
     *            - ObjectOutputStream
     * @param path
     *            - The path to save the file
     * @param fileName
     *            - The file name
     * @return True if file received successfully, False otherwise
     */
    public static boolean receiveFile(ObjectInputStream in, ObjectOutputStream out,
 String path, String fileName) {

        byte[] buffer = new byte[Configurations.DATA_BLOCK];
        FileOutputStream file = null;
        String fullPath;

        try {
            // Answers if destination is valid
            if (path.equals("") || path == null) {
                fullPath = fileName;
            } else {
                fullPath = path + "/" + fileName;
            }

            File dest = new File(path);
            if (dest.exists() && dest.isDirectory()) {

                out.writeBoolean(true);
                out.flush();

                // Answers if file already exists
                File checkIfExists = new File(fullPath);

                if (checkIfExists.exists()) {
                    System.out.println("File already exists locally!");
                    out.writeBoolean(true);
                    out.flush();
                    return false;

                } else {
                    System.out.println("Receive file!");
                    out.writeBoolean(false);
                    out.flush();

                    // Receive file size
                    long fileSize = in.readLong();

                    // Receive file
                    long recvd = 0;
                    if (fileSize > 0) {

                        file = new FileOutputStream(fullPath);

                        int count;
                        recvd = 0;
                        while (recvd < fileSize) {
                            count = in.read(buffer);
                            file.write(buffer, 0, count);
                            recvd += count;
                        }

                        file.close();
                    }

                    // Send received bytes amnount
                    out.writeLong(recvd);
                    out.flush();
                    return true;
                }

            } else {
                System.out.println("User or group does not exist!");
                out.writeBoolean(false);
                out.flush();
                return false;
            }

        } catch (IOException e) {
            System.err.println("Error receiving file!");
            // e.printStackTrace();
        }

        return false;
    }
}

```

```java
package functionality;

import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 * This class handles group operations
 */
public class Group {

    /**
     * Verifies if a groups exists in the groups file
     *
     * @param groupName
     *            - Name of the group to be searched for
     * @return True if group exists in groups file
     */
    static boolean groupExists(String groupName) {

        try {

            BufferedReader br = new BufferedReader(new FileReader(Configurations.GROUPS_FILENAME));

            String line;
            while ((line = br.readLine()) != null) {

                if (line.equals(groupName)) {
                    br.close();
                    System.out.println("Group exists in groups file!");
                    return true;
                }
            }

            br.close();

        } catch (IOException e) {
            System.err.println("Error verifying if group exists!");
            // e.printStackTrace();
        }
        return false;
    }

    /**
     * Checks if contact is a client or a group
     *
     * @param input
     *            - the contact's id
     * @return 1 if it's a user, 0 if it's a group and -1 in case of error
     */
    public static int isUserOrGroup(String input) {

        try {
            // Check if is User
            if (User.userExists(input) != "") {
                return 1;

                // Check if is group
            } else if (groupExists(input)) {
                return 0;
            }

        } catch (IOException e) {
            System.err.println("Erro verifying if it's a user or a group!");
            // e.printStackTrace();
        }
        return -1;
    }

    /**
     * Creates a new group if it doesn't exist, or adds a member if it does
     *
     * @param username
     *             - Username of the user to be added
     * @param groupName
     *             - Name of the group to be created
     * @param callingUser
     *             - The user who invokes the method
     * @param out
     *             - ObjectOutputStream
     * @param in
     *             - ObjectInputStream
     * @return True if group was created/user added to group successfully, False
     *         otherwise
     * @throws IOException
     */
    public static boolean addGroup(String username, String groupName, String callingUser, ObjectOutputStream out,
            ObjectInputStream in) throws IOException {

        try {
            // Check if there's a user with the supplied group name
            if (User.userExists(groupName).equals("")) {

                // Check if group already exists
                if (groupExists(groupName)) {

                    // If group exists, check if calling user is admin
                    if (isAdmin(groupName, callingUser)) {

                        // Check if user to be added already exists in group
                        // If it doesn't, add it!
                        if (!isInGroup(groupName, username) && !(User.userExists(username).equals(""))) {

                            BufferedWriter bw = new BufferedWriter(
                                    new FileWriter(Configurations.GROUPS_FOLDER + "/" + groupName + ".cfg", true));

                            bw.append(username);
                            bw.newLine();
                            bw.close();

                            out.writeObject("User added with sucess!");
                            out.flush();
                            return true;

                            // If it's already added to the group
                        } else {
                            out.writeObject("Failed to add " + username + " to group " + groupName
                                    + "!\nUser doesn't exist or it's already on
```

```java
this group");
                    out.flush();
                    System.out.println("Failed to add " + username + " to group " + groupName
                            + "!\nUser doesn't exist or it's already on this group");
                    return false;
                }

                // The user is not an admin
            } else {
                out.writeObject("Failed to add " + username + " to the group " + groupName
                        + ". You are not the admin of this group.");
                out.flush();
                System.out.println("Failed to add " + username + " to the group " + groupName
                        + ". You are not the admin of this group.");
                return false;

            }

            // The group doesn't exist
        } else {

            // Create group Folder
            File createFolder = new File(Configurations.MESSAGES_FOLDER + "/" + groupName);

            createFolder.mkdir();

            // Writes the group properties file
            BufferedWriter bw = new BufferedWriter(
                    new FileWriter(Configurations.GROUPS_FOLDER + "/" + groupName + ".cfg", true));

            // Check if user is trying to add himself
            // It that's the case, write only one line
            if (username.equals(callingUser)) {

                bw.append(username);
                bw.newLine();

                // If user not trying to add himself
            } else {
                // Adicionar linha do admin
                bw.append(callingUser);
                bw.newLine();

                // Add group to calling user user file
                BufferedWriter callingUserPersonalFile = new BufferedWriter(
                        new FileWriter(Configurations.USERS_FOLDER + "/" + callingUser + ".cfg", true));
                callingUserPersonalFile.write(groupName);
                callingUserPersonalFile.newLine();
                callingUserPersonalFile.close();

                // Check if user to be added exists
                if (!User.userExists(username).equals("")) {
                    // If so add user to group list member
                    bw.append(username);
                    bw.newLine();

```

```java
                            // Add group to user file
                            BufferedWriter usrPersonalFile = new BufferedWriter(
                                    new FileWriter(Configurations.USERS_FOLDER + "/" + username + ".cfg", true));
                            usrPersonalFile.write(groupName);
                            usrPersonalFile.newLine();
                            usrPersonalFile.close();

                            System.out.println("Group created succesfully");
                            out.writeObject("Group created succesfully");
                            out.flush();

                            // User to eb added is not registered
                        } else {

                            out.writeObject("Group created but user " + username
                                    + " could not be added because it doesn't exist");
                            out.flush();
                            System.out.println("Group created but user " + username
                                    + " could not be added because it doesn't exist");
                        }
                    }

                    bw.close();

                    // Register group in groups file
                    BufferedWriter gf = new BufferedWriter(new FileWriter(Configurations.GROUPS_FILENAME, true));
                    gf.write(groupName);
                    gf.newLine();
                    gf.close();
                }

                // If the group to be created conflicts with an already
                // registered group/user
            } else {
                out.writeObject("Failed to create group! A user with this name already exists!");
                out.flush();
                System.out.println("Failed to create group! A user with this name already exists");
                return false;
            }

        } catch (IOException e) {
            System.out.println("Failed to create group/add new element!");
            out.writeObject("Failed to create group! A user with this name already exists!");
            out.flush();
            // e.printStackTrace();
        }

        return false;
    }

    /**
     * Verifies if a user is admin of a group
     *
     * @param groupName
     *            - The name of the group for this query
     * @param username
     *            - The username of the client for this query
     * @return True if user is admin of group, otherwise, False
     */
    private static boolean isAdmin(String groupName, String username) {

        try {

            BufferedReader br = new BufferedReader(
                    new FileReader(Configurations.GROUPS_FOLDER + "/" + groupName + ".cfg"));

            String line = br.readLine();

            if (line.equals(username)) {
                br.close();
                System.out.println("User " + username + " is Admin of group " + groupName);
                return true;
            }
            br.close();

        } catch (IOException e) {
            // e.printStackTrace();
            System.err.println("Error removing user from group!");
        }
        return false;
    }

    /**
     * Verifies if user is in group
     *
     * @param groupName
     *                  - The name of the group
     * @param username
     *                  - The user's username
     * @return True if
     */
    private static boolean isInGroup(String groupName, String username) {

        try {

            BufferedReader br = new BufferedReader(
                    new FileReader(Configurations.GROUPS_FOLDER + "/" + groupName + ".cfg"));

            String line;

            while ((line = br.readLine()) != null) {
                if (line.equals(username)) {
                    br.close();
                    System.out.println("User " + username + " is in group " + groupName);
                    return true;
                }
            }
            br.close();

        } catch (IOException e) {
            System.err.println("Error verifying if user belongs to group!");
            // e.printStackTrace();
        }
        return false;
    }

    /**
```

```
     * Removes an entry from a text file
     *
     * @param filename
     *            - File to be redacted
     * @param entry
     *            - Entry to redact
     * @return True if entry was successfully redacted
     */
    public static boolean removeEntry(String filename, String entry) {

        try {
            BufferedReader in = new BufferedReader(new FileReader(filename));
            StringBuilder sb = new StringBuilder();

            // Stripe the entry
            String line;
            while ((line = in.readLine()) != null) {
                // If it's not the entry we're looking for, append
                if (!line.equals(entry)) {
                    sb.append(line);
                    sb.append("\n");
                }
            }

            in.close();

            // Write the new file
            BufferedWriter out = new BufferedWriter(new FileWriter(filename));
            out.write(sb.toString());
            out.close();

            return true;

        } catch (IOException e) {
            System.err.println("Error in file operations for entry removal!");
            // e.printStackTrace();
        }

        return false;

    }

    /**
     * Removes a user from a group and cleans the respective configuration files
     *
     * @param username
     *            - The username of the user to be removed
     * @param groupName
     *            - The groups name
     * @param callingUser
     *            - The user who invokes this method
     * @return True if user was successfully removed, False otherwise
     */
    public static boolean removeFromGroup(String username, String groupName, String
callingUser) {

        try {
            // Verificar se quem chama o metodo e o admin do grupo
            if (isAdmin(groupName, callingUser)) {
                // Se o utilizador existe e pertence ao grupo
                if (User.userExists(username) != null && isInGroup(groupName,
username)) {

                    // Se o utilizador a remover Â´e admin apaga o grupo todo
```

```java
                    if (username.equals(callingUser)) {

                        // Pega na lista de elementos do grupo e percorre-a
                        BufferedReader in = new BufferedReader(
                                new FileReader(Configurations.GROUPS_FOLDER + "/" + groupName + ".cfg"));

                        // Remove as entradas do ficheiro pessoal
                        String line;
                        while ((line = in.readLine()) != null) {

                            removeEntry(Configurations.USERS_FOLDER + "/" + line + ".cfg", groupName);

                        }

                        // apaga o ficheiro de membros

                        File memberFile = new File(Configurations.GROUPS_FOLDER + "/" + groupName + ".cfg");
                        memberFile.delete();

                        in.close();
                        return true;
                        // Just delete the element
                    } else {

                        // Remover do ficheiro do grupo
                        removeEntry(Configurations.GROUPS_FOLDER + "/" + groupName + ".cfg", username);

                        // remover do ficheiro pessoal
                        removeEntry(Configurations.USERS_FOLDER + "/" + username + ".cfg", groupName);

                        return true;

                    }

                } else {
                    System.out.println("User " + username + "doesn't exist or is not a member of " + groupName);
                    return false;
                }

            } else {
                System.out.println(callingUser + " is not an admin of " + groupName + " and cannot delete " + username);
                return false;
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            System.err.println("Error removing user from group!");
            e.printStackTrace();
        }

        return false;
    }
}
```

```java
package functionality;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * This class handles Message operations
 */
public class Message {

    /**
     * Builds a conversation from the individual message files
     *
     * @param dest
     *             - The destinatary
     * @param callingUser
     *             - The sender
     * @param lastestOnly
     *             - If it shows only the last message exchanged with
     * @return A string with the conversation
     */
    public static String buildConversation(String dest, String callingUser, boolean lastestOnly) {

        // RECONSTRUCT CONVERSATION
        StringBuilder conversationBack = new StringBuilder();
        ;
        try {
            System.out.println("Send all communications with " + dest);

            String destination = Files.getDestination(dest, callingUser);

            String[] filesList = Files.listFiles(Configurations.MESSAGES_FOLDER + "/" + destination);

            if (filesList.length != 0) {
                // SHOW WHO IS THIS CONVERSATION WITH
                conversationBack.append("Contact: " + dest + "\n");

                if (lastestOnly) {
                    String s = filesList[filesList.length - 1];

                    if (s.matches("([^\\s]+(\\.(?i)(msg))$)")) {

                        conversationBack.append(Message.parseMessage(
                                Configurations.MESSAGES_FOLDER + "/" + destination + "/" + s, callingUser));

                    }

                } else {

                    for (String s : filesList) {

                        if (s.matches("([^\\s]+(\\.(?i)(msg))$)")) {

                            conversationBack.append(Message.parseMessage(
                                    Configurations.MESSAGES_FOLDER + "/" + destination + "/" + s, callingUser));
                        }
                    }
                }
            } else {
                conversationBack.append("No conversations found for user/group " + dest);
            }

        } catch (IOException e) {
            System.err.println("Error building conversation!");
            // e.printStackTrace();
        }
        return conversationBack.toString();
    }

    /**
     * Presents a message propperly
     *
     * @param path
     *             - Path to the message file
     * @param callingUser
     *             - The sender
     * @return A string with the parsed message
     */
    public static String parseMessage(String path, String callingUser) throws IOException {

        BufferedReader br;
        StringBuilder sb = new StringBuilder();

        br = new BufferedReader(new FileReader(path));

        String tmp;
        int pos = 0;
        while ((tmp = br.readLine()) != null) {
            if (pos == 0) {
                if (tmp.equals(callingUser))
                    sb.append("me: ");
                else
                    sb.append(tmp + ": ");
            } else {
                sb.append(tmp);
                sb.append("\n");
            }
            pos++;
        }
        br.close();
        return sb.toString();
    }

    /**
     * Receives and stores a message propperly
     *
     * @param out
     *             - Output Stream
     * @param in
     *             - Input Stream
     * @param callingUser
     *             - The user currently logged it to save who sent the message
     */
    public static void receiveMessage(ObjectOutputStream out, ObjectInputStream in,
String contact, String message,
            String callingUser) {

        try {

            String destination = Files.getDestination(contact, callingUser);
            // -1 - not registered
            // 0 - group
            // 1 - user
            if (destination == "") {

                // Tells destination doesn't exist
                // out.writeBoolean(false);
                System.out.println("Message destinatary doesn't exist!");
                out.writeObject("Message destinatary doesn't exist!");
                out.flush();
            } else {

                writeMessage(message, callingUser, destination);
                System.out.println("Message sent from " + callingUser + " to " + contact);
                out.writeObject("Message sent from " + callingUser + " to " + contact);
                out.flush();
            }

        } catch (IOException e) {

            System.err.println("Error receiving message!");
            try {
                out.writeObject("Error sending message");
                out.flush();
            } catch (IOException e1) {
                // e1.printStackTrace();
            }
            // e.printStackTrace();
        }

    }

    /**
     * Writes a message to disk
     *
     * @param message
     *            - The message to be written
     * @param callingUser
     *            - The one who sends the message
     * @param destination
     *            - The one who receives the message
     */
    public static void writeMessage(String message, String callingUser, String destination) throws IOException {
        System.out.println("Message: " + message);

        // Create a conversation folder
        File messagePath = new File(Configurations.MESSAGES_FOLDER + "/" + destination);
        if (!messagePath.exists()) {
            messagePath.mkdir();
        }

        // Build the message
        StringBuilder sb = new StringBuilder();

        sb.append(callingUser);
        sb.append("\n");
        sb.append(message);
        sb.append("\n");

        Date date = new Date();
        SimpleDateFormat ft = new SimpleDateFormat("yyyy-MM-dd hh:mm");
        sb.append(ft.format(date));

        // Write the message to disk
        BufferedWriter msg = new BufferedWriter(
                new FileWriter(Configurations.MESSAGES_FOLDER + "/" + destination + "/" + date.getTime() + ".msg"));
        msg.write(sb.toString());
        msg.close();
    }

    /*************************************************************
     * CLIENT SIDE
     *************************************************************/

    /**
     * Sends a message to the server
     *
     * @param out
     *             - Output Stream
     * @param in
     *             - Input Stream
     * @param destination
     *             - The user or group for the message to be sent
     * @param message
     *             - The message to be sent
     */
    public static void sendMessage(ObjectOutputStream out, ObjectInputStream in) {

        try {
            // Receive status from message delivery
            String result = (String) in.readObject();
            System.out.println(result);

        } catch (IOException | ClassNotFoundException e) {
            // TODO Auto-generated catch block
            System.err.println("Error sending message to server!");
            // e.printStackTrace();
        }
    }
}

```

```java
package functionality;

/**
 * This class handles usage display of application
 */
public class Usage {

    /**
     * Prints the usage menu
     */
    public static void printUsage() {

        System.out.println("myWhats <localUser> <serverAddress> [ -p <password> ]\n"
                         + "                                    [ -m <contact> <message> ]\n"
                         + "                                    [ -f <contact> <file> ]\n"
                         + "                                    [ -r [contact] [file] ]\n"
                         + "                                    [ -a <user> <group> ]\n"
                         + "                                    [ -d <user> <group> ]\n");
    }
}

```

```java
package functionality;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

/**
 * This class handles User operations
 */
public class User {

    /**
     * Adds a user to the credentials file
     *
     * @param login
     *            - Credentials to be added
     */
    static boolean createUser(String login) {

        try {

            // Insert credentials in the creds file
            BufferedWriter bw = new BufferedWriter(new FileWriter(Configurations.CREDENTIALS_FILENAME, true));

            bw.append(login);
            bw.newLine();
            bw.close();

            // Create user's personal folder
            String[] parseCreds = login.split(":");

            BufferedWriter uf = new BufferedWriter(
                    new FileWriter(Configurations.USERS_FOLDER + "/" + parseCreds[0] + ".cfg", true));
            uf.close();

            System.out.println("New user registered");

            return true;

        } catch (IOException e) {

            System.err.println("Error opening credentials file for user insertion");
            // e.printStackTrace();
        }
        return false;
    }

    /**
     * Verifies if a username is present in the credentials file
     *
     * @param username
     *            - User's username
     *
     * @return If username if present it's credentials will be returned
     *         otherwise, empty will be returned
     * @throws IOException
     */
    static String userExists(String username) throws IOException {

        try {

            BufferedReader br = new BufferedReader(new FileReader(Configurations.CREDENTIALS_FILENAME));

            String line;
            String[] credLine;
            while ((line = br.readLine()) != null) {

                credLine = line.split(":");

                if (credLine[0].equals(username)) {
                    br.close();
                    System.out.println("Username exists in credentials file");
                    return line;
                }
            }

            br.close();

        } catch (FileNotFoundException e) {

            System.err.println("Error opening credentials file for user verification");
            // e.printStackTrace();
        }

        System.out.println("Username not found");
        return "";
    }
}

```

```
package server;

/***************************************************************************
 *                    Seguranca e Confiabilidade 2015/16
 ***************************************************************************/

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

import functionality.*;

//Servidor do servico myWhatsServer
public class myWhatsServer {

    //MAIN
    public static void main(String[] args) {
        System.out.println("Server running");
        myWhatsServer sv = new myWhatsServer();
        sv.startServer();
    }

    //START SERVER
    @SuppressWarnings("resource")
    public void startServer() {
        ServerSocket sSoc = null;

        try {

            sSoc = new ServerSocket(23456);

        } catch (IOException e) {

            System.err.println(e.getMessage());
            System.exit(-1);

        }

        //
***************************************************************************

        // Check if USERS file exist
        File credentials = new File(Configurations.CREDENTIALS_FILENAME);
        if (!credentials.exists() && !credentials.isDirectory()) {

            try {

                credentials.createNewFile();

            } catch (IOException e) {
                System.err.println("Error creating user file");
                e.printStackTrace();

            }
        }

        // Check if GROUPS file exist
        File groups = new File(Configurations.GROUPS_FILENAME);
        if (!groups.exists() && !groups.isDirectory()) {

            try {

                groups.createNewFile();

            } catch (IOException e) {
                System.err.println("Error creating groups file");
                e.printStackTrace();

            }
        }

        // Check if users FOLDER exist
        File usersFolder = new File(Configurations.USERS_FOLDER);
        if (!usersFolder.exists()) {

            usersFolder.mkdir();
        }

        // Check if groups FOLDER exist
        File groupsFolder = new File(Configurations.GROUPS_FOLDER);
        if (!groupsFolder.exists()) {

            groupsFolder.mkdir();
        }

        // Check if messages FOLDER exist
        File messagesFolder = new File(Configurations.MESSAGES_FOLDER);
        if (!messagesFolder.exists()) {

            messagesFolder.mkdir();
        }

        //******************************************************************************

        // Server Main - Client reception and thread creation
        while (true) {
            try {
                Socket inSoc = sSoc.accept();
                ServerThread newServerThread = new ServerThread(inSoc);
                newServerThread.start();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }

    //******************************************************************************

    // Threads utilizadas para comunicacao com os clientes
    class ServerThread extends Thread {

        private Socket socket = null;

        ServerThread(Socket inSoc) {
            socket = inSoc;
            System.out.println("Client connected from " + socket.getInetAddress());
        }

        // Run the thread
        public void run() {

            try {
```

```java
                // OPEN STREAMS
                ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
                ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

                String user = null;
                String passwd = null;
                // END OPEN STEAMS

                // RECEIVE CREDENTIALS
                try {
                    user = (String) in.readObject();
                    passwd = (String) in.readObject();

                    System.out.println(
                            "Authentication attempt\n>" + socket.getInetAddress() + " | " + user + " | " + passwd);

                } catch (ClassNotFoundException e1) {
                    e1.printStackTrace();
                }
                // END RECEIVE CREDENTIALS

                // AUTHENTICATION
                boolean wasSuccessful = false;

                if (user.length() > 0 && passwd.length() > 0) {
                    wasSuccessful = Authentication.authenticateUser(in, out, (user + ":" + passwd));
                }

                // END AUTHENTICATION

                if (wasSuccessful) {
                    try {
                        String flag = (String) in.readObject();
                        String[] argArray = (String[]) in.readObject();
                        String destination;

                        // Procede according to flag sent
                        switch (flag) {
                        case "-m":
                            System.out.println("Message To: " + argArray[0] + " - " + argArray[1]);
                            Message.receiveMessage(out, in, argArray[0], argArray[1], user);

                            break;

                        case "-f":
                            System.out.println("File to: " + argArray[0] + " - " + argArray[1]);

                            destination = Files.getDestination(argArray[0], user);

                            if (destination != "") {

                                boolean success = Files.receiveFile(in, out,
                                        Configurations.MESSAGES_FOLDER + "/" + destination, argArray[1]);

                                if (success) {
                                    Message.writeMessage(argArray[1], user,
destination);
                                }

                            } else {

                                // Send that is not a valid destination
                                out.writeBoolean(false);
                                out.flush();

                                System.out.println("ERROR! Not a user or group!");
                            }
                            break;

                        case "-r":
                            if (argArray.length == 0) {
                                System.out.println("Send all communications");

                                // LIST ALL FOLDERS THAT CONTAIN THE USER's
                                // USERNAME

                                String[] folderList =
Files.listFolders(Configurations.MESSAGES_FOLDER);
                                StringBuilder convo = new StringBuilder();
                                for (String s : folderList) {

                                    if (s.matches(".+#.+")) {

                                        String[] tmp = s.split("#");

                                        // RECONSTRUCT CONVERSATION
                                        if (tmp[0].equals(user)) {

                                            convo.append(Message
                                                    .buildConversation(tmp[1],
user, true));
                                        }else if( tmp[1].equals(user)){
                                            convo.append(Message
                                                    .buildConversation(tmp[0],
user, true));
                                        }
                                    }
                                }

                                // SEND
                                out.writeObject(convo.toString());
                                out.flush();

                            } else if (argArray.length == 1) {

                                String convo =
Message.buildConversation(argArray[0], user, false);

                                // SEND
                                out.writeObject(convo);
                                out.flush();

                            } else if (argArray.length == 2) {
                                System.out.println("Get file " + argArray[1] + " in
" + argArray[0]);

                                // OPEN CONVERSATION FOLDER
                                destination = Files.getDestination(argArray[0],
user);
                                File file = new File(Configurations.MESSAGES_FOLDER
+ "/" + destination + "/" + argArray[1]);

                                if(file.exists()){
                                    out.writeBoolean(true);
                                    out.flush();
                                    Files.sendFile(in, out, Configurations.MESSAGES_FOLDER + "/" + destination + "/", argArray[1]);
                                }else{
                                    out.writeBoolean(false);
                                    out.flush();
                                }

                            }
                            break;

                        case "-a":
                            System.out.println("Adds user " + argArray[0] + " to group " + argArray[1]);
                            Group.addGroup(argArray[0], argArray[1], user, out, in);

                            break;

                        case "-d":
                            System.out.println("Delete user " + argArray[0] + " from group " + argArray[1]);
                            Group.removeFromGroup(argArray[0], argArray[1], user);
                            break;

                        default:
                            System.out.println("Incorrect parameters!\n");
                        }

                    } catch (IOException | ClassNotFoundException e) {
                        System.out.println("No command was sent from client or unexpected data type!");
                        // e.printStackTrace();
                    }
                }

                // END SERVER MAIN

                System.out.println("-=*=-=*=-=*=- END OF SESSION -=*=-=*=-=*=-");
                // SOCKT AND STREAM CLOSING
                out.close();
                in.close();

                socket.close();
                // END SOCKT AND STREAM CLOSING

            } catch (IOException e) {
                System.err.println("Error running server thread!");
                //e.printStackTrace();
            }
        }
    }
}
```