
SDIS 2019/2020

Mestrado Integrado em Engenharia Informática e Computação

# Generalized Backup System


| | |
|---|---|
| Duarte Faria | up201607176 |
| Luís Spínola | up201405907 |
| Matheus Gonçalves | up201405081 |
| Tomás Figueiredo | up201607932 |

# Overview

A aplicação desenvolvida contém um Servidor central responsável por manter o estado do sistema, ou seja gerir o acesso dos Peers, gerir as actividades iniciadas por cada Peer e guardar um registo de todas as actividades realizadas.

Os protocolos implementadas foram os seguintes: Register, Login, CheckUserName, Logout, Backup, Restore, Delete e GetFiles.

Para além destes protocolos, e de utilizarmos um Servidor central que lida com todas as operações, a nossa arquitectura também usa threads individuais por actividade tanto para o Servidor como para cada Peer. Ao mesmo tempo, utilizamos ligações TCP para realizar todas as comunicações no nosso sistema como também JSSE e SSLSockets de forma a tornar todas as comunicações mais seguras como também a informação destas.

Por último, em termos de escalabilidade, não utilizamos o Chord mas como referi antes, utilizamos threads de forma a tratar de pedidos simultaneamente sem que haja abrandamentos no sistema. Em termos de fault-tolerance, fazemos uma verificação a cada segundo de forma a saber se algum novo Peer conectou-se ao nosso sistema ou se algum Peer saiu para conseguirmos ter sempre uma lista actualizada de Peers conectados.

# Protocols

## Register

Cada Cliente necessita de se registar de forma a utilizar o nosso serviço. Para isto cada Cliente necessita de escolher um nome de utilizador e uma password. A validade do nome de utilizador é verificado pelo nosso Servidor utilizando o subprotocolo **CheckUserName**. Caso este nome de utilizador seja válido e a sua password também, é criada uma nova conta no nosso servidor e este Cliente passa a estar logged-in.

```
private static void registerClient() throws IOException {
    Scanner scanner = new Scanner(System.in);
    System.out.println();
    System.out.println("Insert a new username");
    String requestedUsername = scanner.nextLine();

    requestedUsername = requestedUsername.trim();

    SSLSocket socket = (SSLSocket) csf.createSocket(host, port);
    String message = "CHECKUSERNAME " + requestedUsername;
    Utils.sendMessage(message, socket);
    String response = Utils.readMessage(socket);

    socket.close();

    if (!response.equals("AVAILABLE")) {
        System.err.println("Username already taken, please try again");
        registerClient();
    }
    SSLSocket socket2 = (SSLSocket) csf.createSocket(host, port);
    Console console = System.console();
    String enteredPassword =
            new String(console.readPassword("Please enter your new password: "));

    String hashedPass = BCrypt.hashpw(enteredPassword, BCrypt.gensalt());

    String registerMessage = "REGISTER " + requestedUsername + " " + hashedPass;

    Utils.sendMessage(registerMessage, socket2);

    user = requestedUsername;

    System.out.println();
    selectOperation();
}
```

Todas as comunicações são feitas usando TCP com JSSE e o formato das mensagens trocadas neste protocolo são:

- CHECKUSERNAME <requestedUsername>, do Cliente que pretende fazer o registo no nosso sistema para o Servidor, de forma a saber se o <requestedUsername> já se encontra em uso.

- AVAILABLE, do Servidor para o Cliente caso <requestedUsername> ainda não esteja associado a nenhuma conta ou TAKEN caso já esteja associado.
- REGISTER <requestedUsername> <hashedPass>, do Cliente que pretende fazer o registo para o Servidor, de forma a realizar o processo de registo.

# Login

Caso um Cliente já esteja registado no nosso sistema, este pode fazer Login para aceder às suas funcionalidades. Para isto, é pedido ao Cliente para introduzir o seu nome de utilizador e password associada, e caso ambos estejam correctos é dado acesso ao Cliente.

```
private static void loginClient() throws IOException {
    Scanner scanner = new Scanner(System.in);
    System.out.println();
    System.out.println("Insert Username:");
    String username = scanner.nextLine();
    Console console = System.console();
    String enteredPassword =
            new String(console.readPassword("Please enter your password: "));

    String message = "LOGIN " + username.trim();

    SSLSocket socket = (SSLSocket) csf.createSocket(host, port);
    Utils.sendMessage(message, socket);

    String response = Utils.readMessage(socket);

    if (!response.equals("User not found")) { //password check
        if (!BCrypt.checkpw(enteredPassword, response)) {
            Utils.sendMessage("Password not match", socket);
            System.out.println("Incorrect password");
            auth();
        }
        Utils.sendMessage("Success", socket);

        String response2 = Utils.readMessage(socket);

        if (!response2.equals("Success")) {
            System.out.println();
            System.out.println(response2);
            auth();
        }
        System.out.println("Login Successfully as " + username);
        user = username;
        selectOperation();
    }

    System.out.println();
    auth();
}
```

Todas as comunicações são feitas usando TCP com JSSE e o formato das mensagens trocadas neste protocolo são:

- LOGIN <username>, do Cliente que pretende fazer Login para o Servidor, de forma a saber se o <username> existe no nosso sistema.
- User not found, do Servidor para o Cliente caso o <username> não exista no nosso sistema.
- User already logged in, do Servidor para o Cliente, caso o <username> e a sua password estiverem associados a uma conta no nosso sistema mas este utilizador já esteja logged-in.
- Success, do Servidor para o Cliente, caso o <username> e a sua password estiverem associados a uma conta no nosso sistema e esta ainda não esteja logged-in.

# Logout

Caso um Cliente esteja logged-in no nosso sistema, este pode fazer logout. Para isto, é apenas mandado um pedido do Cliente em questão para o Servidor, e o Servidor atualiza a sua lista de Clientes Logged-in e envia uma resposta ao Cliente.

```
private static void logoutClient() throws IOException {
    String logoutMessage = "LOGOUT " + user;
    SSLSocket socket = (SSLSocket) csf.createSocket(host, port);
    Utils.sendMessage(logoutMessage, socket);

    System.out.println("Logout successfully");
}
```

Todas as comunicações são feitas usando TCP com JSSE e o formato das mensagens trocadas neste protocolo são:

- LOGOUT <username>, do Cliente que pretende fazer Logout para o Servidor.
- SUCCESS, do Servidor para o Cliente.

# Backup

Depois de o login ser efectuado, o utilizador tem a opção de fazer Backup de um ficheiro que possui. Este protocolo começa por pedir o diretório do ficheiro e o número de vezes que ele tem de ser replicado. Depois disto, é mandada uma mensagem para o Servidor com a informação pertinente ao ficheiro em questão, como tamanho, nome, data da última modificação, data de criação e conteúdo. O Servidor ao receber esta informação redireciona-a para o número de Peers igual ao número de replicação introduzido pelo Cliente. Os Peers recebem esta informação e criam um ficheiro com o nome encriptado.

Todas as comunicações são feitas usando TCP com JSSE e o formato das mensagens trocadas neste protocolo são:

- BACKUP <username> <replicationDegree> <fileName>, que é a mensagem enviada pelo Cliente para o Servidor com o seu username, nome do ficheiro e grau de replicação.
- READY <fileID>, resposta do Servidor perante a mensagem anterior mandada pelo Cliente.
- <fileContent>, Cliente depois da resposta afirmativa do Peer manda o conteúdo do ficheiro.
- BACKUP <fileID> <fileSize>, mensagem que o Servidor manda para os Peers que vão receber o ficheiro.
- READY <fileID>, resposta do Peer perante a mensagem anterior mandada pelo Servidor.
- <fileContent>, Servidor depois da resposta afirmativa do Peer manda o conteúdo do ficheiro.
- STORED <peerID> <fileID>, mensagem que marca o fim do protocolo de Backup.

```java
private static void backupOperation() throws IOException {

    System.out.println(" ");
    System.out.println("Please input the file path");
    String path = null;

    Scanner scanner = new Scanner(System.in);

    path = scanner.nextLine();
    path = path.trim();

    File toBackup = new File(path);
    if (!toBackup.exists()) {
        System.out.println();
        System.err.println("The specified file does not exist in your system");
        System.out.println();
        selectOperation();
    }

    System.out.println();
    System.out.println("Please input the wanted replication degree");
    int replicationDegree = scanner.nextInt();

    BasicFileAttributes attr = Files.readAttributes(Paths.get(toBackup.getPath()), BasicFileAttributes.class);
    String fileInfo = toBackup.getName() + " " + attr.creationTime() + " " + attr.lastModifiedTime() + " " + attr.isRegularFile() + " " + attr.size();
    //first send backup info
    String headerString = "BACKUP " + user + " " + replicationDegree + " " + fileInfo;

    SSLSocket socket = (SSLSocket) csf.createSocket(host, port);
    Utils.sendMessage(headerString, socket);
```

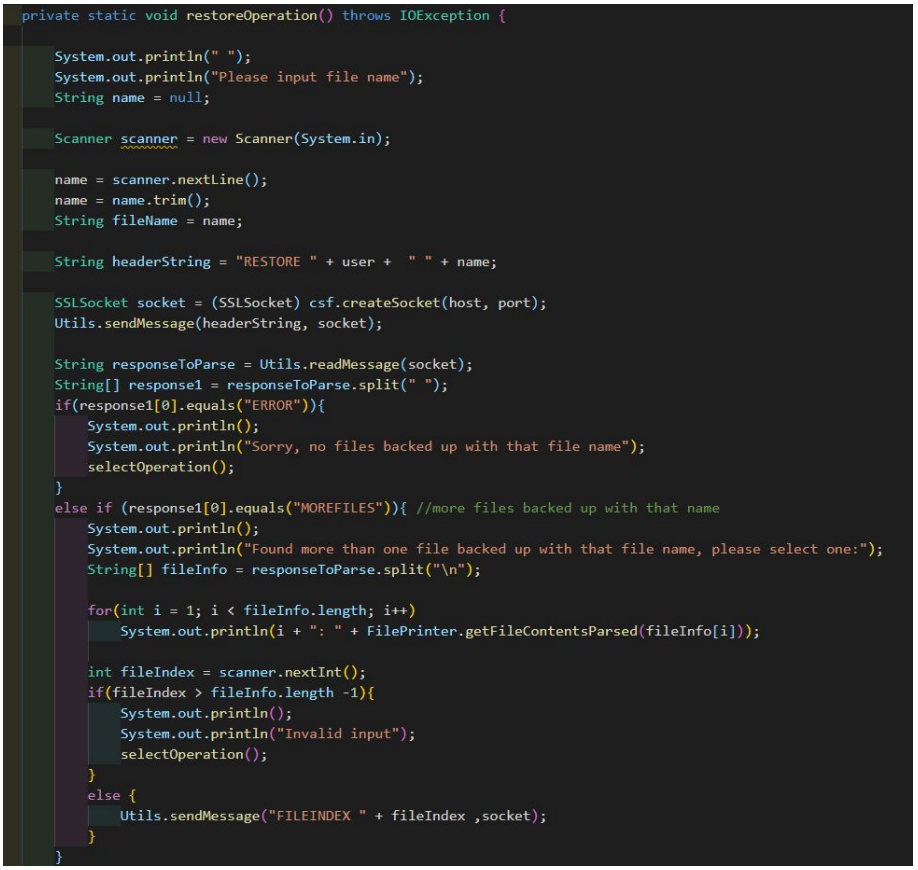
```java
    //check if server is ready to receive
    String response2 = Utils.readMessage(socket);
    String[] response2Ready = response2.split(" ");

    if (response2Ready[0].equals("READY"))
        Utils.loadAndWriteFileContent(toBackup, socket);

    //check if replication was possible
    String response3 = Utils.readMessage(socket);
    String[] response3Repli = response3.split(" ");

    System.out.println();
    if (Integer.parseInt(response3Repli[1]) < replicationDegree)
        System.out.println("Replicated the file in " + response3Repli[1] + " online peers");
    else System.out.println("File Backed up with desired replication");


    selectOperation();

}
```

# Restore

Caso um Cliente esteja logged-in no nosso sistema, este pode fazer Restore de um ficheiro. Para o fazer, o Cliente que está a iniciar o pedido tem de ser o mesmo que iniciou o pedido de Backup, caso contrário o pedido de Restore falha.

O protocolo de Restore é composto por 3 partes:

- A primeira, como referido atrás, consiste na verificação do pedido de Backup do mesmo ficheiro;
- A segunda, na escolha do ficheiro, ou seja, caso tenham sido feitos vários Backups de ficheiros com o mesmo nome mas conteúdos diferentes, numa situação de Restore, o Cliente vai ter mais do que uma opção de escolha. Desta forma, o sistema, através do subprotocolo GETFILES vai apresentar-lhe os diferentes ficheiros que existem no nosso sistema e o Cliente terá que escolher aquele que pretende.
- A terceira e última, o Servidor procura um Peer que tenha realizado Backup do ficheiro em questão, pedindo-lhe as informações necessárias e, com o Cliente, troca informação de forma a preparar o processo de criação do ficheiro, como receber o tamanho do ficheiro e o seu conteúdo.



```
private static void restoreOperation() throws IOException {

    System.out.println(" ");
    System.out.println("Please input file name");
    String name = null;

    Scanner scanner = new Scanner(System.in);

    name = scanner.nextLine();
    name = name.trim();
    String fileName = name;

    String headerString = "RESTORE " + user +  " " + name;

    SSLSocket socket = (SSLSocket) csf.createSocket(host, port);
    Utils.sendMessage(headerString, socket);

    String responseToParse = Utils.readMessage(socket);
    String[] response1 = responseToParse.split(" ");
    if(response1[0].equals("ERROR")){
        System.out.println();
        System.out.println("Sorry, no files backed up with that file name");
        selectOperation();
    }
    else if (response1[0].equals("MOREFILES")){ //more files backed up with that name
        System.out.println();
        System.out.println("Found more than one file backed up with that file name, please select one:");
        String[] fileInfo = responseToParse.split("\n");

        for(int i = 1; i < fileInfo.length; i++)
            System.out.println(i + ": " + FilePrinter.getFileContentsParsed(fileInfo[i]));

        int fileIndex = scanner.nextInt();
        if(fileIndex > fileInfo.length -1){
            System.out.println();
            System.out.println("Invalid input");
            selectOperation();
        }
        else {
            Utils.sendMessage("FILEINDEX " + fileIndex ,socket);
        }
    }
```

```
    }
    else if (response1[0].equals("ALLGOOD")){
        System.out.println("Found one file with that name");
    }


    //prepare to receive
    String response123 = Utils.readMessage(socket);
    int fileSize = Integer.parseInt(response123.split(" ")[1]);

    //load file and send it to server
    byte[] fileContent = Utils.readFileContent(socket, fileSize);

    String path = "Restore" + File.separator;

    File restoreFile = new File( path + fileName);

    String[] fileExtension = fileName.split("\\.");

    for (int i = 1; restoreFile.exists(); i++) {
        if(fileExtension[1].isEmpty())
            restoreFile = new File(path + String.format(fileExtension[0]+"(%d)", i));
        else
            restoreFile = new File(path + String.format(fileExtension[0]+"(%d)." + fileExtension[1], i));
    }

    if (!restoreFile.exists()) {
        try {
            restoreFile.getParentFile().mkdirs();
            restoreFile.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    try (FileOutputStream output = new FileOutputStream(restoreFile)) {
        output.write(fileContent);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        Utils.sendMessage("ERROR", socket);
```

```
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        Utils.sendMessage("ERROR", socket);
    } catch (IOException e) {
        e.printStackTrace();
        Utils.sendMessage("ERROR", socket);
    }

    System.out.println();
    System.out.println("Restore done with success");
    Utils.sendMessage("DONE", socket);
    selectOperation();
}
```

Todas as comunicações são feitas usando TCP com JSSE e o formato das mensagens trocadas neste protocolo são:

- RESTORE <username> <fileName>, do Cliente para o Servidor, de forma ao Servidor verificar se este <username> iniciou algum processo de Backup com este <fileName>.
- ERROR, do Servidor para o Cliente, caso este não tenha iniciado nenhum processo de Backup.

- ALLGOOD, do Servidor para o Cliente, caso exista um ficheiro com <fileName> que o Cliente tenha feito Backup.
- MOREFILES <<fileName> <lastModifiend> <creationTime> <fileSize>>, do Servidor para o Cliente, caso exista mais do que um ficheiros com <fileName> que o Cliente tenha feito Backup.
- FILEINDEX <fileIndex>, do Cliente para o Servidor, em que <fileIndex> é o index da lista de ficheiros recebidos pelo pedido MOREFILES.
- RESTORE <fileID>, do Cliente para o Peer, em que <fileID> é o nome codificado do ficheiro que está a ser guardado por todos os Peers que estão a fazer backup desse ficheiro.
- ERROR <fileHash>, do Peer para o Servidor, caso o <fileID> não exista na lista de ficheiros Backedup do Peer.
- <fileContent>, do Peer para o Servidor, em que <fileContent> é o conteúdo do ficheiro que vai ser Restored.
- READY <fileSize>, do Servidor para o Cliente, onde <fileSize> é o tamanho do ficheiro a fazer Restore.
- <fileContent>, do Servidor para o Cliente, em que <fileContent> é o conteúdo do ficheiro que vai ser Restored.
- ERROR, do Cliente para o Servidor, caso tenha ocorrido um problema a realizar o Restore.
- DONE, do Cliente para o Servidor, caso o Restore tenha sucedido.

# Delete

Caso um Cliente esteja logged-in no nosso sistema, este pode fazer Delete das cópias do ficheiro escolhido existentes em todos os Peers. Para o fazer, o Cliente que está a iniciar o pedido tem de ser o mesmo que iniciou o pedido de Backup, caso contrário o pedido de Delete falha.

O protocolo de Delete é composto por 3 partes:

- A primeira, como referido atrás, consiste na verificação do pedido de Backup do mesmo ficheiro;
- A segunda, na escolha do ficheiro, ou seja, caso tenham sido feitos vários Backups de ficheiros com o mesmo nome mas conteúdos diferentes, numa situação de Delete o Cliente vai ter mais do que uma opção de escolha. Desta forma, o sistema, através do subprotocolo GETFILES vai apresentar-lhe os diferentes ficheiros que existem no nosso sistema e o Cliente terá que escolher aquele que pretende.
- A terceira e última, o Servidor procura todos os Peers que tenham realizado Backup do ficheiro em questão e apaga-o.

```java
private static void deleteOperation() throws IOException {
    System.out.println();
    System.out.println("Please input the file name");
    String fileName = null;
    Scanner scanner = new Scanner(System.in);
    fileName = scanner.nextLine();
    fileName = fileName.trim();
    SSLSocket socket = (SSLSocket) csf.createSocket(host, port);
    String message = "DELETE " + user + " " + fileName;
    Utils.sendMessage(message, socket);
    String response = Utils.readMessage(socket);
    String[] parsedResponse = response.split(" ");

    if(parsedResponse[0].equals("NOFILE")){
        System.out.println();
        System.out.println("No files found with that file name");
        selectOperation();
    }
    else if (parsedResponse[0].equals("MOREFILES")){
        System.out.println();
        System.out.println("Found more than one file with that file name, please select one to delete");
        System.out.println();

        String[] fileInfos = response.split("\n");
        for(int i = 1; i < fileInfos.length; i++){
            System.out.println( i + ": " + FilePrinter.getFileContentsParsed(fileInfos[i]));
        }

        int fileIndex = scanner.nextInt();

        if(fileIndex > fileInfos.length || fileIndex <= 0){
            System.out.println();
            System.out.println("Invalid input");
            selectOperation();
        }

        Utils.sendMessage("FILEINDEX " + fileIndex, socket);
    }
    else {
        System.out.println();
        System.out.println("Found one file with that filename in our database");
    }

    String finalResponse = Utils.readMessage(socket);
    String[] finalM = finalResponse.split(" ");

    if(finalM[0].equals("FAILED")){
        System.out.println();
        System.out.println("Found your file at least in one peer, but somehow the peers don't have it stored in their filesystem");
        socket.close();
        selectOperation();
    }
    else{
        System.out.println();
        System.out.println("Success deleting from " + finalM[2] + " peers");
        socket.close();
        selectOperation();
    }
    selectOperation();
}
```

Todas as comunicações são feitas usando TCP com JSSE e o formato das mensagens trocadas neste protocolo são:

- DELETE <username> <fileName>, do Cliente para o Servidor, de forma ao Servidor verificar se este <username> iniciou algum processo de Backup com este <fileName>.
- NOFILE, do Servidor para o Cliente, caso este não tenha iniciado nenhum processo de Backup.
- ALLGOOD, do Servidor para o Cliente, caso exista um ficheiro com <fileName> que o Cliente tenha feito Backup.
- MOREFILES <<fileName> <lastModifiend> <creationTime> <fileSize>>, do Servidor para o Cliente, caso exista mais do que um ficheiros com <fileName> que o Cliente tenha feito Backup.
- FILEINDEX <fileIndex>, do Cliente para o Servidor, em que <fileIndex> é o index da lista de ficheiros recebidos pelo pedido MOREFILES.
- DELETE <fileID>, do Server para o Peer, em que <fileID> é o nome codificado do ficheiro que está a ser guardado por todos os Peers que estão a fazer backup desse ficheiro.
- NOFILE, do Peer para o Server, caso o <fileID> não exista na lista de ficheiros Backedup do Peer.
- DELETED <peerID> <fileHash>, do Peer para o Servidor, caso o ficheiro tenha sido apagado com sucesso.
- FAILED, do Servidor para o Cliente, caso o ficheiro não tenha sido apagado pelo Peer.
- SUCCESS <nPeers> <nFilesDeleted>, do Servidor para o Cliente, onde <nPeers> é o número de Peers processados pelo protocolo e <nFilesDeleted> é o número de ficheiros apagados pelo protocolo.

# Concurrency Design

Apesar de não serem utilizados thread-pools, a nossa implementação faz uso da criação de threads para cada interação existente no nosso sistema. Ou seja, é criada uma thread sempre que um Cliente faz um pedido ao Servidor, como também é sempre criada uma quando o Servidor interage com um Peer. Desta forma, conseguimos tratar vários pedidos simultaneamente, sem existirem pausas ou abrandamentos.

```
while (true) {
    SSLSocket clientSocket = (SSLSocket) serverSocket.accept();
    new Thread(new HandleConnection(this, clientSocket)).start();
}
```

# JSSE

Na nossa implementação usamos SSLSockets para que a comunicação seja segura, garantindo a integridade e a confidencialidade das mensagens trocadas entre o Servidor, o Cliente e os Peers.Também temos um sistema de autenticação para o cliente, em que a sua implementação já foi referida nos capítulos anteriores.

Abaixo temos alguns segmentos de código utilizado nesta parte da implementação.

```
private static SSLSocketFactory getSocketFactory() throws Exception {
    SSLContext ctx;
    KeyManagerFactory kmf;
    KeyStore ks;
    char[] passphrase = "sdis2020".toCharArray();

    ctx = SSLContext.getInstance("TLS");
    kmf = KeyManagerFactory.getInstance("SunX509");
    ks = KeyStore.getInstance("JKS");

    ks.load(new FileInputStream("keys/client.key"), passphrase);

    kmf.init(ks, passphrase);
    ctx.init(kmf.getKeyManagers(), null, null);

    return ctx.getSocketFactory();
}
```

```
SSLSocket socket = (SSLSocket) csf.createSocket(host, port);
String message = "CHECKUSERNAME " + requestedUsername;
Utils.sendMessage(message, socket);
String response = Utils.readMessage(socket);
```

# Fault-tolerance

Para conseguir garantir que a lista dos Peers conectados está sempre actualizada o Servidor a cada segundo manda uma mensagem a cada Peer, se este não responder, o Servidor remove esse Peer da lista dos Peers conectados.

```
private void start() throws IOException {
    System.out.println("Started Server: " + this.address.getHostAddress() + ":" + this.port);
    SSLServerSocket serverSocket = (SSLServerSocket) ssf.createServerSocket(this.port, 0, this.address);
    serverSocket.setNeedClientAuth(true);

    //fault tolerance
    this.scheduler = Executors.newSingleThreadScheduledExecutor();
    this.scheduler.scheduleAtFixedRate(new FaultToleranceTask(this), 0, 4, TimeUnit.SECONDS);


    while (true) {
        SSLSocket clientSocket = (SSLSocket) serverSocket.accept();
        new Thread(new HandleConnection(this, clientSocket)).start();
    }
}
```