# Programação distribuída

TRABALHO REALIZADO POR:

LEANDRO ADÃO FIDALGO, A2017017144

PEDRO DOS SANTOS ALVES, A2019112789

# Índice

- Aspetos não especificados no enunciado;
- Tratamento de erros;
- Consistência entre servidores;
- Sincronização de BD e ficheiros;
- Receção de vários ficheiros disseminados;
- Estrutura das mensagens trocadas;
- Threads;
- Arquitetura de software;
- Redireccionamento de um cliente;
- Registo;

# Índice

# Aspetos não especificados no enunciado

- Projeto comum onde as classes são partilhadas entre o servidor e o cliente (biblioteca).
  - Foram criados comandos para auxiliar os pedidos entre cliente-servidor e servidor-servidor.

# Tratamento de erros

- A maioria dos erros do cliente são demonstrados em dialogs.
- A maioria dos erros do servidor são demonstrados na consola.
- Abaixo está demonstrado a implementação do failover.

```
} catch (Exception ex) {
    //Mecanismo Failover
    ExceptionHandler.ShowException(ex);
    if (!App.CL_CFG.closing && !App.CL_CFG.ServerList.isEmpty()) {
        App.CL_CFG.server = App.CL_CFG.ServerList.get(0);
        App.connectionToServer(null);
    }else{
        System.exit(-1);
    }
}
```

# Tratamento de erros

- Mecanismo de heartbeat dos servers (ping).

```
InetAddress iAdd;
int udpPort;
int tcpPort;
String serverID;
long serverStart;
synchronized (SV_CFG) {
    iAdd = SV_CFG.ExternalIP;
    udpPort = SV_CFG.UDPPort;
    tcpPort = SV_CFG.TCPPort;
    serverID = SV_CFG.ServerID;
    serverStart = SV_CFG.ServerStart;
}
Server sv = new Server(serverID, serverStart, iAdd, udpPort, tcpPort, 0);
GenericPair<String, Server> svP = new GenericPair<>(SV_CFG.ServerID, sv);
Command cmd = new Command(ECommand.CMD_HEARTBEAT, svP);
while (true) {
    try {
        synchronized (SV_CFG) {
            svP.value.setUserCount(SV_CFG.Clients.size());
            SV_CFG.SetServersDead();
        }
        UDPHelper.SendMulticastCommand(mCS, iA, Port, cmd);
        Thread.sleep(ServerHeartbeatTimeout);
        synchronized (SV_CFG) {
            SV_CFG.RemoveDeadServers();
            SV_CFG.BroadcastServerList();
        }
    } catch (Exception ex) {
        ExceptionHandler.ShowException(ex);
    }
}
```

```
private void HandleHeartbeat(Command cmd) {
    // Add or update server info
    synchronized (SV_CFG) {
        SV_CFG.AddOrUpdateServer(((GenericPair<String, Server>) cmd.Body).value);
    }
}
```

```
public void AddOrUpdateServer(Server s) {
    int Index = ServerList.indexOf(s);
    if (Index == -1) {
        ServerList.add(s);
    } else {
        ServerList.get(Index).setUserCount(s.getUserCount());
        ServerList.get(Index).setAlive(true);
    }
    SortServerList();
}
```

# Consistência entre servidores

- Todos os pedidos do cliente, com a exceção do download, são transmitidos para os outros servidores através do multicast as alterações efetuadas para manter as bases de dados atualizadas e consistentes, assim como os ficheiros.

```
} else {
    // Send OK to the client
    sendCmd = new Command(ECommand.CMD_CREATED, user);
    TCPHelper.SendTCPCommand(oOS, sendCmd);
    Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
    // After created, the client should send the photo asynchronously
    // Announce to other servers via multicast
    synchronized (SV_CFG) {
        SV_CFG.MulticastMessage(new Command(ECommand.CMD_CREATED,
                new GenericPair<>(SV_CFG.ServerID, lastUser)));
    }
}
```

# Sincronização de BD e ficheiros

- A sincronização de um servidor que arranca e deteta a presença de outro(s) já operacional(ais).

```
synchronized (SV_CFG) {
    SV_CFG.MCSocket = mCS;
    SV_CFG.MCAddress = iA;
    Server thisSv = new Server(SV_CFG.ServerID, SV_CFG.ServerStart,
            SV_CFG.ExternalIP, SV_CFG.UDPPort, SV_CFG.TCPPort, 0);
    try {
        Main.Log("[Server]", "Looking for other servers online...");
        UDPHelper.SendMulticastCommand(mCS, iA, Port, new Command(ECommand.CMD_HELLO, new GenericPair<>(SV_CFG.ServerID, thisSv)));
        SV_CFG.wait(ServerLookupTimeout);
    } catch (Exception ex) {
    } finally {
        Main.Log("[Server]", "Seems like no other server is running...");
    }
    if (!SV_CFG.ServerList.isEmpty()) {
        SV_CFG.ServerList.sort(new ServerStartComparator());
        SV_CFG.SortServerList();
        Main.Log("[Server]", "Synchronizing...");
        // Initiate synchronization
        Server syncSv = SV_CFG.ServerList.get(0);
        // Erase DB and files
        SV_CFG.DB.devEraseDatabase();
        String BaseDir = SV_CFG.DBConnection.getSchema() + ExplorerController.BASE_DIR;
        File AvatarDir = new File(BaseDir + ExplorerController.AVATAR_SUBDIR);
        File[] AvatarFiles = AvatarDir.listFiles();
        for (File f : AvatarFiles) {
            f.delete();
        }
        File FilesDir = new File(BaseDir + ExplorerController.FILES_SUBDIR);
        File[] FilesFiles = FilesDir.listFiles();
        for (File f : FilesFiles) {
            f.delete();
        }
        // Ask server for synchronization
        UDPHelper.SendUDPCommand(mCS, syncSv.getAddress(), syncSv.getUDPPort(), new Command(ECommand.CMD_SYNC, new GenericPair<>(SV_CFG.ServerID, thisSv)));
        synchronized (SV_CFG.DB) {
            SV_CFG.DB.wait(ServerDBSyncTimeout);
        }
        Main.Log("[Server]", "Synchronization finished...");
    }
    SV_CFG.notifyAll();
}
```

# Receção de vários ficheiros disseminados

- Capacidade em processar a receção de vários ficheiros disseminados via UDP nos servidores.

```
private void HandleUpload(Command cmd) throws IOException, InterruptedException {
    // CMD_UPLOAD (FileChunk)
    GenericPair<String, FileChunk> gp = (GenericPair<String, FileChunk>) cmd.Body;
    FileChunk fc = gp.value;
    // Write to local files
    boolean hasGUID = (fc.getGUID() != null);
    ExplorerController.WriteFile(SV_CFG.DBConnection.getSchema(),
            !hasGUID ? ExplorerController.AVATAR_SUBDIR : ExplorerController.FILES_SUBDIR,
            !hasGUID ? fc.getUsername() + fc.getExtension() : fc.getGUID().toString() + fc.getExtension(),
            fc.getFilePart(),
            fc.getOffset(),
            fc.getLength());
}
```

```
public static void WriteFile(String DBName, String SubDir, String FileName, byte[] Bytes, long Offset, int Length)
    String BaseDir = DBName + BASE_DIR;
    _WriteFile(BaseDir + SubDir + "/" + FileName, Bytes, Offset, Length);
}

private static void _WriteFile(String Path, byte[] Bytes, long Offset, int Length) throws FileNotFoundException, IO
    try ( FileOutputStream f = new FileOutputStream(Path, true)) {
        if (Length > 0) {
            synchronized (f.getChannel()) {
                while (f.getChannel().position() < Offset) {
                    f.getChannel().wait(10);
                }
            }
            f.write(Bytes, 0, Length);
        }
    }
}
```

# Estrutura das mensagens trocadas

- Todos os pedidos entre cliente-servidor e servidor-servidor são efetuados através da classe Command.

```java
public class Command implements Serializable {

    public int CMD;
    public Object Body;

    @Override
    public String toString() {
        return "Command{" + "CMD=" + CMD + ", Body=" + Body + '}';
    }

    public Command() {
        this(ECommand.CMD_IGNORE);
    }

    public Command(int CMD) {
        this(CMD, null);
    }

    public Command(int CMD, Object Body) {
        this.CMD = CMD;
        this.Body = Body;
    }
}
```

# Threads

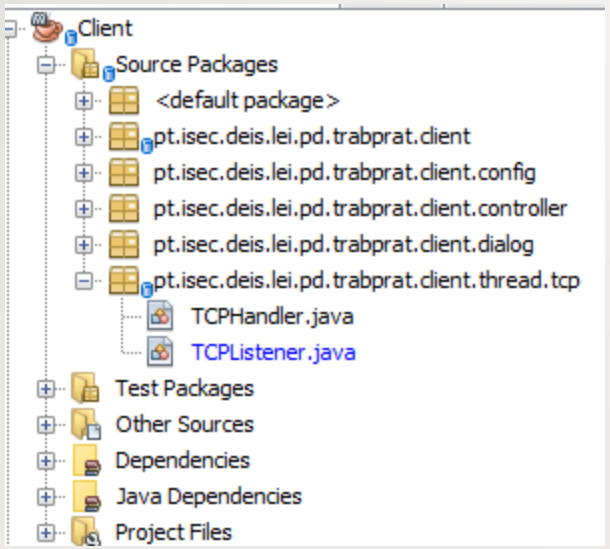
- Recurso a um conjunto apropriado de threads nos clientes e servidores para suportar diversas tarefas de um modo concorrente e assíncrono.

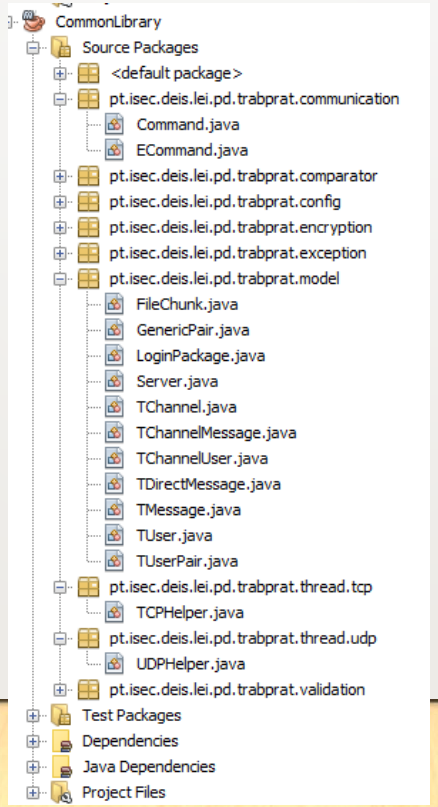
```
while (true) {
    Socket ClSocket = SvSocket.accept();
    IP = ClSocket.getInetAddress().getHostAddress() + ":" + ClSocket.getPort();
    Main.Log("Established connection with", IP);
    Thread td = new Thread(new TCPHandler(ClSocket, IP, SV_CFG));
    td.setDaemon(true);
    td.start();
}
```

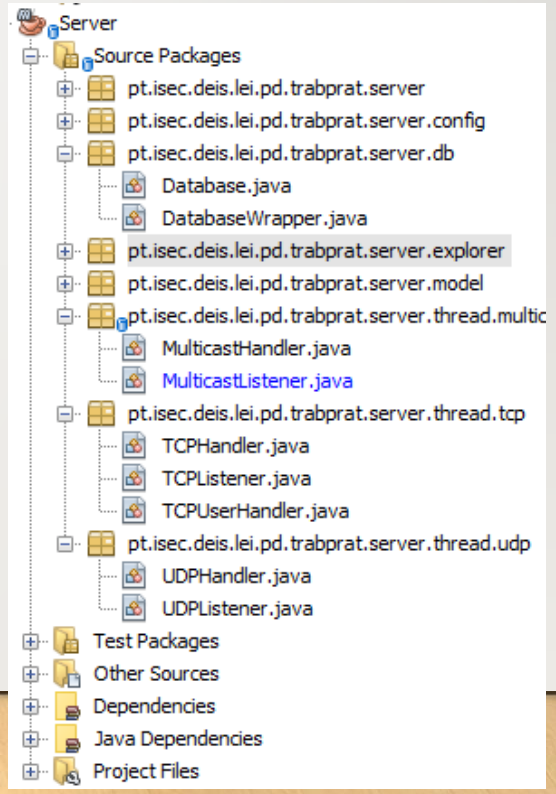
```
Thread[] threads = new Thread[]{
    new Thread(() -> {
        TdChannel();
    }),
    new Thread(() -> {
        TdMessages();
    }),
    new Thread(() -> {
        TdDM();
    }),
    new Thread(() -> {
        TdOnlineUsers();
    })
};
```

# Arquitetura de software

- Alguma separação, em termos da arquitetura de software, da lógica da comunicação (e das queries à BD).



# Redireccionamento de um cliente

- Redireccionamento de um cliente para um servidor com menos clientes ligados.

```
private void HandleConnect(String IP) throws IOException {
    // Ask other servers via multicast if they have less than 50% of the load (use lock)
    Command cmd;
    Server Accept = null; // Get response via multicast
    ArrayList<Server> body = new ArrayList<>();
    Server thisSv;
    synchronized (SV_CFG) {
        thisSv = new Server(SV_CFG.ServerID, SV_CFG.ServerStart, SV_CFG.ExternalIP, SV_CFG.UDPPort, SV_CFG.TCPPort, SV_CFG.Clients.size());
        Accept = thisSv;
        body.add(thisSv);//SV_CFG.ClientList.size()));
        body.addAll(SV_CFG.ServerList);
        if (body.size() > 1) {
            body.sort(SV_CFG.SvComp);
            Server lowestCapSv = body.get(0);
            if ((thisSv.getUserCount() * 0.5) >= lowestCapSv.getUserCount()) {
                Accept = lowestCapSv;
            }
        }
    }
    if (thisSv.equals(Accept)) {
        // Accepted
        cmd = new Command(ECommand.CMD_ACCEPTED, body);
        UDPHelper.SendUDPCommand(ServerSocket, Address, Port, cmd);
    } else {
        // Rejected
        cmd = new Command(ECommand.CMD_MOVED_PERMANENTLY, Accept);
        UDPHelper.SendUDPCommand(ServerSocket, Address, Port, cmd);
    }
    Main.Log("[Server] to " + IP, "" + cmd.CMD);
}
```

# Registo

```
} else {
    int extIndex = user.getUPhoto().lastIndexOf(".");
    String extension = "";
    if (extIndex != -1) {
        extension = user.getUPhoto().substring(extIndex);
    }
    ExplorerController.CreateUserDirectory(DBName, user.getUUsername());
    ExplorerController.Touch(DBName, ExplorerController.AVATAR_SUBDIR, user.getUUsername() + extension);
    String fullDir = DBName + ExplorerController.BASE_DIR + ExplorerController.AVATAR_SUBDIR + "/"
            + user.getUUsername() + extension;
    TUser insUser = new TUser(0, user.getUName(), user.getUUsername(), user.getUPassword(), fullDir, 0);
    TUser lastUser;
    int inserted;
    synchronized (SV_CFG) {
        inserted = db.insertUser(insUser);
        lastUser = db.getLastUser();
    }
    if (inserted <= 0) {
        // Tell client, server couldn't register
        sendCmd = new Command(ECommand.CMD_SERVICE_UNAVAILABLE, DefaultSvMsg.SV_INTERNAL_ERROR);
        TCPHelper.SendTCPCommand(oOS, sendCmd);
        Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
    } else {
        // Send OK to the client
        sendCmd = new Command(ECommand.CMD_CREATED, user);
        TCPHelper.SendTCPCommand(oOS, sendCmd);
        Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
        // After created, the client should send the photo asynchronously
        // Announce to other servers via multicast
        synchronized (SV_CFG) {
            SV_CFG.MulticastMessage(new Command(ECommand.CMD_CREATED,
                    new GenericPair<>(SV_CFG.ServerID, lastUser)));
        }
    }
}
```

- Primeiro verifica se o utilizador já existe, de seguida caso este não exista, cria o utilizador tal como demonstrado na figura abaixo.

# Autenticação

```
if (user.getUPassword().equals(info.getUPassword())) {
    // OK
    boolean LoggedIn;
    info.setPassword();
    var c = new GenericPair<TUser, ObjectOutputStream>(info, oOS);
    synchronized (SV_CFG) {
        LoggedIn = SV_CFG.ClientListContains(c);
    }
    if (!LoggedIn) {
        // Send channel list, online users, DMs
        LoginPackage lp = new LoginPackage(info);
        synchronized (SV_CFG) {
            lp.Users.addAll(SV_CFG.GetAllOnlineUsers());
            var channels = db.getAllChannels();
            lp.Channels.addAll(channels);
            var dms = db.getAllDMByUserID(info.getUID());
            lp.DMUsers.addAll(db.getOtherUserFromDM(dms, info));
            var channelUsers = db.getAllChannelUsers();
            lp.ChannelUsers.addAll(channelUsers);
        }

        sendCmd = new Command(ECommand.CMD_LOGIN, lp);
        TCPHelper.SendTCPCommand(oOS, sendCmd);
        Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
        Main.Log("[User: (" + info.getUID() + ") " + info.getUUsername() + "]", "has logged in.");
        // Add user to the client list
        synchronized (SV_CFG) {
            SV_CFG.Clients.put(UserSocket, c);
            // Send to other users that the list of users has been updated
            SV_CFG.BroadcastOnlineActivity();
            // Announce to other servers via multicast
            SV_CFG.MulticastMessage(new Command(ECommand.CMD_LOGIN,
                    new GenericPair<>(SV_CFG.ServerID, info)));
        }
    } else {
        // User already logged in
        sendCmd = new Command(ECommand.CMD_UNAUTHORIZED, DefaultSvMsg.SV_USER_LOGGED_IN);
        TCPHelper.SendTCPCommand(oOS, sendCmd);
        Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
    }
} else {
    // Password doesn't match
    sendCmd = new Command(ECommand.CMD_UNAUTHORIZED, DefaultSvMsg.SV_PASSWORD_DOES_NOT_MATCH);
    TCPHelper.SendTCPCommand(oOS, sendCmd);
    Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
}
```

- Primeiro verifica se o utilizador já existe, de seguida verifica se a password está correta, finalmente verifica se o utilizador já fez login.

# Canais

- Criação, edição e eliminação de canais, apenas pelos respetivos criadores. Abaixo segue o exemplo da criação de um canal.

```
private void HandleCreateChannel() throws IOException {
    // Insert channel into database
    DatabaseWrapper db;
    TChannel channel = (TChannel) Cmd.Body;
    ArrayList<TChannel> c;
    TChannel lastChannel;
    int ErrorNumber;
    synchronized (SV_CFG) {
        db = SV_CFG.DB;
        ErrorNumber = db.insertChannel(channel);
        if (ErrorNumber > 0) {
            // Success
            c = db.getAllChannels();
            lastChannel = db.getLastChannel();
            // Broadcast new channel created to all users
            SV_CFG.BroadcastMessage(new Command(ECommand.CMD_CREATED, c));
            // Send through multicast
            SV_CFG.MulticastMessage(new Command(ECommand.CMD_CREATED,
                    new GenericPair<>(SV_CFG.ServerID, lastChannel)));
        } else {
            // Operation failed
            TCPHelper.SendTCPCommand(oOS, new Command(ECommand.CMD_BAD_REQUEST, DefaultSvMsg.SV_CREATE_CHANNEL_FAIL));
            Main.Log("[Server] to " + IP, "" + ECommand.CMD_BAD_REQUEST);
        }
    }
}
```

# Aderir a um canal

- Primeiro verifica se o utilizador é o criador do canal ou se não existe password ou se já se encontra neste canal, senão verifica se a password introduzida está correta.

```
public static boolean ShowDialog2(TChannel tchannel) throws Exception {
    if (tchannel.getCUID().equals(App.CL_CFG.MyUser) || tchannel.getCPassword() == null) {
        return true;
    }
    synchronized (App.CL_CFG.LockCU) {
        for (int i = 0; i < App.CL_CFG.ChannelUsers.size(); i++) {
            if (App.CL_CFG.ChannelUsers.get(i).getCID().equals(tchannel)
                    && App.CL_CFG.ChannelUsers.get(i).getUID().equals(App.CL_CFG.MyUser)) {
                return true;
            }
        }
    }
    TextInputDialog dialog = new TextInputDialog();
    dialog.setHeaderText(null);
    dialog.setTitle("Password of channel");
    dialog.setContentText("Please enter the password of channel:");
    Optional<String> result = dialog.showAndWait();
    if (result.isPresent()) {
        return AES.Encrypt(result.get()).equals(tchannel.getCPassword());
    }
    return false;
}
```

# Envio e receção de mensagens

- Envio e receção de mensagens (individual e canal).

```
if (dm == null) {
    // Channel Message
    if (cm.getMID().getMPath() != null) {
        int extIndex = cm.getMID().getMText().lastIndexOf(".");
        String Extension = "";
        if (extIndex != -1) {
            Extension = cm.getMID().getMText().substring(extIndex);
        }
        synchronized (SV_CFG) {
            String InternalPath = ExplorerController.FILES_SUBDIR
                    + "/" + cm.getMID().getMPath() + Extension;
            msg = new TMessage(0, cm.getMID().getMUID(), cm.getMID().getMText(), InternalPath, 0);
        }
    } else {
        msg = cm.getMID();
    }
    synchronized (SV_CFG) {
        i += db.insertChannelMessage(cm.getCID(), msg);
        if (i > 0) {
            cmL = db.getAllMessagesFromChannelID(cm.getCID().getCID());
            lastMessage = db.getLastMessage();
            TChannelMessage lastCM = new TChannelMessage(cm.getCID(), lastMessage);
            // Broadcast new message to all users
            SV_CFG.BroadcastMessage(new Command(ECommand.CMD_CREATED, cmL));
            // Send through multicast
            SV_CFG.MulticastMessage(new Command(ECommand.CMD_CREATED,
                    new GenericPair<>(SV_CFG.ServerID, lastCM)));
        } else {
            sendCmd = new Command(ECommand.CMD_BAD_REQUEST, DefaultSvMsg.SV_MESSAGE_FAIL);
        }
    }
} else {
```

```
} else {
    // Direct Message
    if (dm.getMID().getMPath() != null) {
        int extIndex = dm.getMID().getMText().lastIndexOf(".");
        String Extension = "";
        if (extIndex != -1) {
            Extension = dm.getMID().getMText().substring(extIndex);
        }
        synchronized (SV_CFG) {
            String InternalPath = ExplorerController.FILES_SUBDIR
                    + "/" + dm.getMID().getMPath() + Extension;
            msg = new TMessage(0, dm.getMID().getMUID(), dm.getMID().getMText(), InternalPath, 0);
        }
    } else {
        msg = dm.getMID();
    }
    synchronized (SV_CFG) {
        var temp = dm.getUID();
        if (dm.getUID().getUID() == 0) {
            temp = db.getUserByUsername(dm.getUID().getUUsername());
            if (temp == null) {
                temp = db.getUserByName(dm.getUID().getUUsername());
            }
        }
        dm = new TDirectMessage(msg, temp);
        i += db.insertDirectMessage(dm.getUID(), msg);
        if (i > 0) {
            lastMessage = db.getLastMessage();
            TDirectMessage lastDM = new TDirectMessage(lastMessage, dm.getUID());
            dmL = db.getAllDMByUserIDAndOtherID(dm.getMID().getMUID().getUID(), dm.getUID().getUID());
            // Broadcast new message to all users
            dmU = db.getOtherUserFromDM(db.getAllDMByUserID(dm.getUID().getUID()), dm.getUID());
            sendCmd = new Command(ECommand.CMD_CREATED, new GenericPair<>(dmL, dmU));
            var ou = SV_CFG.GetUser(dm.getUID());
            if (ou != null) {
                TCPHelper.SendTCPCommand(ou.value, sendCmd);
            }
            dmU = db.getOtherUserFromDM(db.getAllDMByUserID(dm.getMID().getMUID().getUID()), dm.getMID().getMUID());
            sendCmd.Body = new GenericPair<>(dmL, dmU);
            // Send through multicast
            SV_CFG.MulticastMessage(new Command(ECommand.CMD_CREATED,
                    new GenericPair<>(SV_CFG.ServerID, lastDM)));
        } else {
            sendCmd = new Command(ECommand.CMD_BAD_REQUEST, DefaultSvMsg.SV_MESSAGE_FAIL);
        }
    }
}
```

# Consultar mensagens

- Consultar todas as mensagens incluindo os ficheiros (canal ou individual).

```
private void HandleGetChannelMessages() throws IOException {
    // Add channel user if they don't exist
    DatabaseWrapper db;
    TChannelUser cU = (TChannelUser) Cmd.Body;
    ArrayList<TChannelMessage> messages = new ArrayList<>();
    Command sendCmd;
    synchronized (SV_CFG) {
        db = SV_CFG.DB;
        if (!db.doesUserBelongToChannel(cU.getCID(), cU.getUID())) {
            db.insertChannelUser(cU.getCID(), cU.getUID());
            var cUs = db.getAllChannelUsers();
            SV_CFG.BroadcastMessage(new Command(ECommand.CMD_UPDATE_CHANNEL_USERS, cUs));
            // Send through multicast
            SV_CFG.MulticastMessage(new Command(ECommand.CMD_GET_CHANNEL_MESSAGES,
                    new GenericPair<>(SV_CFG.ServerID, cU)));
        }
        messages.addAll(db.getAllMessagesFromChannelID(cU.getCID().getCID()));
    }
    // Send messages from channel
    sendCmd = new Command(ECommand.CMD_GET_CHANNEL_MESSAGES, messages);
    TCPHelper.SendTCPCommand(oOS, sendCmd);
    Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
}
```

```
private void HandleGetDMMessages() throws IOException {
    DatabaseWrapper db;
    TUserPair pair = (TUserPair) Cmd.Body;
    Command sendCmd;
    ArrayList<TDirectMessage> DMs;
    synchronized (SV_CFG) {
        db = SV_CFG.DB;
        DMs = db.getAllDMByUserIDAndOtherID(pair.User1.getUID(), pair.User2.getUID());
    }
    sendCmd = new Command(ECommand.CMD_GET_DM_MESSAGES, DMs);
    TCPHelper.SendTCPCommand(oOS, sendCmd);
    Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
}
```

# Gestão de ficheiros com nomes idênticos

- Para que cada ficheiro enviado para o servidor possa ter nomes iguais decidimos utilizar um identificador único universal (UUID).

# Upload de ficheiros

- Upload de ficheiros pelos clientes nos respetivos servidores (individual e grupo).

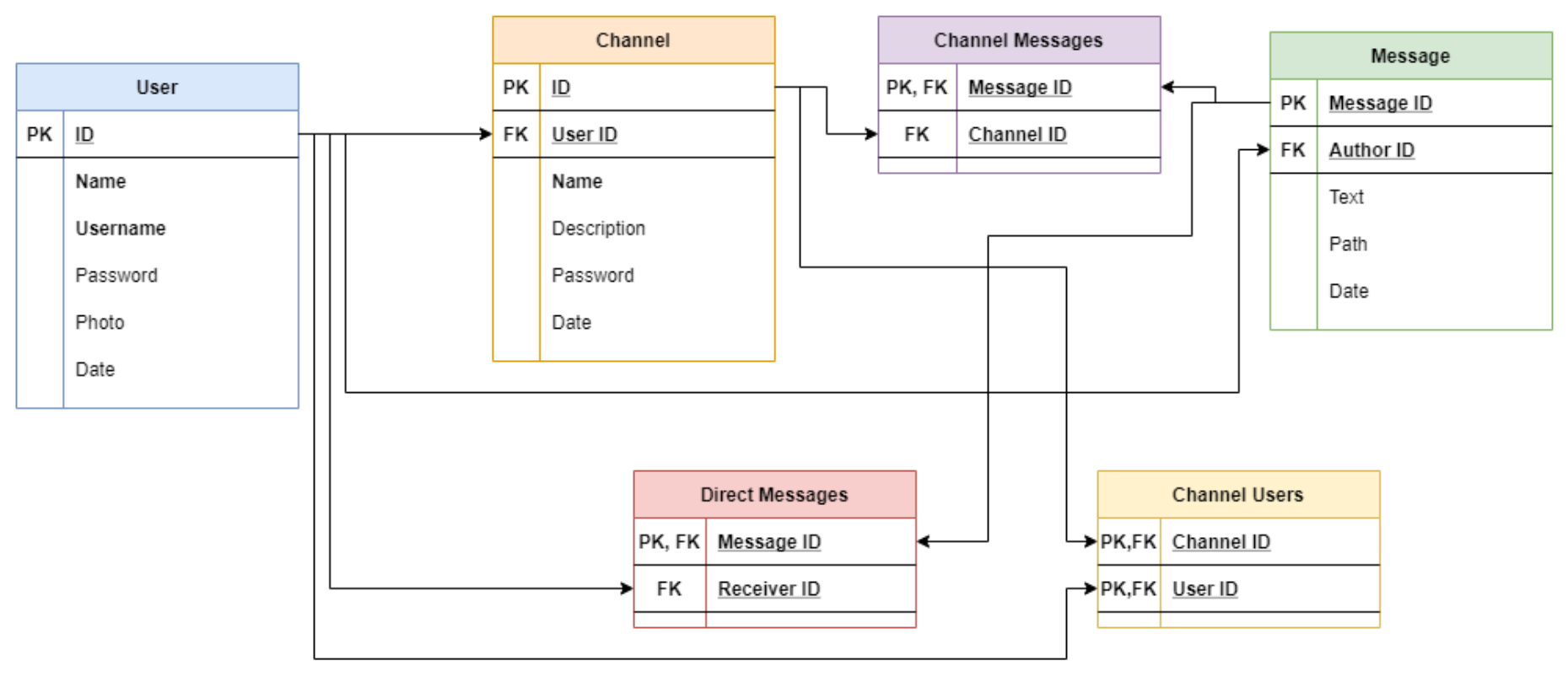
```
private void HandleUpload() throws IOException {
    // Check if user is in database
    DatabaseWrapper db;
    FileChunk fc = (FileChunk) Cmd.Body;
    TUser user;
    synchronized (SV_CFG) {
        db = SV_CFG.DB;
        user = db.getUserByUsername(fc.getUsername());
    }
    if (user != null) {
        try {
            // Write File
            boolean hasGUID = (fc.getGUID() != null);
            ExplorerController.WriteFile(SV_CFG.DBConnection.getSchema(),
                    !hasGUID ? ExplorerController.AVATAR_SUBDIR : ExplorerController.FILES_SUBDIR,
                    !hasGUID ? fc.getUsername() + fc.getExtension() : fc.getGUID().toString() + fc.getExtension(),
                    fc.getFilePart(),
                    fc.getOffset(),
                    fc.getLength());
            // Send through multicast
            synchronized (SV_CFG) {
                SV_CFG.MulticastMessage(new Command(ECommand.CMD_UPLOAD,
                        new GenericPair<>(SV_CFG.ServerID, fc)));
            }
        } catch (Exception ex) {
            ExceptionHandler.ShowException(ex);
        }
    } else {
        Command sendCmd = new Command(ECommand.CMD_FORBIDDEN);
        TCPHelper.SendTCPCommand(oOS, sendCmd);
        Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
    }
}
```

# Download de ficheiros

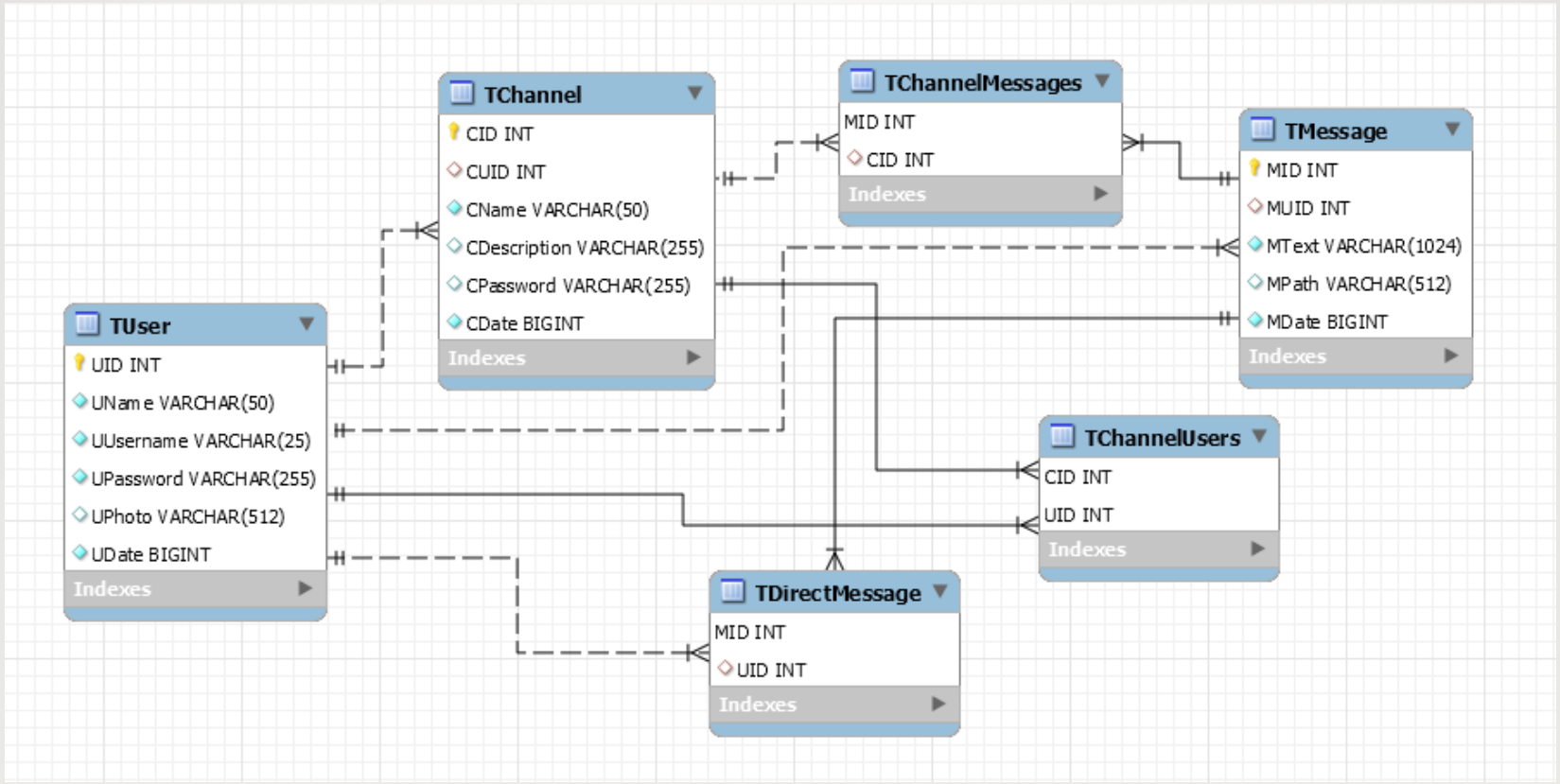
```
private void HandleDownload() throws IOException {
    // Get TMessage or TUser
    String Path = null;
    String _Username = null;
    Command sendCmd;
    if (Cmd.Body instanceof TMessage) {
        TMessage msg = (TMessage) Cmd.Body;
        Path = msg.getMPath();
        _Username = msg.getMText();
    } else if (Cmd.Body instanceof TUser) {
        TUser usr = (TUser) Cmd.Body;
        Path = usr.getUPhoto();
        _Username = usr.getUUsername();
    }
    if (Path != null) {
        // Send file to user
        FileChunk fc;
        try {
            String BaseDir = SV_CFG.DBConnection.getSchema() + ExplorerController.BASE_DIR;
            Path = BaseDir + Path;
            int extIndex = Path.lastIndexOf(".");
            String Extension = "";
            if (extIndex != -1) {
                Extension = Path.substring(extIndex);
            }
            int Length = DefaultConfig.DEFAULT_TCP_PACKET_SIZE;
            int Offset = 0;
            byte[] buffer = ExplorerController._ReadFile(Path, Offset, Length);
            while (buffer.length > 0) {
                fc = new FileChunk(buffer, Offset, buffer.length, _Username, null, Extension);
                sendCmd = new Command(ECommand.CMD_DOWNLOAD, fc);
                TCPHelper.SendTCPCommand(oOS, sendCmd);
                Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
                Offset += Length;
                buffer = ExplorerController._ReadFile(Path, Offset, Length);
            }
            fc = new FileChunk(null, 0, 0, _Username, null, Extension);
            sendCmd = new Command(ECommand.CMD_DOWNLOAD, fc);
            TCPHelper.SendTCPCommand(oOS, sendCmd);
            Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
        } catch (Exception ex) {
            sendCmd = new Command(ECommand.CMD_SERVICE_UNAVAILABLE, DefaultSvMsg.SV_DOWNLOAD_FILE_FAIL2);
            TCPHelper.SendTCPCommand(oOS, sendCmd);
            Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
        }
    } else {
        sendCmd = new Command(ECommand.CMD_BAD_REQUEST, DefaultSvMsg.SV_DOWNLOAD_FILE_FAIL);
        TCPHelper.SendTCPCommand(oOS, sendCmd);
        Main.Log("[Server] to " + IP, "" + sendCmd.CMD);
    }
}
```

- Download de ficheiros pelos clientes (individual ou de um grupo a que pertence).

# Modelo lógico da base de dados

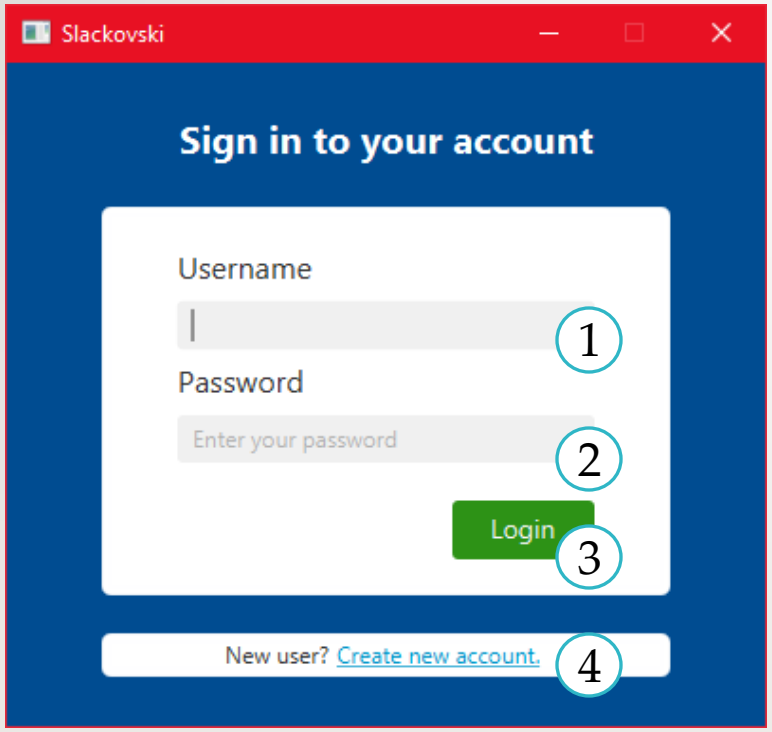


# Modelo físico da base de dados



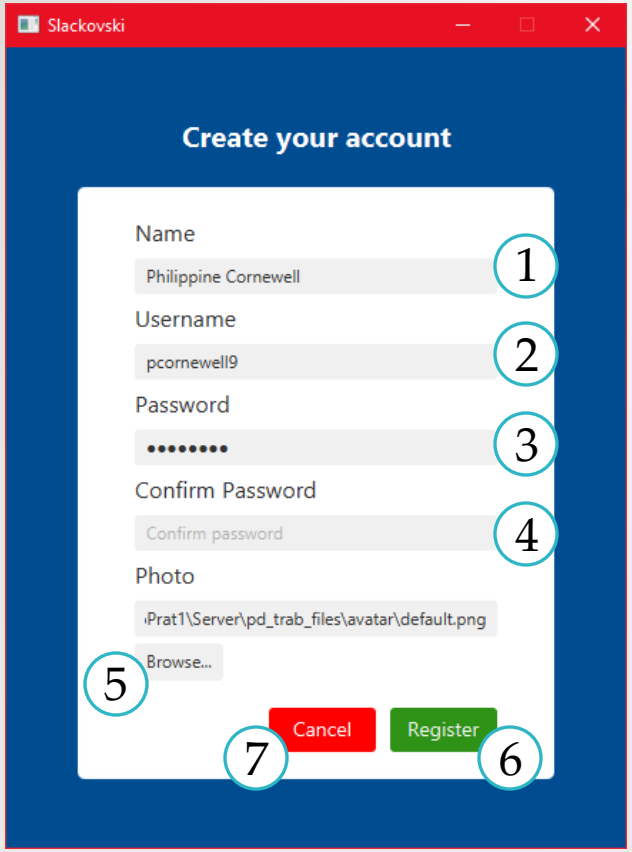
# Manual de utilizador

# Login



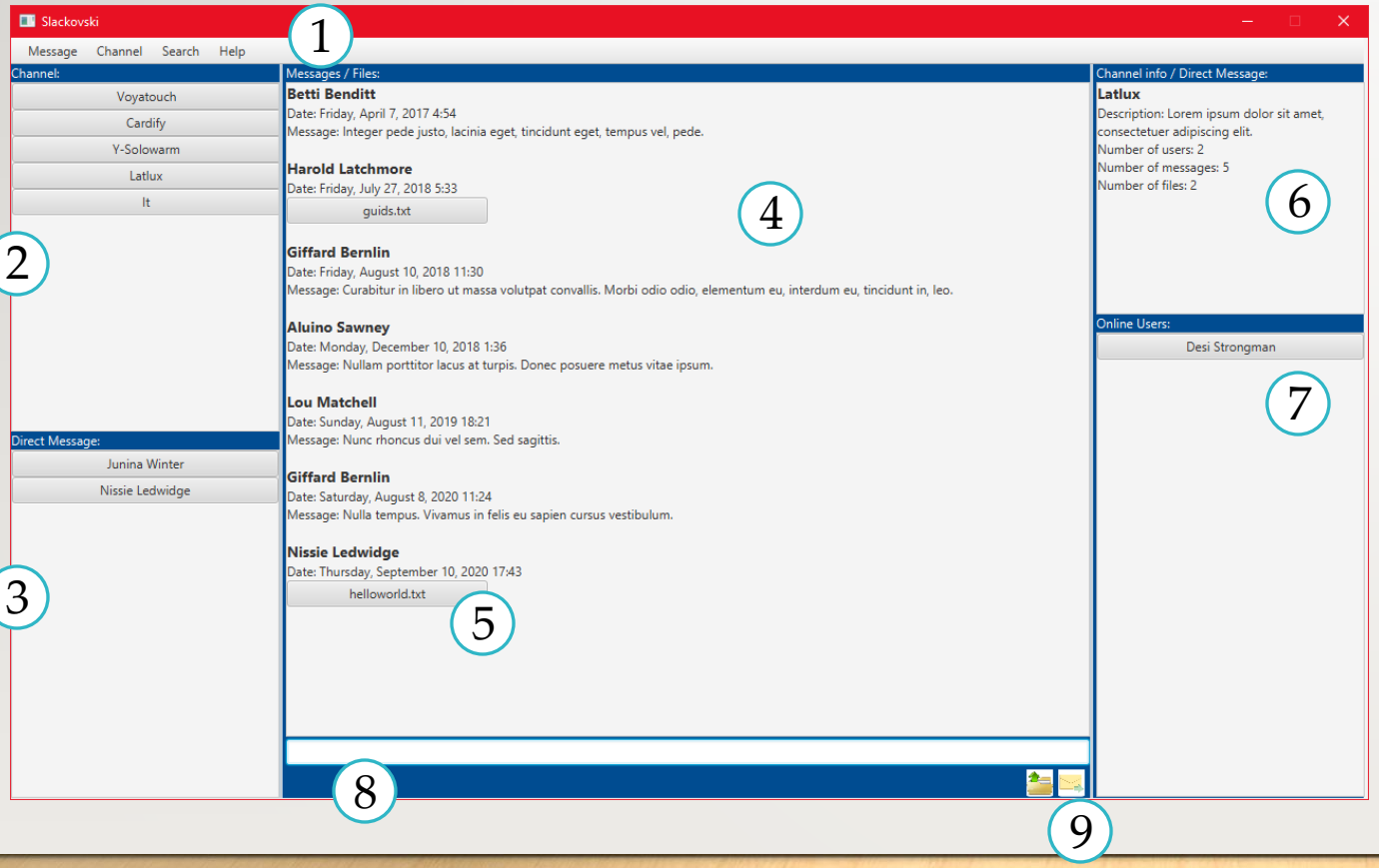

1. Caixa de texto do username
   - O utilizador introduz o seu username
2. Caixa de texto da password
   - O utilizador introduz a sua password
3. Botão de login
   - Após a introdução de ambos, o username e a password, o utilizador poderá efetuar o login
4. Botão de registo
   - Se o utilizador não estiver ainda registado, poderá faze-lo através desta opção

# Registo



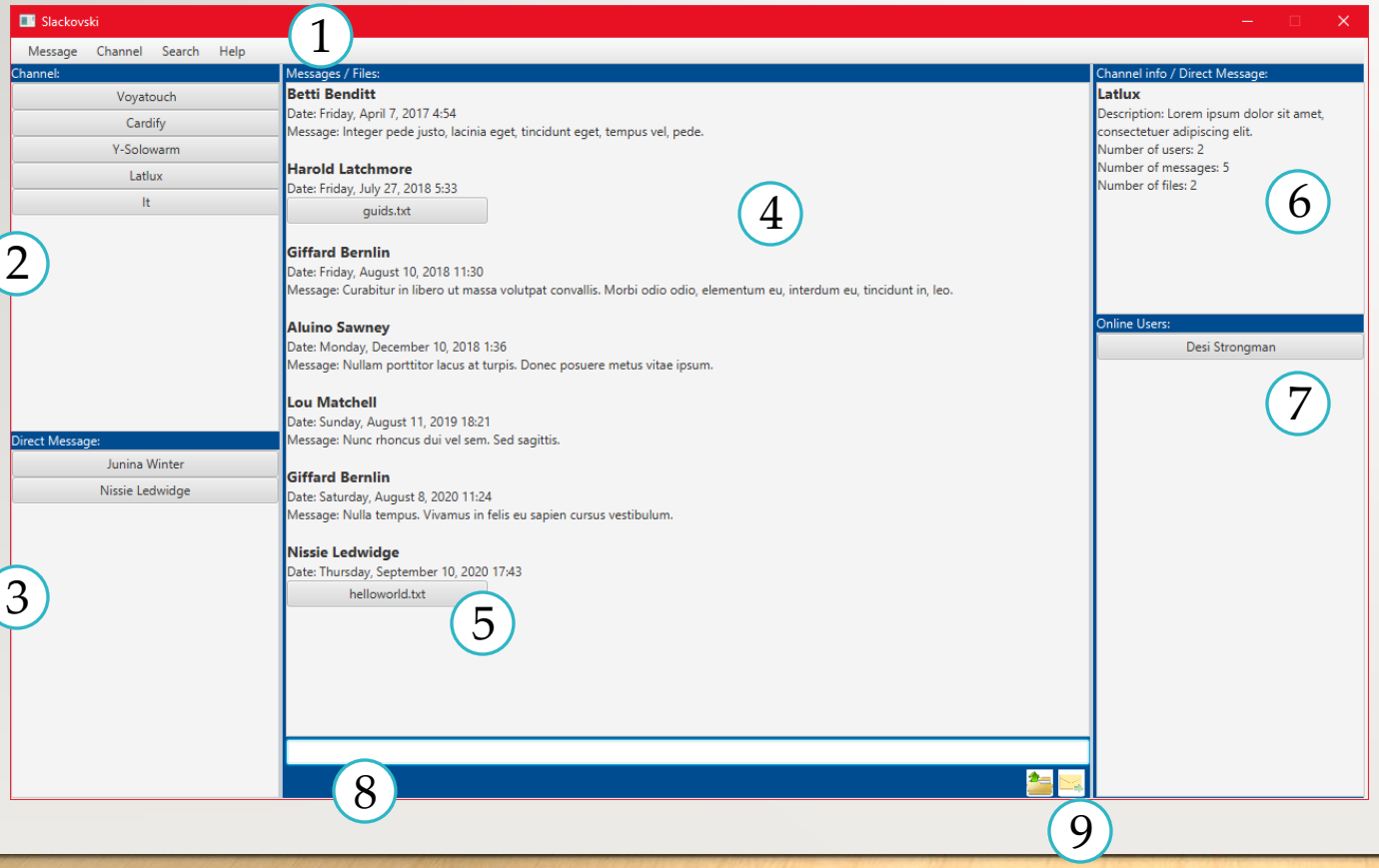

1. Caixa de texto do nome do utilizador
   - O utilizador introduz o seu nome
2. Caixa de texto do username
   - O utilizador introduz o seu username
3. Caixa de texto da password
   - O utilizador introduz a sua password
4. Caixa de texto da confirmação da password
   - O utilizador confirma a sua password
5. Botão de seleção de fotografia
   - O utilizador seleciona uma fotografia (jpg, jpeg ou png)
6. Botão de registo
   - Após preencher os campos todos, o utilizador poderá registar-se
7. Botão de cancelar
   - O utilizador poderá cancelar o seu registo

# Janela principal




1. Menu principal
   - Várias opções serão dispostas ao utilizador
2. Lista de canais
   - Todos os canais disponíveis no servidor
3. Lista de mensagens diretas
   - Todas as mensagens trocadas com outros utilizadores
4. Chat
   - Área de texto e ficheiros partilhados
   - Também poderá ser arrastado um ficheiro para o enviar
5. Link de download de ficheiro
   - Ao clicar nesse botão, irá fazer download do ficheiro

# Janela principal

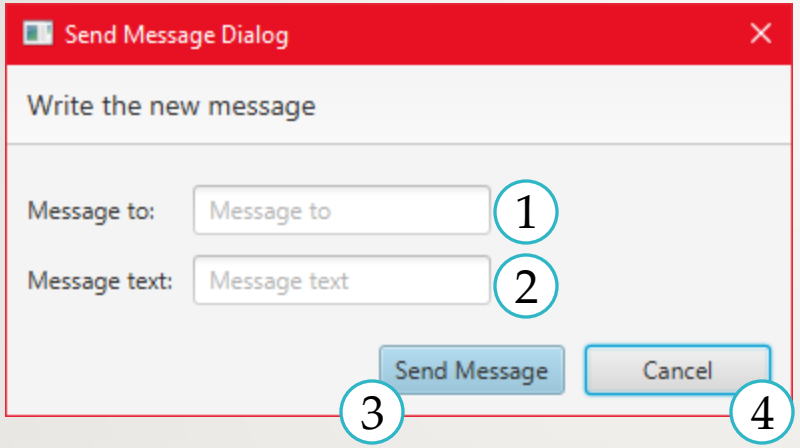



6. Informação sobre o canal ou mensagem direta
   - Disponibiliza as informações acerca do determinado canal ou mensagem direta
7. Utilizadores online
   - Todos os utilizadores que tiverem efetuado o login, através dos multiplos servidores
8. Caixa de texto para escrever mensagens
   - Poderá ser inserido texto para ser enviado
9. Botões de envio
   - O botão da esquerda serve para enviar ficheiros
   - O botão da direita serve para enviar o texto introduzido na caixa de texto

# Menu principal

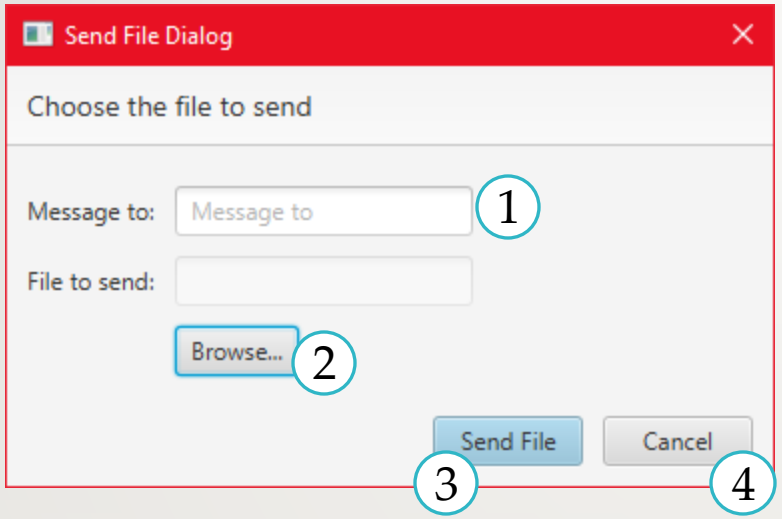

1. Message
    1. Send Message: Envia uma mensagem direta a um utilizador especificado
    2. Send File: Envia um ficheiro a um utilizador especificado, por mensagem direta
2. Channel
    1. Add: Adiciona um canal com um nome, descrição e password
3. Search
    1. Search Users: Procura por um utilizador que tenha um nome ou username idêntico ao introduzido
4. Help
    1. About: Apresenta informação sobre o programa (quem o fez, para que disciplina)

# Enviar mensagem direta



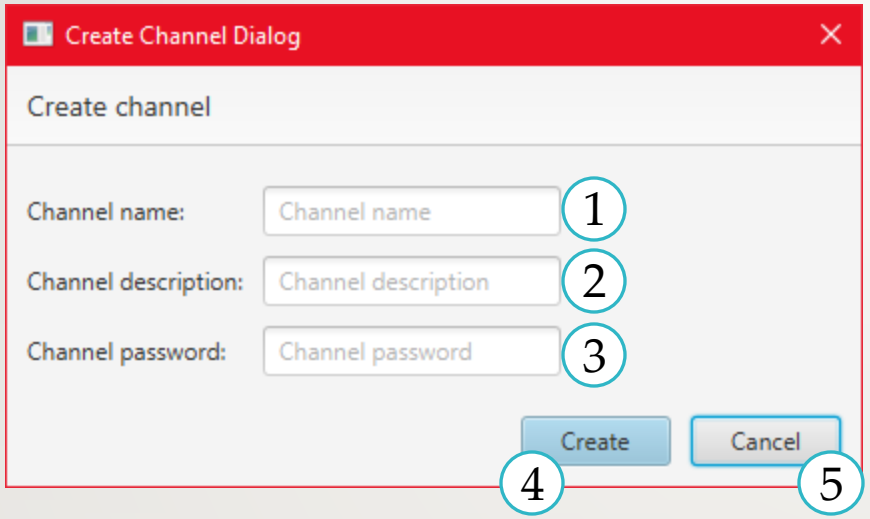

1. Caixa de texto do destinatário
   - O utilizador introduz o nome ou username do destinatário
2. Caixa de texto da mensagem
   - O utilizador introduz o texto a ser enviado ao destinatário
3. Botão de envio
   - Envia a mensagem
4. Botão de cancelar
   - Cancela o envio

# Enviar ficheiro apenas a um utilizador



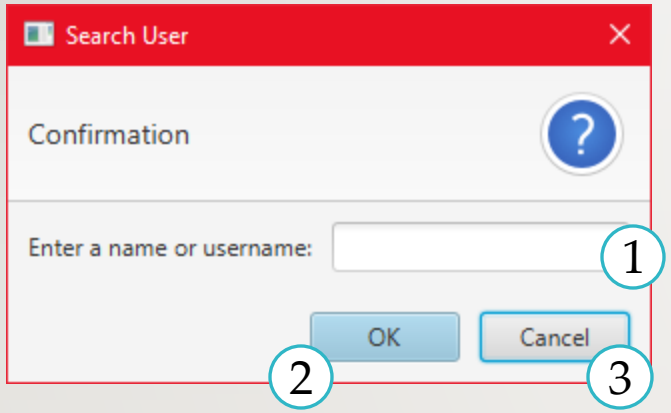

1. Caixa de texto do destinatário
   - O utilizador introduz o nome ou username do destinatário
2. Botão de seleção do ficheiro
   - O utilizador seleciona um ficheiro a enviar ao destinatário
3. Botão de envio
   - Envia o ficheiro
4. Botão de cancelar
   - Cancela o envio

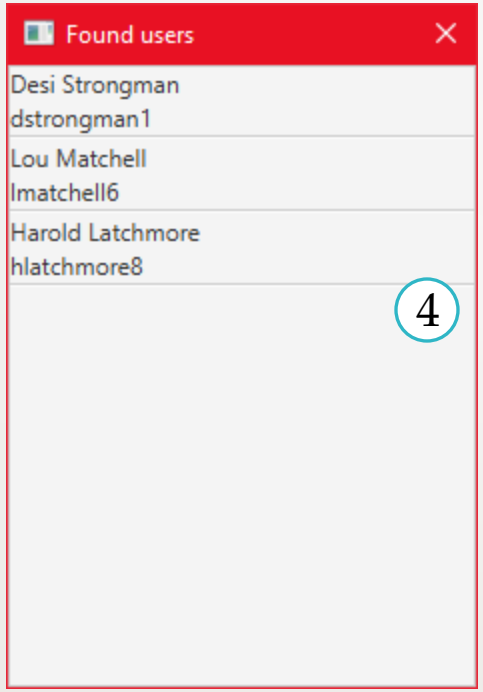
# Adicionar canal




1. Caixa de texto do nome do canal
   - O utilizador introduz um nome para o canal a ser criado
2. Caixa de texto da descrição do canal
   - O utilizador introduz uma descrição para o canal a ser criado
3. Caixa de texto da password do canal
   - O utilizador poderá ou não introduzir uma password para o canal a ser criado
4. Botão de criação de canal
   - Cria o canal consoante os dados introduzidos
5. Botão de cancelar
   - Cancela a criação do canal

# Procurar utilizadores




1. Caixa de texto de nome ou username
   - O utilizador introduz parte de um nome ou username
2. Botão de procura
   - Efetua a procura de todos os utilizadores consoante o texto introduzido
3. Botão de cancelar
   - Cancela a procura
4. Lista de utilizadores encontrados
   - Listagem de todos os utilizadores encontrados na base de dados (ex.: todos os utilizadores com "m" no nome ou username)