

CHATW.

Alexis Andrés Rodríguez Rodríguez

CHATW

CICLO SUPERIOR D.A.M

(Desarrollo de Aplicación Multiplataforma)

Alexis Andrés Rodríguez Rodríguez



Indice:

Capítulo 1º. Resumen

Capítulo 2º. Introducción

Capítulo 3º. Objetivos y características del proyecto

Capítulo 4º. Finalidad

Capítulo 5º. Medios materiales usados

Capítulo 6º. Planificación del proyecto

- Inicio
- Grupos
- Solicitudes
- Chat privado

Capítulo 7º. Fase de pruebas

Capítulo 8º. Conclusiones y trabajos futuros o posibles mejoras

Capítulo 9º. Apéndices

- Código importante
- Manual de usuario
- Manual de instalación

Capítulo 1º. Resumen

Para la creación de este programa se ha tenido en cuenta la necesidad de que un grupo de personas, por ejemplo, Juan, Emmanuel, Alexis y Andrés se quieren comunicar de forma instantánea y exclusiva por mensajes de texto privados, ya que el administrador de la plataforma y base de datos es el que permite la creación de los usuarios. Además de los mensajes de texto, necesitan enviarse documentos en formato pdf, word e imágenes, pudiéndose tener la opción de eliminar los mensajes recibidos o enviados entre ellos. También se plantea la opción de crear grupos para comunicarse con varios usuarios. Para tener una gestión clara de los mensajes, se necesita que se indique la hora y fecha, así como la creación de perfiles donde se pueda establecer el estado de cada usuario.

CHATW.

Alexis Andrés Rodríguez Rodríguez

Capítulo 2º.Introducción

En este proyecto integrado se realiza un programa de mensajería instantánea, con el nombre de *ChatW*. Esta aplicación se realizara utilizando el entorno de desarrollo IDE de Android estudio y complementado con la base de datos de *Firebase*, la cual, es válida para este proyecto, aunque tenga una limitación de funciones, al ser gratuito. En esta aplicación habrá dos tipos de chats, los grupales, entre varias personas, en los cuales se podrá crear dichos grupos para una comunicación más general. Los privados, donde sólo se comunican dos personas, para una comunicación más individual en el que tendrá la capacidad de enviar documentos en formato imagen, pdf y word, ademas de eliminar los mensajes que se deseen. Por último, tenemos que hablar de la gestión de perfiles, tanto en la creación como en la actualización de los mismos. La creación de los usuarios es bajo la supervisión del administrador del programa y base de datos, para que su uso sea de forma individualizada y restrictiva.

Capítulo 3º. Objetivos y características del proyecto

Los objetos que tendrá que afrontar la aplicación serán los siguientes:

- Guardado de información, tanto de los usuarios como de los mensajes que recibirá la aplicación en una base de datos, que en nuestro caso es la *Firebase*.
- Uso de Perfiles de usuario, los cuales se podrán crear y actualizar. Éstos pueden incluir, nombres, edad, dirección, mensajes de estado y foto del perfil.
- Envío y recepción de mensajes, que será la característica principal de la aplicación y eje fundamental de la misma.
- Creación de usuarios por medio de una solicitud, gestionada por el administrador del programa.
- Comunicación mediante chats privados entre usuarios individuales.
- Comunicación entre grupos de usuarios mediante la creación de chats entre varias usuarios a la vez.
- Tener un registro de las fechas y hora de los mensajes.
- Envíos de archivos en formatos imagen (jpg y png), pdf y doc, siendo esta opción disponible sólo para la realización del chats privados, debido a la limitación de los métodos de android estudio.
- Eliminación de mensajes.

Capítulo 4º.Finalidad

ChatW es una aplicación de mensajería instantánea con la que se pueden comunicar varias personas (usuarios), enviando y recibiendo mensajes de texto o archivos adjuntos. Además de los mensajes de texto, también se pueden enviar archivos en formato doc y pdf, así como imágenes en formato jpg y png. Todo se complementa con la creación de grupos de Chat. Los datos se guardan en una base de datos externa, necesaria para el uso de la aplicación.

Los usuarios se pueden registrar, creando perfiles en los que pueden indicar su nombre, edad, dirección, estado y foto para identificarse.

CHATW.

Alexis Andrés Rodríguez Rodríguez

Capítulo 5º. Medios materiales usados.

- 5º.1. Medios humanos

Para la realización de este proyecto se han utilizado sólo los medios humanos del alumno, como principal programador, Alexis Andrés Rodríguez Rodríguez.

- 5º.2. Hardware.

El hardware utilizado es un ordenador de torre con un procesador AMD Ryzen 7 3700X 8-Core, 16GB de RAM, tarjeta grafica de GeForce GTX 1660 SUPER y con el sistema Windows 10.

- 5º.3. Software.

Parte del software empleado es le lenguaje de programación Java (versión de ME “Micro Edition”), debido a que es un lenguaje muy conocido en implementación de aplicaciones. Además de que la versión de java es la ideal para la realización de un proyecto del entorno Android.

Se ha empleado la plataforma de desarrollo de Android Studio, que es un IDE (Integrated Development Environment), que para este proyecto, ofrece facilidades en el desarrollo de aplicaciones para dispositivos móviles. Siendo una de las ventajas, el usar su simulador, facilitando la realización de pruebas e implementaciones. Este sistema tiene facilidades para el empaquetamiento del programa dentro del IDE y la Firebase, gracias a sus funciones de autentificación, realtime Database y Cloud Storage, que será de importancia para el proyecto, donde nos facilitaran el trabajo, así como la sencillez de conectarlo entre ellos.

CHATW.

Alexis Andrés Rodríguez Rodríguez

*Capítulo 6º. Planificación del proyecto**1. Inicio*

Para empezar tenemos que hablar de la creación del proyecto que iniciamos con el *Empty Activity*, debido a que la clase *main* ya esta creada y tiene algunas importaciones que vamos a necesitar. A la clase *Main* se le realiza los siguientes cambios

```

1 package com.example.chatw;
2 import ...
5   public class MainActivity extends AppCompatActivity {
6     @Override
7     protected void onCreate(Bundle savedInstanceState) {
8       super.onCreate(savedInstanceState);
9       setContentView(R.layout.activity_main);
10      Thread thread = new Thread() {
11        try {
12          sleep( 2000 );
13        } catch (Exception e) {
14          e.printStackTrace();
15        } finally {
16          Intent intent = new Intent( packageContext: MainActivity.this, LoginActivity.class );
17          startActivity(intent);
18        }
19      };
20      thread.start();
21    }
22    protected void onPause(){
23      super.onPause();
24      finish();
25    }
26  }

```

En ella se implementó el *thread* en el que se iniciará en la clase *login* una vez que entre a la aplicación. Para ello e creado el siguiente *Layout*.



En él se puede apreciar los elementos más importantes como los *textfield* y el *button* que solicita el numero del usuario. Cuando se entra con el numero a la cuenta pide un código de verificación que está presente, debido a la propiedad *gone*.

<EditText
 android:id="@+id/codigo"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentStart="true"
 android:layout_marginStart="30dp"

```
    android:layout_marginEnd="30dp"
    android:layout_below="@+id/Imagen"
    android:layout_marginTop="30dp"
    android:hint="Ingresa codigo"
    android:background="@drawable/textos"
    android:padding="10dp"
    android:visibility="gone"
    android:inputType="phone"/>
```

Esta línea de código se aplicó para el código de verificación. Para la realización de la clase de java empezaremos hablando del evento *enviarT* que su función es enviar el numero de teléfono a la base de datos.

```
enviarT.setOnClickListener(new View.OnClickListener(){
    public void onClick(View v){
        phoneNumber = numero.getText().toString();
        if(TextUtils.isEmpty(phoneNumber)){
            Toast.makeText(LoginActivity.this,"Ingrese numero",Toast.LENGTH_SHORT).show();
        }else{
            loadignBar.setTitle("Enviando codigo");
            loadignBar.setMessage("Espera... ");
            loadignBar.show();
            loadignBar.setCancelable(true);
            PhoneAuthOptions
options=PhoneAuthOptions.newBuilder(mAuth).setPhoneNumber(phoneNumber).setTimeout(60L,
TimeUnit.SECONDS).setActivity(LoginActivity.this).setCallbacks(callbacks).build();
            PhoneAuthProvider.verifyPhoneNumber(options);
        }
    }
});
```

Si el numero que se desea ingresar está en la base de datos, se accederá sin ningún tipo de problema. Pero si no está registrado, aparecerá una alerta indicando lo siguiente “*Numero equivocado, prueba otra vez*”. Para la verificación del código realizamos lo siguiente:

Primero ocultamos el botón del número de teléfono y el *Textfield* del teléfono.

```
enviarC.setOnClickListener(new View.OnClickListener(){
    public void onClick(View v){
        numero.setVisibility(View.GONE);
        enviarT.setVisibility(View.GONE);
        String verificacionCode = codigo.getText().toString();
        if(TextUtils.isEmpty(verificacionCode)){
            Toast.makeText(LoginActivity.this,"Ingresa codigo
recibido",Toast.LENGTH_SHORT).show();
        }else{
            loadignBar.setTitle("Entrando");
            loadignBar.setMessage("... ");
            loadignBar.show();
            loadignBar.setCancelable(true);
            PhoneAuthCredential credential=
            PhoneAuthProvider.getCredential(mverificacion,verificacionCode);
            signInPhoneAuhCredential(credential);
        }
    });
});
```

Segundo, cuando entre el usuario, se verificará si el código ha sido correcto y permitirá pasar al usuario a la pantalla principal. Para una correcta verificación se realiza un *callbacks* para la verificación y el método *signInPhoneAuhCredential*.



En el inicio, si el usuario no esta registrado lo pasará al *layout* de *Setup* el cual consta de los siguientes elementos: Un *CircleImageView* (importado) para la foto de perfil, cuatro *textfield* en los cuales se tendrá que llenar dichos campos obligatoriamente y un botón de guardar. En esta clase hay un método importante para guardar la información en la base de datos *Firebase*.

```
private void GuardarDB(){
    String nom = Nombre.getText().toString();
    String ciu = Ciudad.getText().toString();
    String est = Estado.getText().toString();
    String eda = Edad.getText().toString();
    if(TextUtils.isEmpty(nom)){
        Toast.makeText( context: this, text: "Debes ingresar el nombre",Toast.LENGTH_SHORT).show();
    }else if (TextUtils.isEmpty(ciu)){
        Toast.makeText( context: this, text: "Debes ingresar la ciudad",Toast.LENGTH_SHORT).show();
    }else if(TextUtils.isEmpty(est)){
        Toast.makeText( context: this, text: "Debes ingresar un estado",Toast.LENGTH_SHORT).show();
    }else if(TextUtils.isEmpty(eda)){
        Toast.makeText( context: this, text: "Debes ingresar tu edad",Toast.LENGTH_SHORT).show();
    }else{
        dialog.setTitle("Guardando datos");
        dialog.setTitle("Por favor espere a que se termine el proceso");
        dialog.show();
        dialog.setCanceledOnTouchOutside(false);
        FirebaseMessaging.getInstance().getToken().addOnCompleteListener(new OnCompleteListener<String>() {
            @Override
            public void onComplete(@.NonNull Task<String> task) {
                if(task.isSuccessful()){
                    token=task.getResult();
                    HashMap map = new HashMap();
                    map.put("uid",CurrentUserID);
                    map.put("nombre",nom);
                    map.put("ciudad", ciu);
                    map.put("estado",est);
                    map.put("Edad",eda);
                    map.put("token",token);
                    UserRef.child(CurrentUserID).updateChildren(map).addOnCompleteListener(new OnCompleteListener() {
                        @Override
                        public void onComplete(@NonNull Task task) {
                            if(task.isSuccessful()){
                                Toast.makeText( context: SetupActivity.this, text: "Datos guardados",Toast.LENGTH_SHORT).show();
                                dialog.dismiss();
                                EnviarAlInicio();
                            }else{
                                String err= task.getException().getMessage();
                                Toast.makeText( context: SetupActivity.this, text: "Error"+err,Toast.LENGTH_SHORT).show();
                            }
                        }
                    });
                }
            }
        });
    }
}
```

Lo que realiza este método es recoger los datos del *layout* y guardarlos en la base de datos, de forma organizada. Una vez se presione el botón de guardar información, vuelve de forma rápida al inicio. Esta clase sólo aparece cuando el usuario no está registrado en la base de datos. En el inicio sólo hay dos elementos que conecta con toda la aplicación, un *TabAdapter*, en el cual está conectado los chats privados, los grupos, contactos y las solicitudes. En el menú de la esquina superior derecha, están las funciones de buscar contacto, nuevo grupo, mi perfil y cerrar sesión.

2. *grupos*

En el menú del *InícioActivity* se creó un método para poder crear los chats de grupo, que se realizará mediante una alerta emergente para pedir al usuario el nombre del grupo, tal como se muestra a continuación.

```
private void NuevoGrupo() {  
    AlertDialog.Builder builder = new AlertDialog.Builder(InicioActivity.this,R.  
    builder.setTitle("Nombre de grupo:");  
    final EditText nombregrupo = new EditText(InicioActivity.this);  
    nombregrupo.setHint("ejem. Roll or Die");  
    builder.setView(nombregrupo);  
    builder.setPositiveButton("Crear", new DialogInterface.OnClickListener() {  
        public void onClick(DialogInterface dialog, int which) {  
            String nombreG = nombregrupo.getText().toString();  
            if(TextUtils.isEmpty(nombreG)){
```

```

        Toast.makeText(InicioActivity.this, "Ingrese el nombre del
        grupo", Toast.LENGTH_SHORT).show());
    }else {
        CrearGrupoFirebase(nombreG);
    }
}

)).setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
        dialog.cancel();
    }
});
builder.show();
}

```

Para la creación de dichos grupos en la base de datos se utiliza un método dentro del *InicioActivity*, donde se creara el grupo en la base de datos de *Firebase*.

```

private void CrearGrupoFirebase(String nombreG) {
    RootRef.child(nombreG).setValue("").addOnCompleteListener(new
    OnCompleteListener<Void>() {
        public void onComplete(@NonNull Task<Void> task) {
            if(task.isSuccessful()){
                Toast.makeText(InicioActivity.this, "Grupo creando con
                existo...", Toast.LENGTH_SHORT).show();
            }else{
                String error= task.getException().getMessage().toString();
                Toast.makeText(InicioActivity.this, "Error"+error, Toast.LENGTH_SHORT).show();
            }
        }
    });
}

```

Item 1 Sub Item 1
Item 2 Sub Item 2
Item 3 Sub Item 3
Item 4 Sub Item 4
Item 5 Sub Item 5
Item 6 Sub Item 6
Item 7 Sub Item 7
Item 8 Sub Item 8
Item 9 Sub Item 9
Item 10 Sub Item 10

Para la visualización de los chats de grupos ya existentes se utiliza un *ListView* en el *GrupoFagment* para su visualización, con los nombres de los grupos, gracias a la base de datos. Para cargar los nombres de los grupos se realizó un *ArrayList* para llenarla con los nombres de los grupos. Hecho el *ArrayList* se actualizará mediante el método de *MostrarListaG*, en caso de crear un nuevo grupo para actualizarlo al instante. Para entrar en los chats de cada grupo el usuario tendrá sólo que tocar al que se desee entrar.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    grupoFramentoView=
    inflater.inflate(R.layout.fragment_grupos,container,false);
    GrupoRef=
```

```
FirebaseDatabase.getInstance().getReference().child("Grupos");
IniciarLista();
MostraListaG();
list_view.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String CurrenGrupoNombre = parent.getItemAtPosition(position).toString();
        Intent intent = new Intent(getApplicationContext(), GrupoChatActivity.class);
        intent.putExtra("nombregrupo", CurrenGrupoNombre);
        startActivity(intent);
    }
});
return grupoFramentoView;
}
```

Dentro del *layout*, el *GrupoChatActivity* se diseñó con el objetivo de tener una visualización y escritura sencilla para los mensajes enviados y recibidos por los usuarios, sin demasiado consumo de recursos.



Para enviar un mensaje se escribe en el *Textfield* y se pulsa el botón de enviar mensaje, que se guardará en forma de texto y también con un registro de día, mes, año, hora y minutos en el que se escribió, así como el nombre del remitente del mensaje en la base de datos.

Para evitar problemas con los mensajes a la hora de guardarlos en la base de datos, se mapeó con el uso *getters* y *setters*.

```
private void GuardarMensaje() {
    String mensaje= MensajeUsuario.getText().toString();
    String mensajekey = GrupoRef.push().getKey();
    if(TextUtils.isEmpty(mensaje)){
        Toast.makeText( context: this, text: "Por favor introduca un mensaje",Toast.LENGTH_SHORT).show();
    }else{
        Calendar fechaC = Calendar.getInstance();
        SimpleDateFormat currentFecha = new SimpleDateFormat( pattern: "MMM dd, yyyy");
        Fecha=currentFecha.format(fechaC.getTime());
        Calendar horaC = Calendar.getInstance();
        SimpleDateFormat currentHora = new SimpleDateFormat( pattern: "hh:mm a");
        Hora=currentHora.format(horaC.getTime());
        HashMap<String, Object> mensajegrupo = new HashMap<>();
        GrupoRef.updateChildren(mensajegrupo);
        GrupoMensajekeyRef = GrupoRef.child(mensajekey);
        HashMap<String, Object> mensajeinformacion = new HashMap<>();
        mensajeinformacion.put("nombre",CurrentUserName);
        mensajeinformacion.put("mensaje",mensaje);
        mensajeinformacion.put("fecha",Fecha);
        mensajeinformacion.put("hora",Hora);
        GrupoMensajekeyRef.updateChildren(mensajeinformacion);
    }
}
```

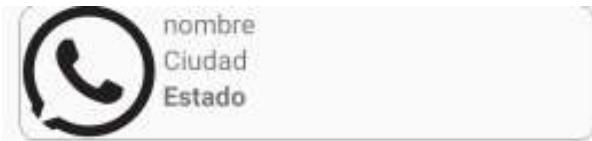
La visualización de dichos mensajes, se muestran tomados de la base de datos, con el uso del método *Mostrarmensaje*.

```
private void Mostrarmensaje(DataSnapshot snapshot) {
    Iterator iterator = snapshot.getChildren().iterator();
    while (iterator.hasNext()){
        String fecha=(String)((DataSnapshot)iterator.next()).getValue();
        String hora=(String)((DataSnapshot)iterator.next()).getValue();
        String mensaje=(String)((DataSnapshot)iterator.next()).getValue();
        String nombre=(String)((DataSnapshot)iterator.next()).getValue();
        verMensaje.append(nombre+"\n"+mensaje+"\n"+fecha+" "+hora+"\n\n");
        scrollView.fullScroll(ScrollView.FOCUS_DOWN);
    }
}
```

3. solicitudes

Antes de explicar como se realizó las solicitudes en los chats, comentaremos la clase *MiPerfilActivity* donde su función es la actualización de los perfiles, siendo lo único que podríamos destacar es el método de *ActualizarPerfil*, el cual actualiza el perfil del usuario. Las demás funciones son las que se utilizaron en el *SetupActivity*.

```
private void ActualizarPerfil() {
    String nom = nombre.getText().toString();
    String ciu = ciudad.getText().toString();
    String est = estado.getText().toString();
    String eda = edad.getText().toString();
    if (TextUtils.isEmpty(nom)) {
        Toast.makeText(context: this, text: "Debes ingresar el nombre", Toast.LENGTH_SHORT).show();
    } else if (TextUtils.isEmpty(ciu)) {
        Toast.makeText(context: this, text: "Debes ingresar la ciudad", Toast.LENGTH_SHORT).show();
    } else if (TextUtils.isEmpty(est)) {
        Toast.makeText(context: this, text: "Debes ingresar un estado", Toast.LENGTH_SHORT).show();
    } else if (TextUtils.isEmpty(eda)) {
        Toast.makeText(context: this, text: "Debes ingresar tu edad", Toast.LENGTH_SHORT).show();
    } else {
        HashMap profile=new HashMap();
        profile.put("uid",CurrentuserID);
        profile.put("nombre",nom);
        profile.put("ciudad",ciu);
        profile.put("estado",est);
        profile.put("Edad",eda);
        RootRef.child("Usuarios").child(CurrentuserID).updateChildren(profile).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    EnviarInicio();
                    Toast.makeText(context: MiPerfilActivity.this, text: "Guardado exitosamente", Toast.LENGTH_SHORT).show();
                } else {
                    String err = task.getException().getMessage().toString();
                    Toast.makeText(context: MiPerfilActivity.this, text: "Error" + err, Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```



Una vez explicado la actualización del perfil, pasamos a comentar la clase *BuscarAmigo*. En el *Layout* de *BuscarAmigoActivity* se usó un *recyclerview* para la visualización de los demás usuarios, donde empleamos el *user_display_layout* en el que mostrará la imagen de perfil de los usuarios, el nombre, la ciudad y en esta parte del programa, su estado. Para poder usar los datos de usuario se mapeó los *Contactos* usando *getters* y *setters*. De esta forma se pueden ver los usuarios con más detalle y enviar la solicitud para el chat. sólo se tiene que pulsar el *layaout* del usuario para entrar en el Perfil.



Aquí en *PerfilActivity* se podrá ver con detalle el perfil del usuario y además se podrá realizar las solicitudes de los chats privados. Para ello,

se utilizó el método de *EnviarRequerimiento* en que para sólo se tendrá que pulsar el botón que “ENVIAR MENSAJE”. Este método tiene la función de enviar la petición por la base de datos que le llegará al otro usuario con el que se quiera contactar.

```

private void EnviarRequerimiento() {
    SolicitudRef.child(usuario_enviarid).addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if(snapshot.hasChild(usuario_revid)){
                String requerimiento=snapshot.child(usuario_revid).child("tipo").getValue().toString();
                if(requerimiento.equals("enviado")){
                    CurrenEstado="enviada";
                    enviarmensaje.setText("Cancelar Solicitud");
                }
                else if(requerimiento.equals("Recibido")){
                    CurrenEstado="recibida";
                    enviarmensaje.setText("Aceptar Solicitud");
                    cancelarmensaje.setVisibility(View.VISIBLE);
                    cancelarmensaje.setEnabled(true);
                    cancelarmensaje.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) { CancelarSolicitudM(); }
                    });
                }
            }else{
                ContactosRef.child(usuario_enviarid).addListenerForSingleValueEvent(new ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot snapshot) {
                        if(snapshot.hasChild(usuario_revid)){
                            CurrenEstado="amigos";
                            enviarmensaje.setText("Eliminar Contacto");
                        }
                    }

                    @Override
                    public void onCancelled(@NonNull DatabaseError error) {

                    };
                });
            }
        }
    });
    if(!usuario_enviarid.equals(usuario_revid)){
        enviarmensaje.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                enviarmensaje.setEnabled(false);
                if(CurrenEstado.equals("nuevo")){
                    EnviarSolicitarM();
                }
                if(CurrenEstado.equals("enviada")){
                    CancelarSolicitudM();
                }
                if(CurrenEstado.equals("recibida")){
                    AceptarSolicitudM();
                }
                if(CurrenEstado.equals("amigos")){
                    EliminarContacto();
                }
            }
        });
    }else{
        enviarmensaje.setVisibility(View.GONE);
    }
}

```

Para facilitar este método, usamos dentro de éste, otros cuatro métodos como el *EnviarSolicitarM*, que entra en acción cuando se envió la solicitud.

```

private void EnviarSolicitarM()
{ SolicitudRef.child(usuario_enviarid).child(usuario_revid).child("tipo").setValue("enviado").addOnCompleteListener(new OnCompleteListener<Void>() {
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful())
{ SolicitudRef.child(usuario_revid).child(usuario_enviarid).child("tipo").setValue("Recibido").addOnCompleteListener(new OnCompleteListener<Void>() {
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful()){
            HashMap<String,String> chatNotification=new HashMap<>();
            chatNotification.put("de",usuario_revid);
            chatNotification.put("tipo","requerimiento");
NotificacionesRef.child(usuario_enviarid).push().setValue(chatNotification).addOnCompleteListener(new OnCompleteListener<Void>() {
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful()){
            enviarmensaje.setEnabled(true);
            CurrenEstado="vieja";
            enviarmensaje.setText("Cancelar Requerimiento");
        }
    }
});
}
});
}
});
}
});
```

El otro método es el *AceptarSolicitudM* que gestionará a los dos usuarios que serán agregados como contactos, una vez aceptada la solicitud de chat, siendo registrados en la base de datos.

```

    }
});

}

```

El método *CancelarSolicitudM* se emplea por si el usuario que solicitó el chat privado, desea cancelar la petición. De esta forma se elimina dicha solicitud en la base de datos, que sólo está disponible, si aún no se ha aceptado la solicitud.

```

private void CancelarSolicitudM()
{ SolicitudRef.child(usuario_enviarid).child(usuario_revid).removeValue().addOnCompleteListener
(new OnCompleteListener<Void>() {
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful())
{ SolicitudRef.child(usuario_revid).child(usuario_enviarid).removeValue().addOnCompleteListener
(new OnCompleteListener<Void>() {
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful()){
            enviarmensaje.setEnabled(true);
            CurrenEstado="nueva";
            enviarmensaje.setText("Enviar Mensaje");
            cancelarmensaje.setVisibility(View.GONE);
            cancelarmensaje.setEnabled(false);
        }
    }
});
}
}
});
}
}

```

Y por último, el método *EliminarContacto* se emplea una vez se haya aceptado la solicitud del chat privado, donde estará disponible el botón de “Eliminar Contacto”. Con lo que se consigue eliminar de la base de datos el contacto de ambos usuarios.

```

private void EliminarContacto()
{ ContactosRef.child(usuario_enviarid).child(usuario_revid).removeValue().addOnCompleteListener

```

```
er(new OnCompleteListener<Void>() {
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful())
            { ContactosRef.child(usuario_revid).child(usuario_enviarid).removeValue().addOnCompleteListener(new OnCompleteListener<Void>() {
                public void onComplete(@NonNull Task<Void> task) {
                    if(task.isSuccessful()){
                        enviarmensaje.setEnabled(true);
                        CurrenEstado="nueva";
                        enviarmensaje.setText("Enviar Mensaje");
                        cancelarmensaje.setVisibility(View.GONE);
                        cancelarmensaje.setEnabled(false);
                    }
                }
            });
        }
    }
});
```

```
private void ObtenerInformacionDB() {
    UserRef.child(usuario_revid).addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if(snapshot.exists()&&snapshot.hasChild(path: "Imagen")){
                String nombreUser=snapshot.child("nombre").getValue().toString();
                String ciudadUser=snapshot.child("ciudad").getValue().toString();
                String estadouser=snapshot.child("estado").getValue().toString();
                String imagenUser=snapshot.child("Imagen").getValue().toString();
                String eddaduser=snapshot.child("Edad").getValue().toString();
                Picasso.get().load(imagenUser).placeholder(R.drawable.error).into(usuarioim);
                usuarioNombre.setText(nombreUser);
                usuarioCiudad.setText(ciudadUser);
                usuarioEstado.setText(estadouser);
                usuarioEdad.setText(eddaduser);
                EnviarRequerimiento();
            }else{
                String nombreUser=snapshot.child("nombre").getValue().toString();
                String ciudaduser=snapshot.child("ciudad").getValue().toString();
                String estadouser=snapshot.child("estado").getValue().toString();
                String eddaduser=snapshot.child("Edad").getValue().toString();
                usuarioNombre.setText(nombreUser);
                usuarioCiudad.setText(ciudaduser);
                usuarioEstado.setText(estadouser);
                usuarioEdad.setText(eddaduser);
                EnviarRequerimiento();
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
        }
    });
}
```

Para terminar con esta clase les mostraremos el método de *ObtenerInformacionDB* que obtendrá la información del usuario en el *layout*, llenando los *TextView* con dichos datos

Antes de acabar este apartado, mencionamos *fragment* *SolicitudesFrgmet* para aceptar o rechazar las solicitudes desde el *layout* de *InicioActivity*, teniendo más sencillez de uso de la aplicación.

4. chat privado

En este último apartado de la clase *InícioActivity*, se añadió un método para saber si el usuario está conectado o la última conexión que tuvo con el nombre *ActualizarActividad*.

```
protected void ActualizarActividad(String estado){
    String CurrentTime, CurrentDate;
    Calendar calendar= Calendar.getInstance();
    SimpleDateFormat dateFormat=new SimpleDateFormat("MMM dd,yyyy");
    CurrentDate=dateFormat.format(calendar.getTime());
    SimpleDateFormat dateFormatI= new SimpleDateFormat("hh:mm a");
    CurrentTime=dateFormatI.format(calendar.getTime());
    HashMap<String, Object>EstadoOnline=new HashMap<>();
    EstadoOnline.put("hora",CurrentTime);
    EstadoOnline.put("fecha",CurrentDate);
    EstadoOnline.put("estadoAct",estado);
    UserRef.child(CurrentUserId).child("estadoUser").updateChildren(EstadoOnline);
}
```

Cuando se ejecuta, sólo se tendrá que estar en *InícioActivity* y se actualizará el estado a conectado gracias al método *onStart*.

```
protected void onStart(){
    super.onStart();
    FirebaseAuth curUser= mAuth.getCurrentUser();
    if(curUser==null){
        EnviarAlogin();
    }else{
        VerificarUsuario();
        ActualizarActividad("activo");
    }
}
```

Para el registro de la última actividad sólo se tendrá que salir de la app o cerrar sección .

```
protected void onStop() {
    super.onStop();
    FirebaseAuth curUser= mAuth.getCurrentUser();
```

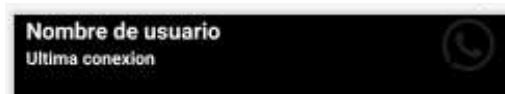
```

if(curUser !=null){
    ActualizarActividad("inactivo");
}
}

protected void onDestroy() {
    super.onDestroy();
    FirebaseAuth curUser= mAuth.getCurrentUser();
    if(curUser !=null){
        ActualizarActividad("inactivo");
    }
}
}

```

En la visualización de los contactos se decidió introducir la opción de reciclar *RecyclerView* y el *user_display_layout* para la visualización de los chats, con un cambio en el *user_display_layout* que permite ver la conexión o ultima actividad.

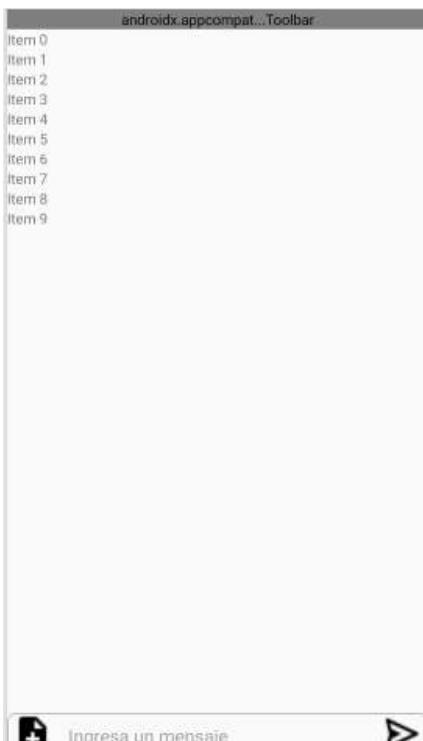


En el *layout_chat_bar* aparecerá los datos de usuario como el nombre, la foto de perfil y su conexión extraída de la base de datos.

En el *Layout usuario_mensajes* se puso dos *TextView* y *ImageView* (debido a que no hay una imagen, no se puede

visualizar) en la visualización de los mensajes.

Terminamos con los *layout* donde se integraron los elementos que se mencionaron anteriormente en *layout_chat_bar*, donde en el *recyclerview* aparecerán los mensajes de los usuarios mediante el *layout* de *usuario_mensajes*. Aparte de este punto también se implementó el *editText* y el botón de enviar mensaje del chat de grupo, para acabar implementando un botón adicional para poder pasar documentos y imágenes.



En este punto se hablará del método de *EnviarMensaje* del *ChatActivity*, siendo en este caso para enviar mensajes de textos, donde utilizando el mapeado de los mensajes se enviarán los mensajes a la base de datos, guardándose para los dos usuarios.

```
private void EnviarMensaje() {
    String mensajeTexto=mensaje.getText().toString();
    if(TextUtils.isEmpty(mensajeTexto)){
        Toast.makeText( context: this, text: "Por favor escriba su mensaje", Toast.LENGTH_SHORT).show();
    }else{
        String mensajeEnviadoRef="Mensajes/"+EnviarUserID+"/"+RecibirUserID;
        String mensajeRecibidoRef="Mensajes/"+RecibirUserID+"/"+EnviarUserID;
        DatabaseReference usuarioMensajeRef=RootRef.child("Mensajes").child(EnviarUserID).child(RecibirUserID).push();
        String mensajePushID=usuarioMensajeRef.getKey();
        Map mensajeTxt= new HashMap();
        mensajeTxt.put("mensaje",mensajeTexto);
        mensajeTxt.put("tipo","texto");
        mensajeTxt.put("de",EnviarUserID);
        mensajeTxt.put("para",RecibirUserID);
        mensajeTxt.put("mensajeID",mensajePushID);
        mensajeTxt.put("fecha",CurrentDate);
        mensajeTxt.put("hora",CurrentTime);
        Map mensajeTxtfull= new HashMap();
        mensajeTxtfull.put(mensajeEnviadoRef+"/"+mensajePushID,mensajeTxt);
        mensajeTxtfull.put(mensajeRecibidoRef+"/"+mensajePushID,mensajeTxt);
        RootRef.updateChildren(mensajeTxtfull).addOnCompleteListener(new OnCompleteListener() {
            @Override
            public void onComplete(@NonNull Task task) {
                if(task.isSuccessful()){
                    Toast.makeText( context: ChatActivity.this, text: "Mensaje Enviado...",Toast.LENGTH_SHORT).show();
                }else{
                    Toast.makeText( context: ChatActivity.this, text: "Error al enviar",Toast.LENGTH_SHORT).show();
                }
                mensaje.setText("");
            }
        });
    }
}
```

La visualización de los mensajes de textos se realiza mediante una clase *MensajeAdapter* que permite darle funcionalidad al *layout* de *usuario_mensajes* cargando los mensajes de la siguiente forma:

```
if(TipoMensaje.equals("texto")){
    if(DeUsuarioId.equals(mensajeEnviadoID)){
        holder.enviarMensajeTexto.setVisibility(View.VISIBLE);
        holder.enviarMensajeTexto.setBackgroundResource(R.drawable.enviar_mensaje_layout);
        holder.enviarMensajeTexto.setTextColor(Color.WHITE);
        holder.enviarMensajeTexto.setText(mensajes.getMensaje()+"\n\n"+mensajes.getFecha()
+ "-" +mensajes.getHora());
    }else{
        holder.recibirImagenPerfil.setVisibility(View.GONE);
    }
}
```

```

holder.recibirMensajeTexto.setVisibility(View.VISIBLE);
holder.recibirMensajeTexto.setBackgroundResource(R.drawable.recibir_mensaje_layout);
holder.recibirMensajeTexto.setTextColor(Color.BLACK);
holder.recibirMensajeTexto.setText(mensajes.getMensaje()+"\n\n"+mensajes.getFecha())
+"-"+mensajes.getHora());
}
}

```

Las opciones de escoger que archivos se desean compartir, se realiza mediante un menú con las opciones de imagen, PDF y WORD.

```

botonarchivo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        CharSequence opciones[] = new CharSequence[]{
            "Imagenes",
            "PDF",
            "Word"
        };
        AlertDialog.Builder builder = new AlertDialog.Builder(context: ChatActivity.this);
        builder.setTitle("Selecciona el archivo");
        builder.setItems(opciones, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int i) {
                if(i==0){
                    check="image";
                    Intent intent = new Intent();
                    intent.setAction(Intent.ACTION_GET_CONTENT);
                    intent.setType("image/*");
                    startActivityForResult(intent.createChooser(intent, title: "Seleccionar imagen"), requestCode: 438);
                }
                if(i==1){
                    check="pdf";
                    Intent intent = new Intent();
                    intent.setAction(Intent.ACTION_GET_CONTENT);
                    intent.setType("application/pdf");
                    startActivityForResult(intent.createChooser(intent, title: "Seleccionar un archivo PDF"), requestCode: 438);
                }
                if(i==2){
                    check="docx";
                    Intent intent = new Intent();
                    intent.setAction(Intent.ACTION_GET_CONTENT);
                    intent.setType("application/docx");
                    startActivityForResult(intent.createChooser(intent, title: "Seleccionar un archivo DOCX"), requestCode: 438);
                }
            }
        });
        builder.show();
    }
});

```

El envío de la imagen o documento tendrá la estructura de los mensajes de texto, pero con el cambio del tipo de archivo que se seleccionó como tipo de mensaje.

```
public void onComplete(@NonNull Task<Uri> task) {
    if(task.isSuccessful()){
        Uri downloadUri=task.getResult();
        myUrl=downloadUri.toString();
        Map mensajeTxt= new HashMap();
        mensajeTxt.put("mensaje",myUrl);
        mensajeTxt.put("tipo",check);
        mensajeTxt.put("de",EnviarUserID);
        mensajeTxt.put("para",RecibirUserID);
        mensajeTxt.put("mensajeID",mensajePushID);
        mensajeTxt.put("fecha",CurrentDate);
        mensajeTxt.put("hora",CurrentTime);
        Map mensajeTxtfull= new HashMap();
        mensajeTxtfull.put(mensajeEnviadoRef+"/"+mensajePushID,mensajeTxt);
        mensajeTxtfull.put(mensajeRecibidoRef+"/"+mensajePushID,mensajeTxt);
        RootRef.updateChildren(mensajeTxtfull).addOnCompleteListener(new OnCompleteListener() {
            @Override
            public void onComplete(@NonNull Task task) {
                if(task.isSuccessful()){
                    dialog.dismiss();
                    Toast.makeText(context: ChatActivity.this, text: "Mensaje Enviado...",Toast.LENGTH_SHORT).show();
                }else{
                    dialog.dismiss();
                    Toast.makeText(context: ChatActivity.this, text: "Error al enviar",Toast.LENGTH_SHORT).show();
                }
                mensaje.setText("");
            }
        });
    }
}
```

Pero las imágenes tendrá el siguiente agregado de forma que se gestionen de forma correcta.

```
StorageReference storageReference=
FirebaseStorage.getInstance().getReference().child("Archivo");
String mensajeEnviadoRef="Mensajes/"+EnviarUserID+"/"+RecibirUserID;
String mensajeRecibidoRef="Mensajes/"+RecibirUserID+"/"+EnviarUserID;
DatabaseReference
usuarioMensajeRef=RootRef.child("Mensajes").child(EnviarUserID).child(RecibirUserID).push();
String mensajePushID=usuarioMensajeRef.getKey();
final StorageReference filePath= storageReference.child(mensajePushID+"."+ "jpg");
uploadTask=filePath.putFile(fileUri);
uploadTask.continueWithTask(new Continuation() {
    public Object then(@NonNull Task task) throws Exception {
        if(!task.isSuccessful()){

```

```

        throw task.getException();
    }
    return filePath.getDownloadUrl();
}
})

```

En la visualización de imágenes se usará esta parte de la clase *MensajeAdapter*.

```

else if(TipoMensaje.equals("imagen")){
    if(DeUsuarioId.equals(mensajeEnviadoID)){
        holder.mensajeImagenEnviar.setVisibility(View.VISIBLE);
        Picasso.get().load(mensajes.getMensaje()).into(holder.mensajeImagenEnviar);
    }else{
        holder.recibirImagenPerfil.setVisibility(View.VISIBLE);
        holder.mensajeImagenRecibir.setVisibility(View.VISIBLE);
        Picasso.get().load(mensajes.getMensaje()).into(holder.mensajeImagenRecibir);
    }
}

```

Y para los PDF y WORD se usará este fragmento de la misma clase.

```

else if(TipoMensaje.equals("pdf")||TipoMensaje.equals("docx")){
    if(DeUsuarioId.equals(mensajeEnviadoID)){
        holder.mensajeImagenEnviar.setVisibility(View.VISIBLE);
        Picasso.get().load("https://firebasestorage.googleapis.com/v0/b/chat-w-2fcc1.appspot.com/
o/Archivo%2FArchivos.png?alt=media&token=e2301f99-e795-4e2f-9ddc-
6df5e8e90edf").into(holder.mensajeImagenEnviar);
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent intent =new Intent(Intent.ACTION_VIEW,
                Uri.parse(usuarioMensajes.get(i).getMensaje()));
                holder.itemView.getContext().startActivity(intent);
            }
        });
    }else {
        holder.recibirImagenPerfil.setVisibility(View.VISIBLE);
    }
}

```

```

holder.mensajeImagenRecibir.setVisibility(View.VISIBLE);
Picasso.get().load("https://firebasestorage.googleapis.com/v0/b/chat-w-2fcc1.appspot.com/
o/Archivo%2FArchivos.png?alt=media&token=e2301f99-e795-4e2f-9ddc-
6df5e8e90edf").into(holder.mensajeImagenRecibir);
}
}

```

Aquí comentamos las ultimas funciones que se incorporaron al programa siendo la eliminación de mensajes, descarga de archivos y ver imágenes. El método *EliminarMensajesEnviados* se usa en la la eliminación de los mensajes enviados, pero sólo para el usuario que los recibió vera el mensaje perfectamente.

```

private void EliminarMensajesEnviados(final int position, final MensajesViewHolder holder){
    DatabaseReference rootRef= FirebaseDatabase.getInstance().getReference();
    rootRef.child("Mensajes").child(usuarioMensajes.get(position).getDe())
        .child(usuarioMensajes.get(position).getPara())
        .child(usuarioMensajes.get(position).getMensajeID())
        .removeValue().addOnCompleteListener(new OnCompleteListener<Void>() {
            public void onComplete(@NonNull Task<Void> task) {
                if(task.isSuccessful()){
                    Toast.makeText(holder.itemView.getContext(),"Mensaje
                    Eliminado",Toast.LENGTH_SHORT).show();
                }else{
                    Toast.makeText(holder.itemView.getContext(),"Error al
                    eliminar",Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}

```

Con el método *EliminarMensajesRecibidos* se eliminan los mensajes que se han recibido, pero sólo para el usuario que lo recibió los mensajes.

```

private void EliminarMensajesRecibido(final int position, final MensajesViewHolder holder){
    DatabaseReference rootRef= FirebaseDatabase.getInstance().getReference();
    rootRef.child("Mensajes").child(usuarioMensajes.get(position).getPara())

```

```
.child(usuarioMensajes.get(position).getDe())
.child(usuarioMensajes.get(position).getMensajeID())
.removeValue().addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if(task.isSuccessful()){
Toast.makeText(holder.itemView.getContext(),"Mensaje
Eliminado",Toast.LENGTH_SHORT).show();
} else{
Toast.makeText(holder.itemView.getContext(),"Error al
eliminar",Toast.LENGTH_SHORT).show();
}
}
});
```

A diferencia de los anteriores métodos el *EliminarMensajesTodos* sólo lo puede utilizar el usuario que envió el mensaje, aparte de eliminarlos para él, también se eliminan para el usuario que los recibió.

```
private void EliminarMensajesTodos(final int position, final MensajesViewHolder holder) {  
    DatabaseReference rootRef= FirebaseDatabase.getInstance().getReference();  
    rootRef.child("Mensajes").child(usuarioMensajes.get(position).getPara())  
        .child(usuarioMensajes.get(position).getDe())  
        .child(usuarioMensajes.get(position).getMensajeID())  
        .removeValue().addOnCompleteListener(new OnCompleteListener<Void>() {  
    public void onComplete(@NonNull Task<Void> task) {  
        if(task.isSuccessful()){  
            rootRef.child("Mensajes").child(usuarioMensajes.get(position).getDe())  
                .child(usuarioMensajes.get(position).getPara())  
                .child(usuarioMensajes.get(position).getMensajeID())  
                .removeValue().addOnCompleteListener(new OnCompleteListener<Void>() {  
        public void onComplete(@NonNull Task<Void> task) {  
            if(task.isSuccessful()){  
                holder.mensajeEliminado(true);  
            } else {  
                holder.mensajeEliminado(false);  
            }  
        }  
    }  
}
```

```
        Toast.makeText(holder.itemView.getContext(),"Mensaje  
Eliminado",Toast.LENGTH_SHORT).show();  
    }  
}  
});  
}  
}  
}  
}  
});  
}  
}
```

La opción de ver imagen se pasó a la clase *ImagenActivity* donde se visualiza la imagen con más calidad. Este método aparece si es de tipo “imagen”.

```
        builder.show();
    } else if (usuarioMensajes.get(i).getTipo().equals("Imagen")) {
        CharSequence opciones[] = new CharSequence[]{
            "Eliminar para mi",
            "Ver Imagen",
            "Cancelar"
        };
        AlertDialog.Builder builder = new AlertDialog.Builder(holder.itemView.getContext());
        builder.setItems(opciones, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int position) {
                if (position == 0) {
                    EliminarMensajesRecibido(i,holder);
                    Intent intent= new Intent(holder.itemView.getContext(),InicioActivity.class);
                    holder.itemView.getContext().startActivity(intent);
                } else if (position == 1) {
                    Intent intent= new Intent(holder.itemView.getContext(),ImagenActivity.class);
                    intent.putExtra( name: "url",usuarioMensajes.get(i).getMensaje());
                    holder.itemView.getContext().startActivity(intent);
                }
            }
        });
        builder.show();
    }
}
```

En las descarga de documentos lo que se realizará es salir por un momento de la app y lo enviá a un link donde se descarga y se visualiza.

```
if (usuarioMensajes.get(i).getTipo().equals("pdf") ||  
usuarioMensajes.get(i).getTipo().equals("docx")) {  
    CharSequence opciones[] = new CharSequence[] {  
        "Eliminar para mí".
```

```
"Descargar y Ver",
"Cancelar"
};

AlertDialog.Builder builder = new AlertDialog.Builder(holder.itemView.getContext());
builder.setItems(opciones, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int position) {
        if (position == 0) {
            EliminarMensajesRecibido(i,holder);
            Intent intent= new Intent(holder.itemView.getContext(),InicioActivity.class);
            holder.itemView.getContext().startActivity(intent);
        } else if (position == 1) {
            Intent intent = new Intent(Intent.ACTION_VIEW,
            Uri.parse(usuarioMensajes.get(i).getMensaje()));
            holder.itemView.getContext().startActivity(intent);
        }
    }
});

builder.show();
}
```


Capítulo 7º. Fase de pruebas

Mientras se realizaba el proyecto, se fueron haciendo las pruebas por el siguiente orden. Lo primero fue probar las funciones del inicio (registro de usuario, creación de usuario, implementación de los fragmentos de chat de privado, chat de grupos, solicitudes, actualizar perfil, cerrar sección y buscar contactos). Luego se probaron los chats de grupos (creación de grupos, envío de mensajes y lecturas), a continuación comprobaron las solicitudes de chats (funciones para actualizar perfil, ver los perfiles de los demás usuarios y las solicitudes) y por último se revisaron los chats privados (enviar mensajes de texto, imágenes, documentos, visualización de los mensajes y eliminar mensajes).

En todas las fases de pruebas se fueron corriendo los errores que fueron surgiendo, hasta conseguir su funcionamiento correcto.

Capítulo 8º. Conclusiones y trabajos futuros o posibles mejoras

Como conclusión, en este proyecto se implementaron, de una forma u otra, los conocimientos aplicados en el curso, como el diseño de las interfaces; uso, almacenamiento y administración de una base de datos, así como el manejo de archivos y implementación de funciones.

Las mejoras que se podrían implementar al proyecto podrían ser el copiado y pegado de mensajes, mejora de los grupos de chats, de forma que se puedan añadir imágenes y documentos, eliminar mensajes, además de agregar estados, añadir videollamadas o llamadas, etc.

Comentar que este proyecto ha ayudado al desarrollo de una app en el entorno de *android*, aunque los recursos están limitados, como la capacidad de almacenamiento de la memoria interna del móvil tanto del programa y datos, además de la compatibilidad del móvil por la limitación del procesador. En caso de continuar realizando trabajos en este entorno, habría que empezar con una base de datos que no esté tan limitada y sea gratuita.

*Capítulo 9º. Apendices**Referencia de códigos*

Clase <i>main</i>	Pág. 13
Fragmento del <i>layout</i> de la clase <i>LoginActivity</i>	Pág. 13
Evento del <i>button EnviarT</i>	Pág. 14
Evento del <i>button EnviarC</i>	Pág. 15
Método <i>GuardarDB</i> de la clase <i>SetupActivity</i>	Pág. 16
Método <i>NuevoGrupo</i> de la clase <i>InicioActivity</i>	Pág. 16
Método <i>CrearGrupoFirebase</i> de la clase <i>InicioActivity</i>	Pág. 17
Evento del <i>List_view</i> del fragmento <i>GruposFragment</i>	Pág. 18
Método <i>GuardarMensaje</i> de la clase <i>GrupoChatActivity</i>	Pág. 19
Método <i>MostrarMensaje</i> de la clase <i>GrupoChatActivity</i>	Pág. 19
Método <i>ActualizarPerfil</i> de la clase <i>MiPerfilActivity</i>	Pág. 20
Método <i>EnviarRequerimiento</i> de la clase <i>PerfilActivity</i>	Pág. 21
Método <i>EnviarSolicitarM</i> de la clase <i>PerfilActivity</i>	Pág. 22
Método <i>AceptarSolicitudM</i> de la clase <i>PerfilActivity</i>	Pág. 23
Método <i>CancelarSolicitudM</i> de la clase <i>PerfilActivity</i>	Pág. 24
Método <i>EliminarContacto</i> de la clase <i>PerfilActivity</i>	Pág. 24
Método <i>ObtenerInformacionDB</i> de la clase <i>PerfilActivity</i>	Pág. 25
Método <i>ActualizarActividad</i> de la clase <i>InicioActivity</i>	Pág. 26
Método <i>onStart</i> de la clase <i>InicioActivity</i>	Pág. 26
Método <i>onStop</i> de la clase <i>InicioActivity</i>	Pág. 26
Método <i>onDestroy</i> de la clase <i>InicioActivity</i>	Pág. 27
Método <i>EnviarMensaje</i> de la clase <i>ChatActivity</i>	Pág. 28
Fragmento del método <i>onBindViewHolder</i> de la clase <i>MensajeAdapter</i>	Pág. 29
Evento del <i>button botonarchivo</i>	Pág. 29
Fragmento del método <i>onActivityResult</i> de la clase <i>ChatActivity</i>	Pág. 30
Fragmento del método <i>onActivityResult</i> de la clase <i>ChatActivity</i>	Pág. 31
Fragmento del método <i>onBindViewHolder</i> de la clase <i>MensajeAdapter</i>	Pág. 31
Método de <i>EliminarMensajeEnviados</i> de la clase <i>MensajeAdapter</i>	Pág. 32

Método de *EliminarMensajeRecibidos* de la clase Pág. 32

MensajeAdapter

Método de *EliminarMensajeTodos* de la clase Pág. 33

MensajeAdapter

Fragmento del método *onBindViewHolder* de la Pág. 34

clase *MensajeAdapter*

Fragmento del método *onBindViewHolder* de la Pág. 34

clase *MensajeAdapter*

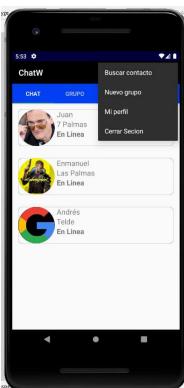
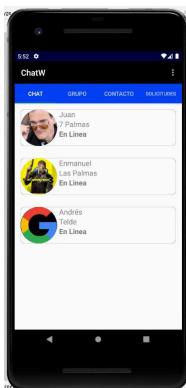
Manual de usuarios



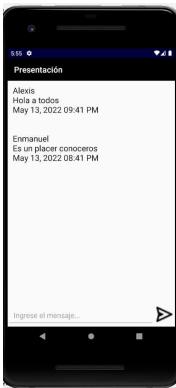
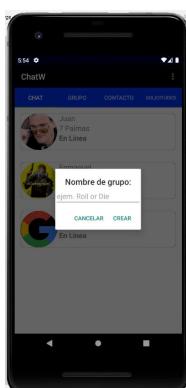
Para entrar entrar a la app hay que ingresar el numero de teléfono y el código de verificación que dio el administrador.



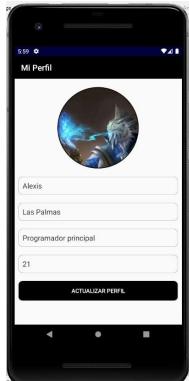
Si es la primera vez que se registra actividad de los usuarios se procederá a crear su cuenta en el que se completará los campos, poniendo una imagen de perfil para utilizar todas las funciones de la app.



Dentro de la app se puede apreciar todas las funciones de una forma ordenada y sencilla.



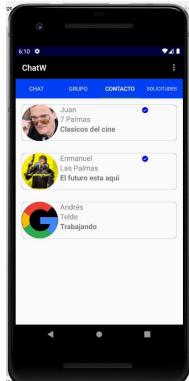
Para ver los grupos ya existentes se tiene que de deslizar a la izquierda o derecha, en caso de estar en otro apartado. Cuando se pulsa crear nuevo grupo aparecerá un mensaje pidiendo al usuario el nombre del grupo, una vez creado el grupo, se puede enviar y recibir los mensajes.



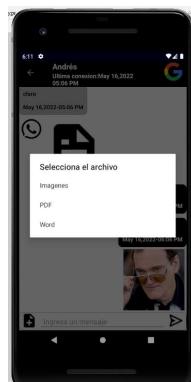
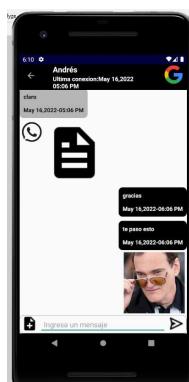
En el apartado de “Mi Perfil” se podrá realizar actualizaciones sobre el perfil del usuario. Una vez hecho los cambios se debe pulsar el botón que situado debajo para realizar los cambios.



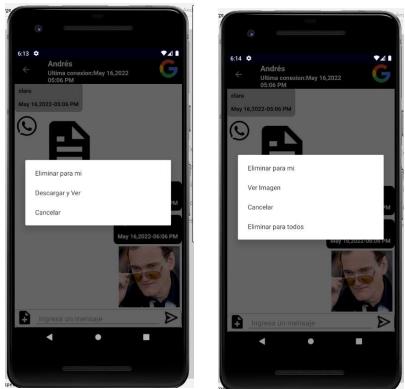
En “Buscar amigos” se puede realizar las solicitudes de enviar mensajes en las que optando por aceptar o rechazar.



En el apartado de contactos los usuarios que aceptaron la solicitud aparecerán si están conectados o su última vez que estuvo Online.

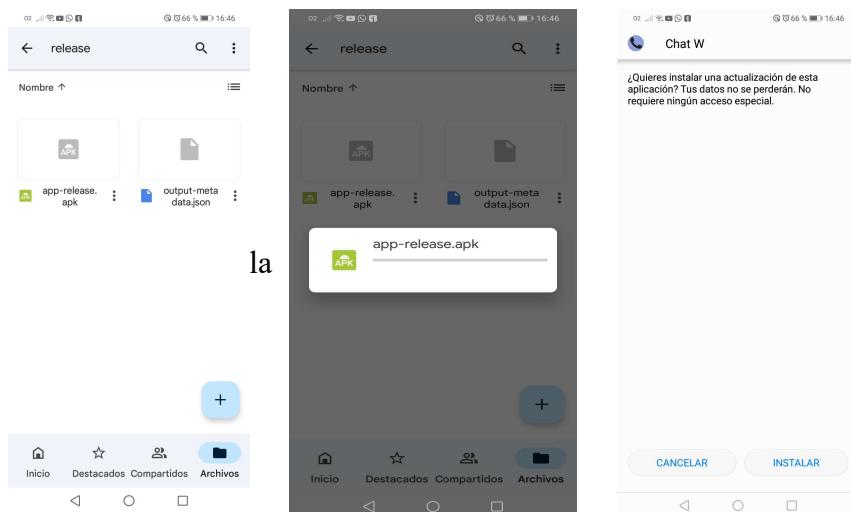


Dentro de los chats privados se puede ver los mensajes de textos, y además, los archivos PDF, WORD e imágenes.



Cuando se pulsa sobre un mensaje aparecerán las opciones de eliminar mensajes (por si hubo alguna equivocación), ver imagen (para ver las imágenes con más detalles) y descargar y ver (para los archivos PDF y Word).

Manual de instalación



Para la instalación es necesario tener el apk de la aplicación y descargarlo en la memoria del dispositivo. Y una vez finalizada la descarga, se podrá instalar app.