

**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

**Carrera de Ingeniería en Sistemas e Informática**

**PROGRAMACIÓN MÓVIL**

Martínez Carlos

Monteros María Paula

NRC: 6112

FECHA: 06 - agosto – 2020

SEMESTRE: May 2020 - Sep 2020

**CALCULADORA HACIENDO USO DEL MODELO MVP**

**Contenido**

[**OBJETIVO 3**](#_Toc47388824)

[**MARCO TEÓRICO 3**](#_Toc47388825)

[**Firebase 3**](#_Toc47388826)

[**Patrón M-V-P (Modelo-Vista-Presentador) 4**](#_Toc47388827)

[**Diferencia entre MVC y MVP 5**](#_Toc47388828)

[**Capas del patrón MVP 5**](#_Toc47388829)

[**MVP en Android 5**](#_Toc47388830)

[**DESARROLLO 6**](#_Toc47388831)

[**CONCLUSIONES 36**](#_Toc47388832)

[**ANEXO N°1: Modelo 37**](#_Toc47388833)

[**ANEXO N°2: Requisitos 38**](#_Toc47388834)

[**ANEXO N°3: Unit Test 40**](#_Toc47388835)

[**ANEXO N°4: JavaDoc 42**](#_Toc47388836)

[**ANEXO N°5: Código 43**](#_Toc47388837)

OBJETIVO

* Desarrollar un chat para dispositivos móviles mediante la herramienta de Android Studio y el lenguaje de programación java y su integración con Firebase para entender y aplicar la arquitectura de desarrollo MVP (Modelo Vista Presentador) y persistencia de datos en programación móvil.

MARCO TEÓRICO

## **Firebase**

Firebase es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles desarrollada por Google en 2014. Es una plataforma ubicada en la nube, integrada con Google Cloud Platform, que usa un conjunto de herramientas para la creación y sincronización de proyectos que serán dotados de alta calidad, haciendo posible el crecimiento del número de usuarios y dando resultado también a la obtención de una mayor monetización.

Ofrece ventajas como:

* Sincronizar fácilmente los datos de sus proyectos sin tener que administrar conexiones o escribir lógica de sincronización compleja.
* Usa un conjunto de herramientas multiplataforma: se integra fácilmente para plataformas web como en aplicaciones móviles. Es compatible con grandes plataformas, como IOS, Android, aplicaciones web, Unity y C++.
* Usa la infraestructura de Google y escala automáticamente para cualquier tipo de aplicación, desde las más pequeñas hasta las más potentes.
* Crea proyectos sin necesidad de un servidor: Las herramientas se incluyen en los SDK para los dispositivos móviles y web, por lo que no es necesario la creación de un servidor para el proyecto.

Firebase dota a sus usuarios de una gran documentación para crear aplicaciones usando esta plataforma. Aparte de esto, ofrece soporte gratuito mediante correo electrónico para todos sus usuarios, y además sus desarrolladores participan activamente en plataformas como Github y StackOverflow.

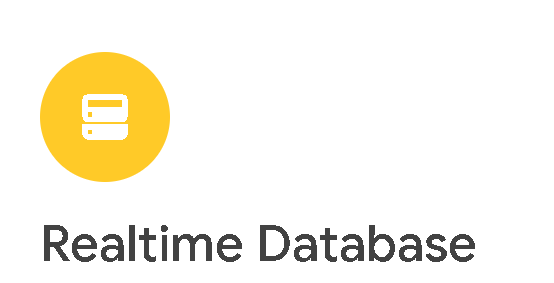


Entre sus productos más usados se encuentra:

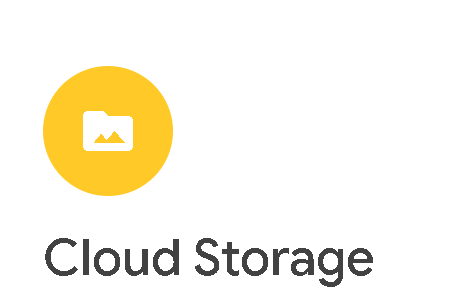
* **Authentication**: Administra tus usuarios de manera simple y segura. Firebase Auth ofrece varios métodos de autenticación (por ejemplo, mediante correo electrónico y contraseña) proveedores externos como Google o Facebook, o tu sistema de cuentas existente de manera directa. Crea tu propia interfaz o aprovecha nuestra IU de código abierto y totalmente personalizable.



* **RealTime Database:** Realtime Database es la base de datos original de Firebase. Esta eficiente solución de baja latencia para aplicaciones móviles exige que el estado de los clientes permanezca sincronizado en tiempo real. Recomendamos Cloud Firestore en lugar de Realtime Database para casi todos los desarrolladores que comienzan un proyecto nuevo.

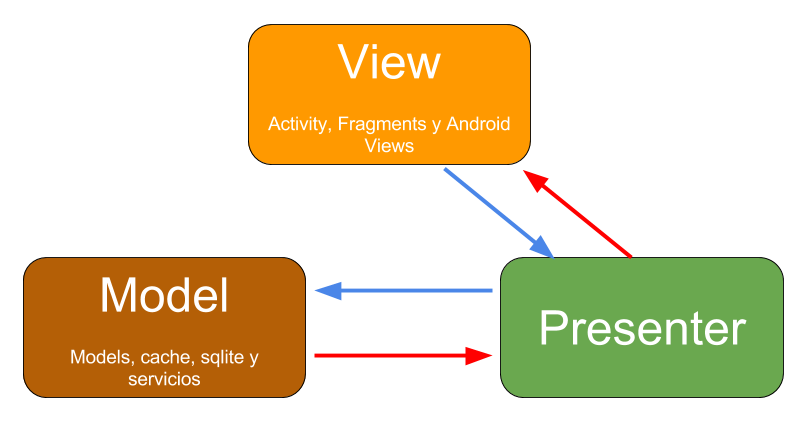


* **Cloud Storage:** Almacena y comparte contenido generado por usuarios, como imágenes, audio y video, con el potente, simple y rentable almacenamiento de objetos creado para la escala de Google. Los SDK de Firebase para Cloud Storage agregan la seguridad de Google a las operaciones de carga y descarga de archivos para tus apps de Firebase, sin importar la calidad de la red



## **Patrón M-V-P (Modelo-Vista-Presentador)**

Modelo–Vista–Presentador (MVP) es una derivación del patrón arquitectónico modelo–vista–controlador (MVC), y es utilizado mayoritariamente para construir interfaces de usuario. En MVP el presentador asume la funcionalidad del "medio-hombre". En MVP, toda lógica de presentación es colocada al presentador.



### **Diferencia entre MVC y MVP**

Ambos modelos parecen bastante similares, pero ofrecen importantes diferencias:

* En el MVC, el modelo notifica a la vista cualquier cambio que sufra el estado del modelo. La información puede pasarse en la propia notificación, o después de la notificación, la vista puede consultar el modelo directamente para obtener los datos actualizados. Por el contrario, en el MVP, la vista no sabe nada sobre el modelo y la función del presentador es la de mediar entre ambos, enlazando los datos con la vista.
* En el modelo MVC, la vista tiende a tener más lógica porque es responsable de manejar las notificaciones del modelo y de procesar los datos. En el modelo MVP, esa lógica se encuentra en el presentador, haciendo a la vista "estúpida".  Su única función es representar la información que el presentador le ha proporcionado.
* En MVC, el modelo tiene lógica extra para interactuar con la vista. En el MVP, esta lógica se encontraría en el presentador.

### **Capas del patrón MVP**

* **Modelo**: Esta capa gestiona los datos. Son las clases que denominaríamos de lógica de negocio.
* **Vista**: Se encarga de mostrar los datos. Aquí se encontrarían nuestros Fragmentos y Vistas.
* **Presentador**: Se sitúa entre el modelo y la vista, permitiendo conectar la interfaz gráfica con los datos.

### **MVP en Android**

Android no nos ofrece de forma nativa la posibilidad de desarrollar nuestras aplicaciones bajo el patrón MVP, de hecho, viola mucho de sus principios básicos. Aun así, podemos llevar a cabo alguna aproximación para este fin. Vamos a ver un posible ejemplo de implementación en el que el usuario dispone de un formulario dónde puede introducir contactos.

* El usuario introduce un contacto y pulsa el botón "añadir contacto".
* El Presentador crea el objeto Contacto con los datos introducidos por el usuario y se lo pasa al Modelo para que lo introduzca en la base de datos.
* El Modelo inserta el contacto en la base de datos.
* Si todo ha ido bien, el Presentador limpia el formulario y refresca la lista de contactos para que aparezca el que acaba de añadir. En caso de que se haya producido algún error, muestra una alerta con un mensaje de error.

La comunicación entre las capas se va a llevar a cabo mediante el uso de interfaces. Para este ejemplo necesitaremos cuatro interfaces.

* ***ProvidedPresenterOps***: Operaciones a las cuales tiene acceso la Vista y son implementadas por el Presentador. Permiten que la Vista se comunique con el Presentador.
* ***ProvidedModelOps***: Operaciones a las cuales tiene acceso el Presentador y son implementadas por el Modelo. Permiten que el Presentador se comunique con el Modelo.
* ***RequiredViewOps***: Operaciones a las cuales tiene acceso el Presentador y son implementadas por la Vista. Permiten que el Presentador se comunique con la Vista.
* ***RequiredPresenterOps***: Operaciones a las cuales tiene acceso el Modelo y son implementadas por el Presentador. Permiten que el Modelo se comunique con el Presentador.

Una vez tenemos definidas las capas de nuestro MVP, necesitamos instanciarlas e insertar las referencias necesarias, además de un mecanismo responsable de mantener el estado del Presentador y del Modelo durante el ciclo de vida de nuestro Fragmento o Actividad. [1]

DESARROLLO

|  |
| --- |
| **AdapterMensaje.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc;  import android.content.Context; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup;  import androidx.annotation.NonNull; import androidx.recyclerview.widget.RecyclerView;  import com.bumptech.glide.Glide;  import java.util.ArrayList; import java.util.List;  */\*\*  \* Clase que se encarga de llenar con los datos de los mensajes enviados  \* los Card View Mensaje.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class AdapterMensaje extends RecyclerView.Adapter<HolderMensaje> {  private List<Mensaje> list = new ArrayList<>();  private Context context;   */\*\*  \* Constructor con parámetros de la clase AdpaterMensaje.  \*  \** ***@param*** *context Context de un Activity  \*/* public AdapterMensaje(Context context) {  this.context = context;  }   */\*\*  \* Método que añade elementos a la lista, añade mensajes.  \*  \** ***@param*** *mensaje Mensaje  \*/* public void addMensaje(Mensaje mensaje){  list.add(mensaje);  notifyItemInserted(list.size());  }   */\*\*  \* Método que se encarga de inicializar los elementos del HolderMensaje.  \*  \** ***@param*** *parent ViewGroup  \** ***@param*** *viewType Tipo de Vista  \** ***@return*** *HolderMensaje  \*/* @NonNull  @Override  public HolderMensaje onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  View view = LayoutInflater.*from*(context).inflate(R.layout.*card\_view\_mensajes*,parent,false);  return new HolderMensaje(view);  }   */\*\*  \* Método que infla o llena de datos el card view mensaje.  \*  \** ***@param*** *holder HolderMensaje  \** ***@param*** *position Posición en la lista de mensajes  \*/* @Override  public void onBindViewHolder(@NonNull HolderMensaje holder, int position) {  holder.getNombre().setText(list.get(position).getNombre());  holder.getMensaje().setText(list.get(position).getMensaje());  holder.getHora().setText(list.get(position).getHora());  Glide.*with*(context).load(list.get(position).getFotoPerfil()).into(holder.getFotoMensaje());   if("2".equals(list.get(position).getTipoMensaje())){  holder.getImagenMensaje().setVisibility(View.*VISIBLE*);  holder.getMensaje().setVisibility(View.*VISIBLE*);  Glide.*with*(context).load(list.get(position).getUrlFoto()).into(holder.getImagenMensaje());   }else{  holder.getImagenMensaje().setVisibility(View.*GONE*);  holder.getMensaje().setVisibility(View.*VISIBLE*);  }   }   */\*\*  \* Método que obtiene el número de elementos en la lista.  \*  \** ***@return*** *Número de mensajes.  \*/* @Override  public int getItemCount() {  return list.size();  } } |
|  |
|  |
|  |
|  |
|  |
|  |
| **LoginActivity.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc;  import androidx.annotation.Nullable; import androidx.appcompat.app.AppCompatActivity; import android.content.Intent; import android.net.Uri; import android.os.Bundle; import android.view.View; import android.widget.Button; import android.widget.EditText; import android.widget.ImageButton; import android.widget.ImageView; import android.widget.Toast;  */\*\*  \* Clase que permite instanciar y manejar todos los elementos de la  \* vista o interfaz de para el registro de usuarios, implementa la interfaz iChat.iVistaLogin.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class LoginActivity extends AppCompatActivity implements iChat.iVistaLogin{  private EditText txtNombre;  private EditText txtApellido;  private Button btnIngresar;  private ImageButton btnFoto;  private ImageView foto;  private String nombre="", apellido="";  private Uri u;  private Presentador Presentador;  */\*\*  \* Método que se encarga de inicializar los elementos de la vista.  \*  \** ***@param*** *savedInstanceState Objeto LoginActivity actual  \*/* @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_login*);  Presentador = new Presentador(this);  txtNombre = (EditText)findViewById(R.id.*txtRegNombre*);  txtApellido = (EditText)findViewById(R.id.*txtRegApellido*);  btnIngresar = (Button) findViewById(R.id.*btnRegIngresar*);  btnFoto = (ImageButton) findViewById(R.id.*btnRegImagen*);  foto = (ImageView) findViewById(R.id.*imagenregistro*);   */\*\*  \* Botón que llama al presentador para enviar los datos del usuario para que se registre en la base.  \*/* btnIngresar.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  nombre=txtNombre.getText().toString();  apellido=txtApellido.getText().toString();  Presentador.registrarUsuarioP(nombre,apellido,u,new LoginActivity());  btnIngresar.setEnabled(false);  }  });   */\*\*  \* Botón que abre la galería del teléfono pasa seleccionar una foto del mismo.  \*/* btnFoto.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  subirFoto();  }  });  }  */\*\*  \* Método que permite abrir la galería del teléfono y seleccionar una foto.  \*/* public void subirFoto(){  Intent i = new Intent(Intent.*ACTION\_GET\_CONTENT*);  i.setType("image/\*");  i.putExtra(Intent.*EXTRA\_LOCAL\_ONLY*,true);  startActivityForResult(Intent.*createChooser*(i,"Selecciona foto"),1);  }  */\*\*  \* Método que determina si el usuario actual se encuentra registrado en la base.  \*/* @Override  protected void onStart() {  super.onStart();  if (Presentador.getModelo().firebaseAuth.getCurrentUser() !=null){  startActivity(new Intent(LoginActivity .this, HomeActivity.class));  finish();  }  }  */\*\*  \* Método que muestra en pantalla la foto seleccionada por el usuario.  \*  \** ***@param*** *requestCode  \** ***@param*** *resultCode  \** ***@param*** *data  \*/* @Override  protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {  super.onActivityResult(requestCode, resultCode, data);  if (requestCode==1 && resultCode == *RESULT\_OK* && data!=null ){  u = data.getData();  foto.setImageURI(u);  }  }   */\*\*  \* Método que permite cambiar del LoginActivity al HomeActivity.  \*  \** ***@param*** *confirmacion Confirmación de que el usuario se registró e la base.  \** ***@param*** *mensaje Mensaje de éxito u error en el registro.  \*/* @Override  public void cambiarActivityV(boolean confirmacion, String mensaje) {  if (confirmacion){  startActivity(new Intent(this, HomeActivity.class));  finish();  }else {  btnIngresar.setEnabled(true);  Toast.*makeText*(getApplicationContext(),mensaje,Toast.*LENGTH\_LONG*).show();  }   } } |
| **AdapterUsuario.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc;  import android.content.Context; import android.content.Intent; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup; import android.widget.ImageView; import android.widget.TextView;  import androidx.annotation.NonNull; import androidx.recyclerview.widget.RecyclerView;  import com.bumptech.glide.Glide;  import java.util.List;  */\*\*  \* Clase que se encarga de llenar con los datos de los usuarios registrados  \* los Card View Usuario.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class AdapterUsuario extends RecyclerView.Adapter<AdapterUsuario.ViewHolder> {   private Context context;  private List<Usuario> usuarios;   */\*\*  \* Constructor con parámetros de la clase AdapterUsuario.  \*  \** ***@param*** *context Context del Activity  \** ***@param*** *usuarios Lista de usuarios  \*/* public AdapterUsuario (Context context, List<Usuario>usuarios){  this.usuarios = usuarios;  this.context = context;  }   */\*\*  \* étodo que se encarga de inicializar los elementos del Holder para los Usuarios.  \*  \** ***@param*** *parent ViewGroup  \** ***@param*** *viewType Tipo de vista  \** ***@return*** *Holder para los card view usuarios.  \*/* @NonNull  @Override  public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  View view = LayoutInflater.*from*(context).inflate(R.layout.*card\_view\_usuario*,parent,false);  return new AdapterUsuario.ViewHolder(view);  }   */\*\*  \*  \** ***@param*** *holder Holder de Usuarios  \** ***@param*** *position Posición en la lista de usuarios  \*/* @Override  public void onBindViewHolder(@NonNull ViewHolder holder, int position) {  final Usuario usuario = usuarios.get(position);  holder.username.setText(usuario.getNombre() + usuario.getApellido());  Glide.*with*(context).load(usuario.getFoto()).into(holder.foto);   holder.itemView.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(context, MainActivity.class);  intent.putExtra("userid", usuario.getId());  intent.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  context.startActivity(intent);  }  });   }   */\*\*  \* Método que obtiene el número de elementos en la lista.  \*  \** ***@return*** *Número de usuarios en la lista.  \*/* @Override  public int getItemCount() {  return usuarios.size();  }   */\*\*  \* Método que obtiene los elementos del card view Usuarios por medio de IDs.  \*/* public class ViewHolder extends RecyclerView.ViewHolder{  public TextView username;  public ImageView foto;   public ViewHolder(@NonNull View itemView) {  super(itemView);  username=itemView.findViewById(R.id.*usuario\_nombre*);  foto=itemView.findViewById(R.id.*usuario\_imagen*);  }  } }  **HolderMensaje.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc;  import android.view.View; import android.widget.ImageView; import android.widget.TextView;  import androidx.annotation.NonNull; import androidx.recyclerview.widget.RecyclerView;  import de.hdodenhof.circleimageview.CircleImageView;  */\*\*  \* Clase que actua como modelo del Card View Mensaje, representa  \* al card view.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class HolderMensaje extends RecyclerView.ViewHolder {   private TextView mensaje;  private TextView nombre;  private TextView hora;  private CircleImageView fotoMensaje;  private ImageView imagenMensaje;   */\*\*  \* Método que obtiene el valor del atributo Mensaje de la clase HolderMensaje.  \*  \** ***@return*** *Mensaje  \*/* public TextView getMensaje() {  return mensaje;  }   */\*\*  \* Método que setea el valor del atributo Mensaje de la clase HolderMensaje  \*  \** ***@param*** *mensaje  \*/* public void setMensaje(TextView mensaje) {  this.mensaje = mensaje;  }   */\*\*  \* Método que obtiene el valor del atributo Nombre de la clase HolderMensaje.  \*  \** ***@return*** *Nombre del remitente del mensaje.  \*/* public TextView getNombre() {  return nombre;  }   */\*\*  \* Método que setea el valor del atributo Nombre de la clase HolderMensaje.  \*  \** ***@param*** *nombre Nombre del usuario remitente del mensaje  \*/* public void setNombre(TextView nombre) {  this.nombre = nombre;  }   */\*\*  \* Método que obtiene el valor del atributo Hora de la clase HolderMensaje  \*  \** ***@return*** *Hora de envío del mensaje.  \*/* public TextView getHora() {  return hora;  }   */\*\*  \* Método que setea el valor del atributo Hora de la clase HolderMensaje.  \*  \** ***@param*** *hora Hora de envío del mensaje.  \*/* public void setHora(TextView hora) {  this.hora = hora;  }   */\*\*  \* Método que obtiene el valor del atributo FotoMensaje de la clase HolderMensaje.  \*  \** ***@return*** *Foto de perfil del usuario que envía el mensaje.  \*/* public CircleImageView getFotoMensaje() {  return fotoMensaje;  }   */\*\*  \* Método que setea el valor del atributo FotoMensaje de la clase HolderMensaje.  \*  \** ***@param*** *fotoMensaje Foto de perfil del usuario que envía el mensaje.  \*/* public void setFotoMensaje(CircleImageView fotoMensaje) {  this.fotoMensaje = fotoMensaje;  }   */\*\*  \* Método que setea cada elemento del HolderMensaje igualandolo a los elementos del card View mensaje  \* por medio de IDs.  \*  \** ***@param*** *itemView itemView  \*/* public HolderMensaje(@NonNull View itemView) {  super(itemView);  mensaje = (TextView) itemView.findViewById(R.id.*mensajemensaje*);  nombre = (TextView) itemView.findViewById(R.id.*nombremensaje*);  hora = (TextView) itemView.findViewById(R.id.*fechamensaje*);  fotoMensaje = (CircleImageView) itemView.findViewById(R.id.*fotomensaje*);  imagenMensaje =(ImageView)itemView.findViewById(R.id.*imagenmensaje*);   }   */\*\*  \* Método que obtiene el valor del atributo ImagenMensaje de la clase HolderMensaje.  \*  \** ***@return*** *Imagen enviada por el usuario como mensaje.  \*/* public ImageView getImagenMensaje() {  return imagenMensaje;  }   */\*\*  \* Método que setea el valor del atributo ImagenMensaje de la clase HolderMensaje.  \*  \** ***@param*** *imagenMensaje Imagen enviada por el usuario.  \*/* public void setImagenMensaje(ImageView imagenMensaje) {  this.imagenMensaje = imagenMensaje;  } }  **HomeActivity.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc;  import androidx.appcompat.app.AppCompatActivity; import androidx.recyclerview.widget.LinearLayoutManager; import androidx.recyclerview.widget.RecyclerView; import android.content.Intent; import android.os.Bundle; import android.view.View; import android.widget.ImageButton; import android.widget.ImageView; import android.widget.TextView;  */\*\*  \* Clase que permite instanciar y manejar todos los elementos de la  \* vista o interfaz HomeActivity que muestra las salas de chat y usuarios.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class HomeActivity extends AppCompatActivity{   private ImageButton btnCerrasSesion;  private TextView txtNombres;  private ImageView imgHome;  RecyclerView usuarios;  private Presentador Presentador;   */\*\*  \* Método que se encarga de inicializar los elementos de la vista.  \*  \** ***@param*** *savedInstanceState Objecto HomeActivity actual  \*/* @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_home*);  txtNombres = (TextView)findViewById(R.id.*txtHomeNombre*);  imgHome = (ImageView)findViewById(R.id.*imgHome*);  usuarios =(RecyclerView)findViewById(R.id.*lista\_usuarios*);  Presentador = new Presentador(this);  LinearLayoutManager linearLayoutManager = new LinearLayoutManager(this);  usuarios.setLayoutManager(linearLayoutManager);  btnCerrasSesion = (ImageButton)findViewById(R.id.*btnCerrarSesion*);   */\*\*  \* Botón para cerrar la sesión del usuario acual conetado.  \*/* btnCerrasSesion.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Presentador.cerrarSesionP();  startActivity(new Intent(HomeActivity.this,LoginActivity.class));  finish();  }  });   Presentador.obtenerInfoP(txtNombres,imgHome, getApplicationContext());  Presentador.leerUsuariosP(getApplicationContext(), usuarios);   } }  **iChat.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc; import android.app.Activity; import android.content.Context; import android.content.Intent; import android.net.Uri; import android.widget.ImageView; import android.widget.TextView;  import androidx.annotation.Nullable; import androidx.recyclerview.widget.RecyclerView;  import java.util.List;  */\*\*  \* Interfaz principal del programa, donde se declara todos los métodos  \* que se van a usar para que puedan ser implementados por las diferentes clases.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public interface iChat {   */\*\*  \* Interfaz iVistaLogin que declara los métodos que van a ser utilizados por las clases  \* que implementen esta interfaz.  \*/* interface iVistaLogin{  void cambiarActivityV(boolean confirmacion, String mensaje);  }   */\*\*  \* Interfaz iPresentador que declara los métodos que van a ser utilizados por las clases  \* que implementen esta interfaz.  \*/* interface iPresentador{  void cambiarActivityP(boolean confirmacion, String mensaje);  void registrarUsuarioP(String nombre, String apellido, Uri u, Activity activity);  void cerrarSesionP();  void obtenerInfoP(TextView textView, ImageView imageView, Context context);  void leerUsuariosP(Context context, RecyclerView users);  void enviarMensajeP(String mensaje, String tipo);  void enviarFotoP(int requestCode, int resultCode, @Nullable Intent data);  void infoUsuarioP();  void infoContactoP(String user\_id,TextView textView, ImageView imageView, Context context);  void leerMensajesP(Context context, RecyclerView mensajes);   }   */\*\*  \* Interfaz iModelo que declara los métodos que van a ser utilizados por las clases  \* que implementen esta interfaz.  \*/* interface iModelo{  void registrarUsuarioM(String nombre, String apellido, Uri u, Activity activity);  void cerrarSesionM();  void obtenerInfoM(TextView textView, ImageView imageView, Context context);  void leerUsuariosM(Context context, RecyclerView users);  void enviarMensajeM(String mensaje, String tipo);  void enviarFotoM(int requestCode, int resultCode, @Nullable Intent data);  void infoUsuarioM();  void infoContactoM(String user\_id,TextView textView, ImageView imageView, Context context);  void leerMensajesM(Context context, RecyclerView mensajes);  } } |
| **MainActivity.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc; import androidx.annotation.Nullable; import androidx.appcompat.app.AppCompatActivity; import androidx.recyclerview.widget.LinearLayoutManager; import androidx.recyclerview.widget.RecyclerView; import android.content.Intent; import android.os.Bundle; import android.view.View; import android.widget.EditText; import android.widget.ImageButton; import android.widget.TextView; import com.google.firebase.auth.FirebaseAuth; import com.google.firebase.database.DatabaseReference; import com.google.firebase.database.FirebaseDatabase; import com.google.firebase.storage.FirebaseStorage; import de.hdodenhof.circleimageview.CircleImageView; */\*\*  \* Clase que permite instanciar y manejar todos los elementos de la  \* vista o interfaz MainActivity que muestra una sala de chat entre 2 usuarios.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class MainActivity extends AppCompatActivity {  CircleImageView fotoPerfil;  TextView nombreUsuario;  RecyclerView mensajes;  EditText txtmensaje;  ImageButton btnenviar, btnfoto;  Intent intent;  private Presentador Presentador;  private FirebaseAuth firebaseAuth;  private AdapterMensaje adapterMensajes;  private FirebaseDatabase database;  private DatabaseReference databaseReference;  private DatabaseReference databaseReferenceUsuario;  private DatabaseReference databaseReferenceBuscarUsuario;  private DatabaseReference databaseReferenceMensaje;  private FirebaseStorage storage;  String msj;   */\*\*  \* Método que se encarga de inicializar los elementos de la vista.  \*  \** ***@param*** *savedInstanceState Objeto LoginActivity actual  \*/* @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  Presentador = new Presentador(this);  fotoPerfil = (CircleImageView) findViewById(R.id.perfil);  nombreUsuario = (TextView) findViewById(R.id.nombre);  mensajes = (RecyclerView) findViewById(R.id.mensajes);  txtmensaje = (EditText) findViewById(R.id.txtMensaje);  btnenviar = (ImageButton) findViewById(R.id.btnEnviar);  btnfoto = (ImageButton) findViewById(R.id.btnimagen);  adapterMensajes=new AdapterMensaje(this);  LinearLayoutManager linearLayoutManager = new LinearLayoutManager(this);  mensajes.setLayoutManager(linearLayoutManager);  mensajes.setAdapter(adapterMensajes);   intent = getIntent();  Presentador.getModelo().setUser\_id(intent.getStringExtra("userid"));  final String user\_id = intent.getStringExtra("userid");  database = FirebaseDatabase.getInstance();  firebaseAuth=FirebaseAuth.*getInstance*();  databaseReferenceUsuario = database.getReference("Usuarios").child(user\_id);  databaseReferenceMensaje = database.getReference("chat");  storage=FirebaseStorage.*getInstance*();   */\*\*  \* Botón que llama al presentador para enviar los datos del mensaje.  \*/* btnenviar.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Presentador.enviarMensajeP(txtmensaje.getText().toString(), "1");  txtmensaje.setText("");  }  });   */\*\*  \* Botón que permite abrir la galería del teléfono para seleccionar una foto.  \*/* btnfoto.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent i = new Intent(Intent.ACTION\_GET\_CONTENT);  i.setType("image/\*");  i.putExtra(Intent.EXTRA\_LOCAL\_ONLY,true);  startActivityForResult(Intent.createChooser(i,"Selecciona foto"),1);  }  });   Presentador.infoUsuarioP();  Presentador.infoContactoP(user\_id,nombreUsuario,fotoPerfil,getApplicationContext());  Presentador.leerMensajesP(getApplicationContext(),mensajes);   }   */\*\*  \* Método Scroll que permite movilizarse o desplazarse por la pantalla para visualizar  \* los mensajes.  \*/* private void setScrollbar(){  mensajes.scrollToPosition(adapterMensajes.getItemCount()-1);  }   */\*\*  \* Método que obtiene la data de la foto seleccionada y la envia por medio del  \* presentador.  \*  \** ***@param*** *requestCode  \** ***@param*** *resultCode  \** ***@param*** *data  \*/* @Override  protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {  super.onActivityResult(requestCode, resultCode, data);  Presentador.enviarFotoP(requestCode,resultCode,data);  } } |
| **Mensaje.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc; */\*\*  \* Clase que representa un mensaje del chat.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class Mensaje {  private String mensaje;  private String nombre;  private String fotoPerfil;  private String tipoMensaje;  private String hora;  private String urlFoto;  */\*\*  \* Constructor vacío de la clase Mensaje.  \*/* public Mensaje() {  }  */\*\*  \* Constructor con parámetros de la clase Mensaje para enviar texto.  \*  \** ***@param*** *mensaje Contenido del mensaje  \** ***@param*** *nombre Nombre del usuario que envia el mensaje  \** ***@param*** *fotoPerfil Foto de perfil del usuario que envia el mensaje  \** ***@param*** *tipoMensaje Tipo de mensaje enviado  \** ***@param*** *hora Hora del envio del mensaje  \*/* public Mensaje(String mensaje, String nombre, String fotoPerfil, String tipoMensaje, String hora) {  this.mensaje = mensaje;  this.nombre = nombre;  this.fotoPerfil = fotoPerfil;  this.tipoMensaje = tipoMensaje;  this.hora = hora;  }  */\*\*  \* Constructor con parámetros de la clase Mensaje para enviar una imagen.  \*  \** ***@param*** *mensaje Contenido del mensaje  \** ***@param*** *nombre Nombre del usuario que envia el mensaje  \** ***@param*** *fotoPerfil Foto de perfil del usuario que envia el mensaje  \** ***@param*** *tipoMensaje Tipo de mensaje enviado  \** ***@param*** *hora Hora del envio del mensaje  \** ***@param*** *urlFoto Foro enviada  \*/* public Mensaje(String mensaje, String nombre, String fotoPerfil, String tipoMensaje, String hora, String urlFoto) {  this.mensaje = mensaje;  this.nombre = nombre;  this.fotoPerfil = fotoPerfil;  this.tipoMensaje = tipoMensaje;  this.hora = hora;  this.urlFoto = urlFoto;  }  */\*\*  \* Método que obtiene el valor del atributo Mensaje de la clase Mensaje.  \*  \** ***@return*** *Contenido del Mensaje.  \*/* public String getMensaje() {  return mensaje;  }  */\*\*  \* Método que setea el valor del atributo Mensaje de la clase Mensaje.  \*  \** ***@param*** *mensaje Contenido del mensaje  \*/* public void setMensaje(String mensaje) {  this.mensaje = mensaje;  }  */\*\*  \* Método que obtiene el valor del atributo Nombre de la clase Mensaje.  \*  \** ***@return*** *Nombre del usuario que envia el mensaje.  \*/* public String getNombre() {  return nombre;  }  */\*\*  \* Método que setea el valor del atributo Nombre de la clase Mensaje.  \*  \** ***@param*** *nombre Nombre del usuario que envia el mensaje.  \*/* public void setNombre(String nombre) {  this.nombre = nombre;  }   */\*\*  \* Método que obtiene el valor del atributo fotoPerfil de la clase Mensaje.  \*  \** ***@return*** *Foto de perfil del usuario que envia el mensaje.  \*/* public String getFotoPerfil() {  return fotoPerfil;  }  */\*\*  \* Método que setea el valor del atributo fotoPerfil de la clase Mensaje.  \*  \** ***@param*** *fotoPerfil Foto de perfil del usuario que envia el mensaje.  \*/* public void setFotoPerfil(String fotoPerfil) {  this.fotoPerfil = fotoPerfil;  }  */\*\*  \* Método que obtiene el valor del atributo tipoMensaje de la clase Mensaje.  \*  \** ***@return*** *Tipo de mensaje que se envia.  \*/* public String getTipoMensaje() {  return tipoMensaje;  }  */\*\*  \* Método que setea el valor del atributo tipoMensaje de la clase Mensaje.  \*  \** ***@param*** *tipoMensaje Tipo de mensaje que se envia.  \*/* public void setTipoMensaje(String tipoMensaje) {  this.tipoMensaje = tipoMensaje;  }  */\*\*  \* Método que obtiene el valor del atributo Hora de la clase Mensaje.  \*  \** ***@return*** *Hora de envío del mensaje.  \*/* public String getHora() {  return hora;  }  */\*\*  \* Método que setea el valor del atributo Hora de la clase Mensaje.  \*  \** ***@param*** *hora Hora de envío del mensaje.  \*/* public void setHora(String hora) {  this.hora = hora;  }  */\*\*  \* Método que obtiene el valor del atributo urlFoto de la clase Mensaje.  \*  \** ***@return*** *Foto enviada.  \*/* public String getUrlFoto() {  return urlFoto;  }  */\*\*  \* Método que setea el valor del atributo urlFoto de la clase Mensaje.  \*  \** ***@param*** *urlFoto Foto enviada por el chat.  \*/* public void setUrlFoto(String urlFoto) {  this.urlFoto = urlFoto;  } } |
| **Modelo.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc; import android.app.Activity; import android.content.Context; import android.content.Intent; import android.net.Uri; import android.util.Log; import android.widget.ImageView; import android.widget.TextView; import androidx.annotation.NonNull; import com.bumptech.glide.Glide; import com.google.android.gms.tasks.Continuation; import com.google.android.gms.tasks.OnCompleteListener; import com.google.android.gms.tasks.Task; import com.google.firebase.auth.AuthResult; import com.google.firebase.auth.FirebaseAuth; import com.google.firebase.database.ChildEventListener; import com.google.firebase.database.DataSnapshot; import com.google.firebase.database.DatabaseError; import com.google.firebase.database.DatabaseReference; import com.google.firebase.database.FirebaseDatabase; import com.google.firebase.database.ValueEventListener; import com.google.firebase.storage.FirebaseStorage; import com.google.firebase.storage.StorageReference; import com.google.firebase.storage.UploadTask; import androidx.annotation.Nullable; import androidx.recyclerview.widget.RecyclerView; import java.text.SimpleDateFormat; import java.util.ArrayList; import java.util.Calendar; import java.util.HashMap; import java.util.List; import java.util.Map;  */\*\*  \* Clase que contiene toda la lógica del programa, la misma realiza todas las operaciones  \* e implementa la interfaz iChat.iModelo.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class Modelo implements iChat.iModelo {  private Presentador Presentador;  FirebaseAuth firebaseAuth;  DatabaseReference databaseReference;  private DatabaseReference databaseReferenceSala;  private FirebaseStorage storage;  private StorageReference storageReference;  private List<Usuario> usuarioList;  private AdapterUsuario adapterUsuarios;  private String nombre\_u="", apellid\_u="", foto\_u="", user\_id;  private DatabaseReference databaseReference2;  private AdapterMensaje adapterMensajes;  private DatabaseReference databaseReferenceBuscarUsuario;  private DatabaseReference databaseReferenceUsuario;  private DatabaseReference databaseReferenceMensaje;   */\*\*  \* Constructor vacío de la clase Modelo.  \*/* public Modelo(){   }   */\*\*  \* Constructor con parámetros de la clase Modelo.  \*  \** ***@param*** *Presentador Objecto de la clase Presentador  \*/* public Modelo(Presentador Presentador) {  this.Presentador = Presentador;  firebaseAuth = FirebaseAuth.*getInstance*();  databaseReference = FirebaseDatabase.*getInstance*().getReference();  databaseReferenceBuscarUsuario = FirebaseDatabase.*getInstance*().getReference("Usuarios").child(firebaseAuth.getCurrentUser().getUid());  databaseReferenceMensaje = FirebaseDatabase.*getInstance*().getReference("chat");  databaseReferenceSala = FirebaseDatabase.*getInstance*().getReference("chat");  storage=FirebaseStorage.*getInstance*();  usuarioList= new ArrayList<>();  }   */\*\*  \* Método que obtiene el valor del atributo User\_id de la clase Modelo  \*  \** ***@return*** *Id del usuario logeado.  \*/* public String getUser\_id() {  return user\_id;  }   */\*\*  \* Método que setea el valor del atributo User\_id de la clase Modelo.  \*  \** ***@param*** *user\_id Id del usuario  \*/* public void setUser\_id(String user\_id) {  this.user\_id = user\_id;  }   */\*\*  \* Realiza el registro del usuario en la base de datos.  \*  \** ***@param*** *nombre Nombre del usuario  \** ***@param*** *apellido Apellido del Usuario  \** ***@param*** *u Uri de la foto de perfil del Usuario  \** ***@param*** *activity Activity  \*/* @Override  public void registrarUsuarioM(final String nombre, final String apellido, final Uri u, Activity activity) {  firebaseAuth.signInAnonymously().addOnCompleteListener(activity, new OnCompleteListener<AuthResult>() {  @Override  public void onComplete(@NonNull Task<AuthResult> task) {  if(task.isSuccessful()){  storageReference = storage.getReference("perfiles");  final StorageReference foto = storageReference.child(u.getLastPathSegment() + Math.*random*()\*10);  foto.putFile(u).continueWithTask(new Continuation<UploadTask.TaskSnapshot, Task<Uri>>() {  @Override  public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception {  if(!task.isSuccessful()){  throw new Exception();  }  return foto.getDownloadUrl();  }  }).addOnCompleteListener(new OnCompleteListener<Uri>() {  @Override  public void onComplete(@NonNull Task<Uri> task) {  if (task.isSuccessful()){  String fotoperfil = task.getResult().toString();  String id = firebaseAuth.getCurrentUser().getUid();  Map<String,Object> map = new HashMap<>();  map.put("nombre",nombre);  map.put("apellido",apellido);  map.put("foto",fotoperfil);   databaseReference.child("Usuarios").child(id).setValue(map).addOnCompleteListener(new OnCompleteListener<Void>() {  @Override  public void onComplete(@NonNull Task<Void> task2) {  if(task2.isSuccessful()){  Presentador.cambiarActivityP(true, "OKEY BOOMER");  }  else {  Presentador.cambiarActivityP(false, "ERROR EN LA BASE");   }  }  });  }  else {  Presentador.cambiarActivityP(false, "ERROR EN SUBIR LA FOTO");  }  }  });  }else {  Presentador.cambiarActivityP(false, "ERROR AL CREAR USUARIO");    }  }  });  }   */\*\*  \* Método que cierra la sesión del usuario actual.  \*/* @Override  public void cerrarSesionM() {  firebaseAuth.signOut();  }   */\*\*  \* Método que obtiene la información del usuario logeado, para llenar la vista Home con los  \* datos.  \*  \** ***@param*** *textView TextView del Activity  \** ***@param*** *imageView ImageView del activity  \** ***@param*** *context Context del Activity  \*/* @Override  public void obtenerInfoM(final TextView textView, final ImageView imageView, final Context context) {   String id = firebaseAuth.getCurrentUser().getUid();   databaseReference.child("Usuarios").child(id).addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  //final List<String> informacion = new ArrayList<String>();  String nombre = dataSnapshot.child("nombre").getValue().toString();  String apellido = dataSnapshot.child("apellido").getValue().toString();  String foto = dataSnapshot.child("foto").getValue().toString();   textView.setText(nombre + " " + apellido);  Glide.*with*(context).load(foto).into(imageView);  }   @Override  public void onCancelled(@NonNull DatabaseError databaseError) {   }  });   }   */\*\*  \* Método que obtiene los datos de todos los usuarios registrados para mostrar en  \* la vista HomeActivity.  \*  \** ***@param*** *context Context del Activity  \** ***@param*** *users RecyclerView del activity.  \*/* @Override  public void leerUsuariosM(final Context context, final RecyclerView users) {  databaseReference.child("Usuarios").addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  usuarioList.clear();  for (DataSnapshot snapshot: dataSnapshot.getChildren()){  Usuario usuario = snapshot.getValue(Usuario.class);  usuario.setId(snapshot.getKey());   if(!usuario.getId().equals(firebaseAuth.getUid())){  usuarioList.add(usuario);  }   }  adapterUsuarios = new AdapterUsuario(context,usuarioList);  users.setAdapter(adapterUsuarios);   }   @Override  public void onCancelled(@NonNull DatabaseError databaseError) {   }  });  }   */\*\*  \* Método que envía un mensaje de texto y lo registra en la base.  \*  \** ***@param*** *mensaje Contenido del Mensaje  \** ***@param*** *tipo Tipo de Mensaje  \*/* @Override  public void enviarMensajeM(String mensaje, String tipo) {  Calendar c = Calendar.*getInstance*();  SimpleDateFormat df = new SimpleDateFormat("HH:mm:ss");  String formattedDate = df.format(c.getTime());  Log.*e*("Mensaje: ", mensaje);  databaseReference2.push().setValue(new Mensaje(mensaje, nombre\_u +" " + apellid\_u,foto\_u,tipo,formattedDate));   }   */\*\*  \* Método que envía la imagen seleccionada por el usuario.  \*  \** ***@param*** *requestCode Request Code  \** ***@param*** *resultCode Result Code  \** ***@param*** *data Data de la imagen a enviar  \*/* @Override  public void enviarFotoM(int requestCode, int resultCode, @Nullable Intent data) {   if (requestCode==1 && resultCode == -1 && data!=null ){  Uri u = data.getData();  storageReference = storage.getReference("imagenes");  final StorageReference foto = storageReference.child(u.getLastPathSegment());   foto.putFile(u).continueWithTask(new Continuation<UploadTask.TaskSnapshot, Task<Uri>>() {  @Override  public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception {  Log.*e*("FOTO: ", "ENTRA");  if(!task.isSuccessful()){  throw new Exception();  }  return foto.getDownloadUrl();  }  }).addOnCompleteListener(new OnCompleteListener<Uri>() {  @Override  public void onComplete(@NonNull Task<Uri> task) {  if (task.isSuccessful()){  Calendar c = Calendar.*getInstance*();  SimpleDateFormat df = new SimpleDateFormat("HH:mm:ss");  String formattedDate = df.format(c.getTime());  Mensaje m = new Mensaje(" ", nombre\_u +" " + apellid\_u,foto\_u,"2",formattedDate,task.getResult().toString());  databaseReference2.push().setValue(m);  }  }  });   }  }   */\*\*  \* Método que obtiene la información del usuario logeado.  \*/* @Override  public void infoUsuarioM() {  databaseReferenceBuscarUsuario = FirebaseDatabase.*getInstance*().getReference("Usuarios").child(firebaseAuth.getCurrentUser().getUid());  databaseReferenceBuscarUsuario.addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  Usuario usuario = dataSnapshot.getValue(Usuario.class);  nombre\_u = usuario.getNombre();  apellid\_u = usuario.getApellido();  foto\_u = usuario.getFoto();  }   @Override  public void onCancelled(@NonNull DatabaseError databaseError) {   }  });  }   */\*\*  \* Método que obtiene la información del usuario con el cual se establecerá la sesión o  \* sala de chat.  \*  \** ***@param*** *user\_id Id del usuario  \** ***@param*** *textView textView del Activity  \** ***@param*** *imageView ImageView del Activity  \** ***@param*** *context Context del Activity  \*/* @Override  public void infoContactoM(String user\_id, final TextView textView, final ImageView imageView, final Context context) {  this.user\_id = user\_id;  databaseReferenceUsuario = FirebaseDatabase.*getInstance*().getReference("Usuarios").child(user\_id);  databaseReferenceUsuario.addValueEventListener(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  Usuario usuario = dataSnapshot.getValue(Usuario.class);  textView.setText(usuario.getNombre() + " " + usuario.getApellido());  Glide.*with*(context).load(usuario.getFoto()).into(imageView);  }   @Override  public void onCancelled(@NonNull DatabaseError databaseError) {   }  });  }   */\*\*  \* Método que obtienen los mensajes anteriores de una sala de chat, y los  \* muestra en la vista.  \*  \** ***@param*** *context Context del Activity  \** ***@param*** *mensajes RecyclerView del Activity  \*/* @Override  public void leerMensajesM(Context context, final RecyclerView mensajes) {  final String chat\_id =firebaseAuth.getUid() + user\_id;  final String chat\_id2 = user\_id + firebaseAuth.getUid();  databaseReferenceMensaje = FirebaseDatabase.*getInstance*().getReference("chat");  adapterMensajes=new AdapterMensaje(context);  adapterMensajes.registerAdapterDataObserver(new RecyclerView.AdapterDataObserver() {  @Override  public void onItemRangeInserted(int positionStart, int itemCount) {  super.onItemRangeInserted(positionStart, itemCount);  mensajes.scrollToPosition(adapterMensajes.getItemCount()-1);  }  });   databaseReferenceMensaje.addListenerForSingleValueEvent(new ValueEventListener() {  @Override  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  Boolean bandera = false;  for (DataSnapshot postSnapshot: dataSnapshot.getChildren()) {  if(postSnapshot.getKey().equals(chat\_id) || postSnapshot.getKey().equals(chat\_id2) ){  databaseReference2 = FirebaseDatabase.*getInstance*().getReference("chat").child(postSnapshot.getKey());  salaExisente();  bandera = true;  break;  }  }  if (!bandera){  databaseReference2 = FirebaseDatabase.*getInstance*().getReference("chat").child(chat\_id);  }  }   @Override  public void onCancelled(@NonNull DatabaseError databaseError) {   }  });   mensajes.setAdapter(adapterMensajes);  }   */\*\*  \* Método que verifica si la sala de chat entre dos usuarios ya existía anteriormente.  \*/* public void salaExisente(){   databaseReference2.addChildEventListener(new ChildEventListener() {  @Override  public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {  Mensaje m = dataSnapshot.getValue(Mensaje.class);  adapterMensajes.addMensaje(m);  }   @Override  public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {   }   @Override  public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {   }   @Override  public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {   }   @Override  public void onCancelled(@NonNull DatabaseError databaseError) {   }  });  }   */\*\*  \* Método que obtiene el valor del atributo FirebaseAuth de la clase Modelo.  \*  \** ***@return*** *FirebaseAuth  \*/* public FirebaseAuth getFirebaseAuth() {  return firebaseAuth;  }   */\*\*  \* Método que setea el valor del atributo FirebaseAuth de la clase Modelo.  \*  \** ***@param*** *firebaseAuth  \*/* public void setFirebaseAuth(FirebaseAuth firebaseAuth) {  this.firebaseAuth = firebaseAuth;  } } |
| **Presentador.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc; import android.app.Activity; import android.content.Context; import android.content.Intent; import android.net.Uri; import android.util.Log; import android.widget.ImageView; import android.widget.TextView; import androidx.annotation.Nullable; import androidx.recyclerview.widget.RecyclerView; import java.util.ArrayList; import java.util.List;  */\*\*  \* Clase que actúa como intermediario entre la Vista y  \* el Modelo de la aplicación, implementa la interfaz iChat.iPresentador.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class Presentador implements iChat.iPresentador {  private LoginActivity iVistaLogin;  private HomeActivity iVistaHome;  private MainActivity iVistaMain;  private Modelo Modelo;  */\*\*  \* Constructor vacío de la clase Presentador.  \*/* public Presentador(){  }   */\*\*  \* Constructor con parámetros de la clase Presentador.  \*  \** ***@param*** *iVistaLogin Vista que implementa la interfaz iChat.iVistaLogin  \*/* public Presentador(LoginActivity iVistaLogin){  this.iVistaLogin = iVistaLogin;  this.Modelo =new Modelo(this);  }   */\*\*  \* Constructor con parámetros de la clase Presentador.  \*  \** ***@param*** *iVistaHome Vista que implementa la interfaz iChat.iVistaHome  \*/* public Presentador(HomeActivity iVistaHome){  this.iVistaHome = iVistaHome;  this.Modelo =new Modelo(this);  }   */\*\*  \* Constructor con parámetros de la clase Presentador.  \*  \** ***@param*** *iVistaMain Vista que implementa la interfaz iChat.iVistaMain  \*/* public Presentador(MainActivity iVistaMain){  this.iVistaMain = iVistaMain;  this.Modelo =new Modelo(this);  }   */\*\*  \* Método que obtiene el valor del atributo iVistaLogin de la clase Presentador  \*  \** ***@return*** *iVistaLogin  \*/* public LoginActivity getiVistaLogin() {  return iVistaLogin;  }   */\*\*  \* Método que setea el valor del atributo iVistaLogin de la clase Presentador  \*  \** ***@param*** *iVistaLogin  \*/* public void setiVistaLogin(LoginActivity iVistaLogin) {  this.iVistaLogin = iVistaLogin;  }   */\*\*  \* Método que obtiene el valor del atributo Modelo de la clase Presentador  \*  \** ***@return*** *Modelo  \*/* public Modelo getModelo() {  return Modelo;  }   */\*\*  \* Método que setea el valor del atributo Modelo de la clase Presentador  \*  \** ***@param*** *modelo  \*/* public void setModelo(Modelo modelo) {  this.Modelo = modelo;  }   */\*\*  \* Método que permite cambiar de la activity de registro al activityHome si es que  \* el registro del usuario fue exitoso.  \*  \** ***@param*** *confirmacion Confirmación del registro de usuario  \** ***@param*** *mensaje mensaje de error o exito del registro  \*/* @Override  public void cambiarActivityP(boolean confirmacion, String mensaje) {  try{  if(iVistaLogin!=null){  iVistaLogin.cambiarActivityV(confirmacion,mensaje);  }  }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }   */\*\*  \* Método que envia los datos del usuario al modelo para realizar su registro  \* en la base de datos.  \*  \** ***@param*** *nombre Nombre del Usuario  \** ***@param*** *apellido Apellido del Usuario  \** ***@param*** *u Uri de la foro de perfil del usuario  \** ***@param*** *activity Activity de registro  \*/* @Override  public void registrarUsuarioP(String nombre, String apellido, Uri u, Activity activity) {  try{  if(iVistaLogin!=null){  Modelo.registrarUsuarioM( nombre, apellido, u, activity);  }  }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }   */\*\*  \* Método que para cerrar sesión del usuario.  \*/* @Override  public void cerrarSesionP() {  try{  if(iVistaHome!=null){  Modelo.cerrarSesionM();  }  }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }   */\*\*  \* Método que llama al modelo para obtener información del usuario que logeado.  \*  \** ***@param*** *textView TextView del activity  \** ***@param*** *imageView ImageView del activity  \** ***@param*** *context Context del Activity  \*/* @Override  public void obtenerInfoP(TextView textView, ImageView imageView, Context context) {  List<String> adapterUsuario=new ArrayList<String>();  try{  if(iVistaHome!=null){  Modelo.obtenerInfoM(textView, imageView, context);  }   }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }  */\*\*  \* Método que llama al modelo para obtener a los usuarios registrados en la base.  \*  \** ***@param*** *context Context del Activity  \** ***@param*** *users RecyclerView del Activity  \*/* @Override  public void leerUsuariosP(Context context, RecyclerView users) {  AdapterUsuario adapterUsuario = null;  try{  if(iVistaHome!=null){  Modelo.leerUsuariosM(context, users);  }  }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }   */\*\*  \* Método que envia el contenido del mensaje al modelo.  \*  \** ***@param*** *mensaje Contenido del mensaje  \** ***@param*** *tipo Tipo del mensaje  \*/* @Override  public void enviarMensajeP(String mensaje, String tipo) {  try{  if(iVistaMain!=null){  Modelo.enviarMensajeM(mensaje, tipo);  }   }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }   */\*\*  \* Método que envía la data de la imagen al modelo para enviar como mensaje.  \*  \** ***@param*** *requestCode Request Code  \** ***@param*** *resultCode result Code  \** ***@param*** *data Data de la imagen  \*/* @Override  public void enviarFotoP(int requestCode, int resultCode, @Nullable Intent data) {  try{  if(iVistaMain!=null){  Modelo.enviarFotoM(requestCode, resultCode,data);  }   }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }  */\*\*  \* Método que llama al modelo para obtener información del usuario.  \*/* @Override  public void infoUsuarioP() {  try{  if(iVistaMain!=null){  Modelo.infoUsuarioM();  }   }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }  */\*\*  \* Método que llama al modelo para obtener información del usuario con el cual se interactua en el chat.  \*  \** ***@param*** *user\_id Id del usuario  \** ***@param*** *textView TextView del Activity  \** ***@param*** *imageView ImageView del Activity  \** ***@param*** *context Context del Activity  \*/* @Override  public void infoContactoP(String user\_id,TextView textView, ImageView imageView, Context context) {  try{  if(iVistaMain!=null){  Modelo.infoContactoM(user\_id,textView,imageView,context);   }   }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  }  */\*\*  \* Método que llama al modelo para obtener los mensajes enviados anteriormente en un chat específico.  \*  \** ***@param*** *context Context del Activity  \** ***@param*** *mensajes recycleView del Activity  \*/* @Override  public void leerMensajesP(Context context, RecyclerView mensajes) {  try{  if(iVistaMain!=null){  Modelo.leerMensajesM(context,mensajes);  }  }catch (Exception e){  Log.*e*("ERROR", e.getMessage());  }  } } |
| **Usuario.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 16/07/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc;  */\*\*  \* Clase que representa a un usuario del chat.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* public class Usuario {  String id;  String foto;  String nombre;  String apellido;   */\*\*  \* Constructor vacío de la clase usuario.  \*/* public Usuario(){   }   */\*\*  \* Constructor con parámetros de la clase Usuario.  \** ***@param*** *id Id del usuario  \** ***@param*** *foto Foto del Usuario  \** ***@param*** *nombre Nombre del Usuario  \** ***@param*** *apellido Apellido del Usuario  \*/* public Usuario(String id, String foto, String nombre, String apellido) {  this.id = id;  this.foto = foto;  this.nombre = nombre;  this.apellido = apellido;  }   */\*\*  \* Método que obtiene el valor del atributo Id de la clase Usuario.  \*  \** ***@return*** *id del Usuario.  \*/* public String getId() {  return id;  }   */\*\*  \* Método que setea el valor del atributo Id de la clase Usuario.  \*  \** ***@param*** *id del Usuario  \*/* public void setId(String id) {  this.id = id;  }   */\*\*  \* Método que obtiene el valor del atributo foto de la clase Usuario.  \*  \** ***@return*** *foto del Usuario.  \*/* public String getFoto() {  return foto;  }   */\*\*  \* Método que setea el valor del atributo Foto de la clase Usuario.  \*  \** ***@param*** *foto del Usuario  \*/* public void setFoto(String foto) {  this.foto = foto;  }   */\*\*  \* Método que obtiene el valor del atributo Nombre de la clase Usuario.  \*  \** ***@return*** *nombre del Usuario.  \*/* public String getNombre() {  return nombre;  }   */\*\*  \* Método que setea el valor del atributo Nombre de la clase Usuario.  \*  \** ***@param*** *nombre del Usuario  \*/* public void setNombre(String nombre) {  this.nombre = nombre;  }   */\*\*  \* Método que obtiene el valor del atributo Apellido de la clase Usuario.  \*  \** ***@return*** *apellido del Usuario.  \*/* public String getApellido() {  return apellido;  }   */\*\*  \* Método que setea el valor del atributo Apellido de la clase Usuario.  \*  \** ***@param*** *apellido del Usuario  \*/* public void setApellido(String apellido) {  this.apellido = apellido;  } } |

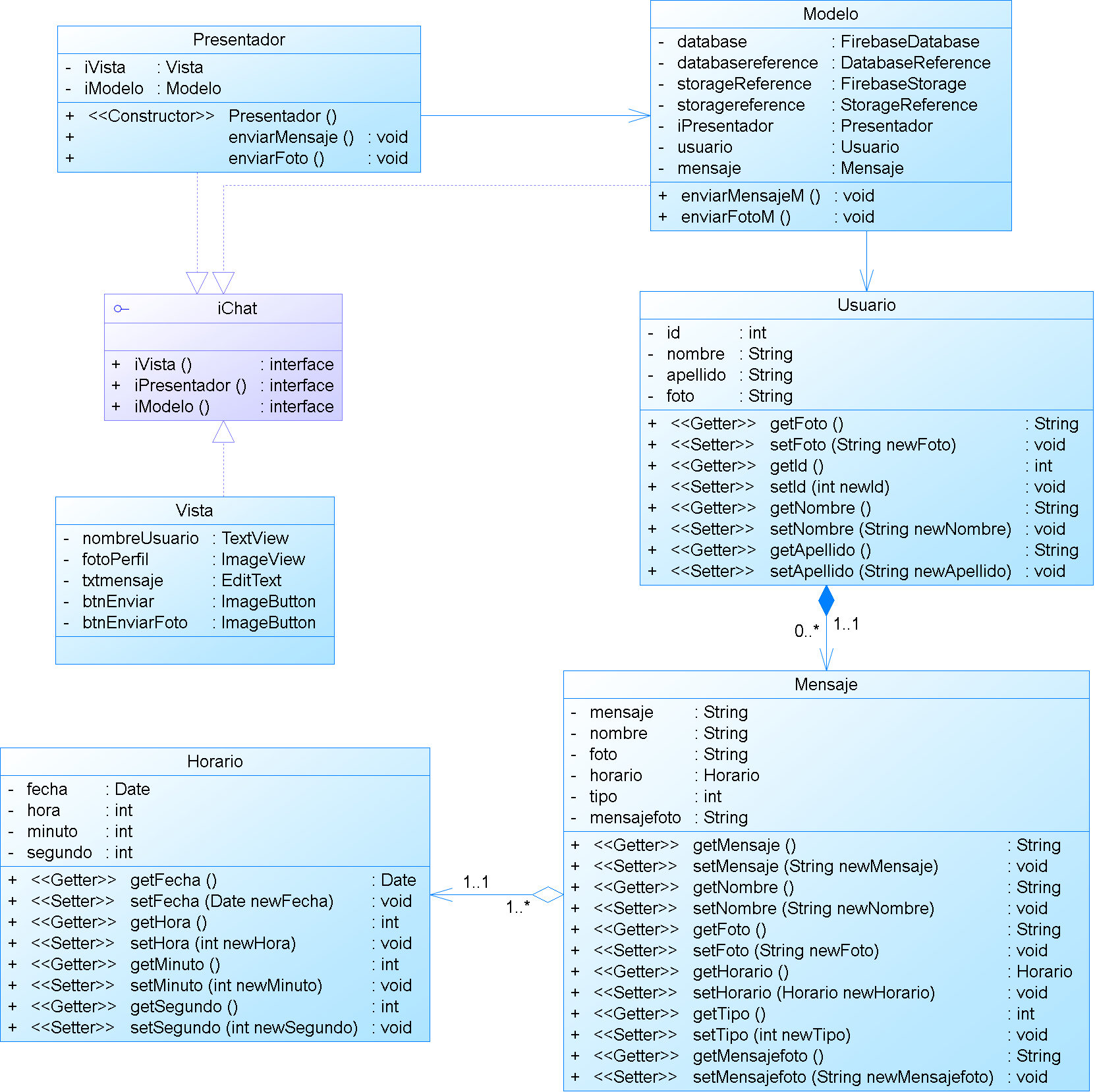
CONCLUSIONES

* Firebase ofrece un catalogo abierto de servicios que mejora y facilita el desarrollo de aplicaciones móviles mediante sus diferentes integraciones y bases de datos en tiempo real.
* Es importante tener en cuenta que Firebase tiene una cuota limite para los planes gratuitos, por lo que se debe tener en cuenta como van a ser usados sus diferentes servicios.
* Al usar el patrón MVP la capa de presentador es independiente de la tecnología de interfaz de usuario.
* En el presente proyecto se hizo uso del patrón MVC lo que nos permitió obtener mayores ventajas al momento de programar en comparación con la programación tradicional sin aplicar patrones de diseño. Nos brindó más rapidez y escalabilidad en cada una de las funciones de los botones.

|  |  |
| --- | --- |
| [1] | "DevelApps," [Online]. Available: http://www.develapps.com/es/noticias/modelo-vista-presentador-mvp-en-android. [Accessed 24 Junio 2020].  [2] «Firebase. » https://firebase.google.com/?hl=es(accedido ago. 03, 2020).  [3] «Pirmeros pasos en Firebasee», *Firebase Docs*. https://firebase.google.com/docs?hl=es (accedido ago. 03, 2020). |
|  |  |

**BIBLIOGRAFÍA**

# **ANEXO N°1: Modelo**

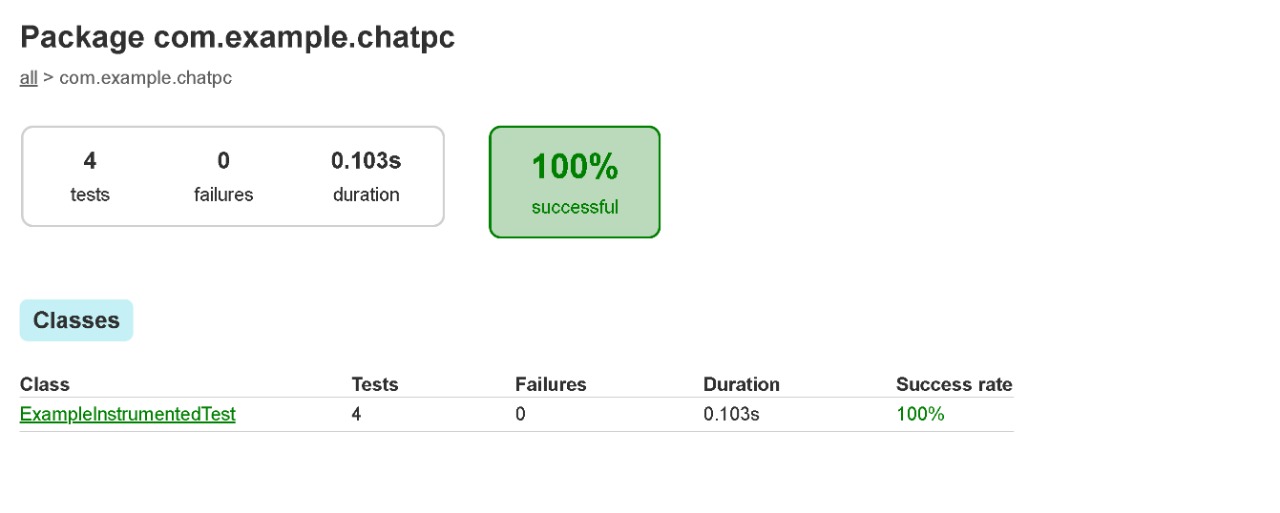


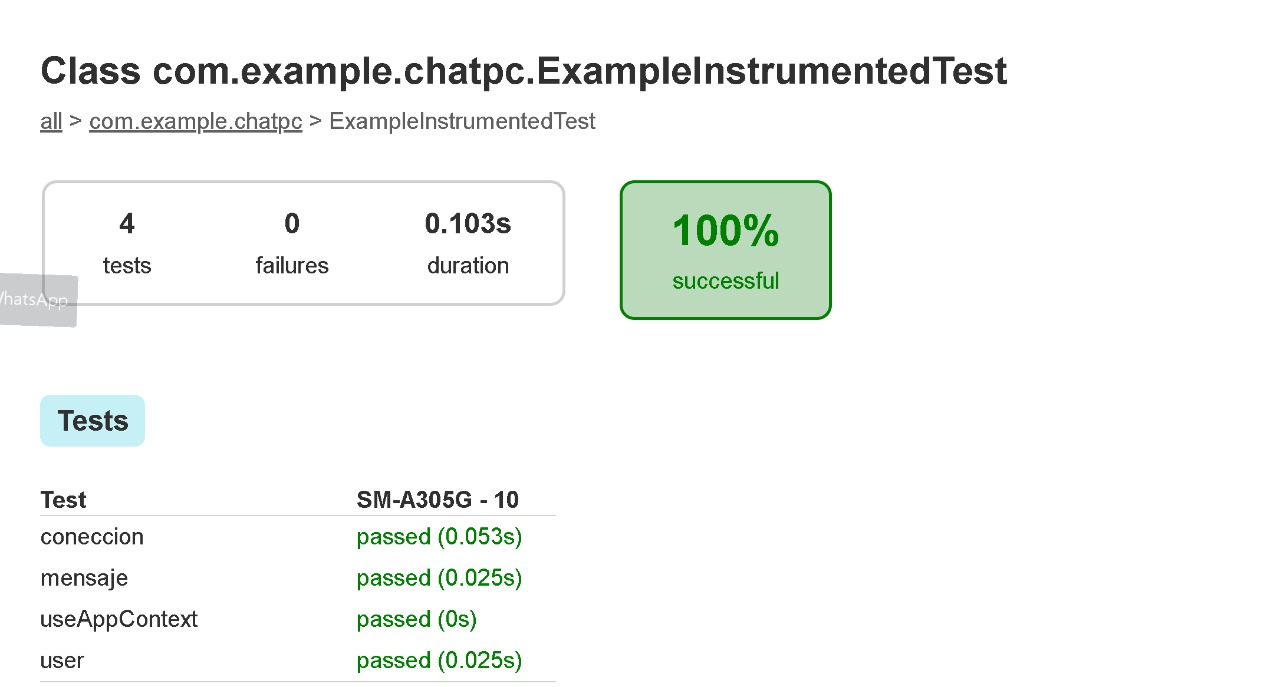
# **ANEXO N°2: Requisitos**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Enunciado de la Historia | | | | Criterios de aceptación | | | |
| (ID) de la historia | **ROL** | **Característica/Funcionalidad** | **Razón/Resultado** | **(#) de escenario** | **Criterio de aceptación Título** | **Contexto** | **Evento** | **Resultado/Comportamiento esperado** |
| R01 | Usuario | Necesito poder enviar un mensaje a otro usuario del chat | Con la finalidad de entablar una conversación con otro usuario | 1 | Envío de mensaje de texto | En caso de que un usuario envié un mensaje de solo texto | Cuando presione el botón enviar | Se mostrará en pantalla el mensaje enviado |
| 2 | Envío de una imagen | En caso de que un usuario decida enviar una foto o imagen | Cuando seleccione el botón de imagen, se abrirá la galería del celular | Se cargará la imagen para enviar como mensaje por el chat. |
|  | | | | | | | | |
| R02 | Usuario | Necesito poder visualizar la hora a la que fue enviado un mensaje en el chat | Con la finalidad de saber la secuencia correcta de los mensajes | 1 | Hora de mensajes | En caso de que un usuario envíe un mensaje | Cuando presione el botón enviar | Se mostrará en pantalla el mensaje enviado conjuntamente con la hora a la que fue enviado |
|  | | | | | | | | |
| R03 | Administrador | Necesito tener una pantalla de registro de usuario | Con la finalidad de identificar a la persona que se registra y usa la aplicación | 1 | Autenticación de usuario | En caso de que un usuario decida usar el chat | Cuando abra la aplicación por primera vez | Se mostrará una pantalla de registro para que ingrese su nombre, apellido y una fotografía de perfil. |
|  | | | | | | | | |
| R04 | Administrador | Necesito guardar la sesión del usuario | Con la finalidad de que el usuario no se registre cada vez que ingresa a la aplicación | 1 | Guardar sesión | En caso de que un usuario salga de la aplicación | Cuando decida volver a entrar al chat | Se mostrará inmediatamente la pantalla inicial del chat, sin mostrar la pantalla de registro. |
|  | | | | | | | | |
| R05 | Usuario | Necesito poder cerrar sesión | Con el fin de que otra persona no acceda a los chats | 1 | Cerrar sesión | En caso de que un usuario decida cerrar sesión | Cuando presione el botón de salir de la sesión actual | Se re-direccionará a la página de inicio de sesión |
|  | | | | | | | | |
| R06 | Usuario | Necesito poder visualizar el nombre y foto de perfil en miniatura de las personas en el chat | Con la finalidad de saber con cual persona se está comunicando | 1 | Entrar a la aplicación | En caso de que un usuario se registre en la aplicación | Cuando entre a la pantalla principal del chat | Se mostrará una lista de usuarios registrados con su nombre y foto en miniatura |
| 2 | Enviar un mensaje | En caso de que un usuario envíe un mensaje | Cuando presione el botón enviar | Se mostrará en pantalla el mensaje enviado conjuntamente con la hora, nombre y foto en miniatura de la persona que envió. |
|  | | | | | | | | |
| R07 | Usuario | Necesito poder visualizar conversaciones anteriores cada vez que ingreso a la aplicación | Con la finalidad de ver en pantalla los mensajes enviados anterior mente | 1 | Persistencia de mensajes | En caso de que un usuario salga de la aplicación | Cuando decida volver a entrar al chat | Podrá visualizar de manera inmediata los mensajes enviados anteriormente |

# **ANEXO N°3: Unit Test**

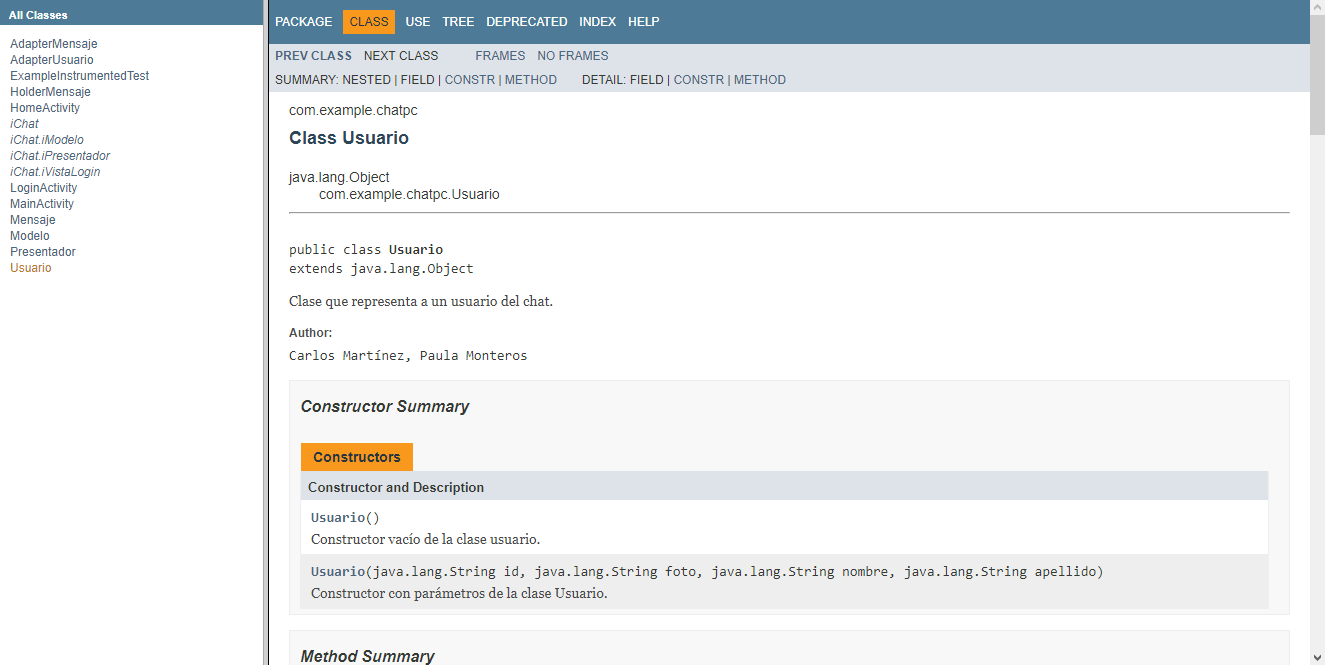
|  |
| --- |
| **ExampleInstrumentedTest.java** |
| /\*  \* ESPE - DCC - PROGRAMACIÓN MÓVIL  \* NRC: 6112  \*  \* Sistema: CHATP&C  \* Creado 02/08/2020  \*  \* Los contenidos de este archivo son propiedad privada y estan protegidos por  \* la licencia BSD  \*  \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  \*/ package com.example.chatpc;  import android.content.Context;  import androidx.test.platform.app.InstrumentationRegistry; import androidx.test.ext.junit.runners.AndroidJUnit4;  import org.junit.Test; import org.junit.runner.RunWith;  import static org.junit.Assert.\*;  */\*\*  \* Clase para las pruebas unitarias del programa desarrollado.  \*  \** ***@author*** *Carlos Martínez  \** ***@author*** *Paula Monteros  \*/* @RunWith(AndroidJUnit4.class) public class ExampleInstrumentedTest {   */\*\*  \* Prueba unitaria para el contexto de la aplicación.  \*/* @Test  public void useAppContext() {  // Context of the app under test.  Context appContext = InstrumentationRegistry.*getInstrumentation*().getTargetContext();  *assertEquals*("com.example.chatpc", appContext.getPackageName());  }   */\*\*  \* Prueba unitaria para la creación de usuarios.  \*/* @Test  public void user(){  Usuario usuario=new Usuario();  *assertNotNull*(usuario);  }   */\*\*  \* Prueba unitaria para la creación de mensajes.  \*/* @Test  public void mensaje(){  Mensaje usuario=new Mensaje();  *assertNotNull*(usuario);  }   */\*\*  \* Prueba unitaria para la conexión con la base.  \*/* @Test  public void coneccion(){  Modelo modelo=new Modelo();  *assertNotNull*(modelo);  } } |

****

****

# **ANEXO N°4: JavaDoc**





# **ANEXO N°5: Código**

