

PDF DE MÉTODOS DE LA APLICACIÓN DE NÉRO



PARTICIPANTES:

NICOLÁS FELIPE TRUJILLO MONTERO

MERLÍN DANIEL SIERING

ÁLVARO LOPERA MENDIZÁBAL

JUAN ANDRÉS PAREJA BALLESTEROS

RAFAEL RUEDA RODRÍGUEZ

FRANCISCO PÉREZ MORENO

ELIO JESÚS JIMÉNEZ LUQUE

Índice de directorios en la carpeta source "src" del proyecto:

A) Directorio Main

- 1.-Paquete "com.example.nero"
- 2.-Paquete "PaqueteClasesPrincipales"
- 3.-Paquete "PaqueteReporteFactoryMethod"
- 4.-Paquete "PaqueteRequest"
- 5.-Paquete "Ajustes"
- 6.-Paquete "PaqueteRequestQueueSingleton"

B) Directorio Jest

- 1.-Paquete "com.example.nero"

C) Directorios fuera de la aplicación

- 1.-Paquetes PHP

1.-Paquete "com.example.nero"

Este paquete va dirigido a controlar el flujo de la aplicación, usando los XML como interfaz activa de la aplicación.

A) Activity_Maps

La actividad mapa, es una clase java con un archivo "xml" donde actúa como interfaz gráfica de la clase en cuestión.

Métodos:

Declaramos las variables e instanciamos los elementos de la interfaz gráfica.

1.- protected void onCreate(Bundle savedInstanceState)

La función onCreate es una función propia de Android donde se inicializa y crea los datos mínimos para que la activity muestre el mapa.

A continuación, se inicializa el elemento de la interfaz gráfica y se instancia a la función para obtener los datos de la activity anterior, posteriormente, se configura un mensaje por defecto cada vez que entre por primera vez en la activity por cada logeo o inicio de la aplicación.

Consecutivamente se obtiene la geo-posición del usuario y se genera una lista de fuentes alrededor suyo.

Si se pulsa en el botón de añadir fuente se genera una nueva activity la cual se explica más adelante.

Directorio Main

```
activity_maps.java
43 public class activity_maps extends FragmentActivity implements OnMapReadyCallback, OnMyLocationButtonClickListener, GoogleMap.OnMarkerClickListener {
44
45     //Mapa y localización
46     private GoogleMap mMap;
47     private FusedLocationProviderClient fusedLocationClient;
48
49     //Constantes y códigos
50     private static final float DEFAULT_ZOOM = 15f;
51     private final static int CODIGO_DE_PERMISO_LOCALIZACION = 123;
52
53     //Elementos de la pantalla
54     private Button btn_anyadirFuente;
55
56     //Datos
57     private Usuario thisUser;
58     private Double latitudThisUser;
59     private Double longitudThisUser;
60     private List<Fuente> fuentes;
61
62     //-----
63     @Override
64     protected void onCreate(Bundle savedInstanceState) {
65         //Default
66         super.onCreate(savedInstanceState);
67         setContentView(R.layout.activity_maps);
68         SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
69             .findFragmentById(R.id.map);
70         mapFragment.getMapAsync(this);
71     }
72 }
```

```
72 //-----R
73 btn_anyadirFuente = (Button) findViewById(R.id.btn_anyadirfuente);
74
75 //-----Obtencion de datos de la Activity anterior
76 obtenerDatosUsuario();
77 Toast.makeText(context, this, text: "Hola " + thisUser.getNombre(), Toast.LENGTH_LONG).show();
78
79 //-----Inicialización de localización y de la lista de fuentes
80 fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
81 fuentes = new ArrayList<>();
82
83 //-----Eventos
84 btn_anyadirFuente.setOnClickListener((v) -> {
85     Intent intent = new Intent(getApplicationContext(), AnyadirFuente.class);
86     intent.putExtra(name: "PasarEsteUsuario", thisUser);
87     intent.putExtra(name: "latitud", latitudThisUser);
88     intent.putExtra(name: "longitud", longitudThisUser);
89     startActivity(intent);
90 });
91
92 }
```

2.- public void onMapReady(GoogleMap googleMap)

La función onMapReady configura los permisos de acceso al servicio de GPS del dispositivo del usuario, donde si este los deniega no podrá acceder o más bien la actividad no mostraría nada y se le volvería a preguntar si da el acceso a estos

Directorio Main

servicios. En caso de que acceda se configura por defecto y se cargan los datos.

```
189 @Override
190 public void onMapReady(GoogleMap googleMap) {
191     //Default
192     mMap = googleMap;
193
194     //-----PERMISOS DE LOCALIZACION-----
195     if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)
196         == PackageManager.PERMISSION_GRANTED) {
197         //Si ya tenemos los permisos, agregamos las funciones GPS
198         agregarFuncionesGPS();
199     } else {
200         //Si no tenemos los permisos, preguntaremos por los permisos
201         //Nota: Una vez el usuario seleccione ACEPTAR o DENEGAR se llama a onRequestPermissionsResult
202         //automaticamente (ver abajo)
203         if (ActivityCompat.shouldShowRequestPermissionRationale( activity: this, Manifest.permission.ACCESS_FINE_LOCATION)) {
204             //Lo dejamos por Default
205         } else {
206             //Pedimos los permisos
207             ActivityCompat.requestPermissions( activity: this,
208                 new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
209                 CODIGO_DE_PERMISO_LOCALIZACION);
210         }
211     }
212 }
213
214
215
```

3.- public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)

La función onRequestPermissionsResult() pide los permisos del GPS al dispositivo del usuario.

```
136 //-----OBTENCIÓN Y COLOCACIÓN DE LAS FUENTES-----
137 obtenerFuentes();
138 //-----EVENTOS-----
139 googleMap.setOnMarkerClickListener(this);
140 }
141
142
143 //-----LOCALIZACIÓN Y PERMISOS-----
144 //Permisos de aplicación (En este caso, solo hay de localización)
145 @Override
146 public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
147     switch (requestCode) {
148         case CODIGO_DE_PERMISO_LOCALIZACION: {
149             if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
150                 //Si los permisos han sido dados, obtenemos la posición GPS
151                 agregarFuncionesGPS();
152             } else {
153                 Toast.makeText( context: this, text: "Error: Permisos denegados.", Toast.LENGTH_LONG).show();
154             }
155             return;
156         }
157     }
158 }
159
160
```

4.- private void agregarFuncionesGPS()

La función agregarFuncionesGPS() activa el GPS en el mapa y obtiene la ubicación.

Directorio Main

5.- public boolean onMyButtonClick()

La función onMyButtonClick() sirve para retornar falso porque no os hace falta después.

6.- private boolean onMarkerClick(Marker marker)

La función onMarkerClick(Marker marker) activa el GPS en el mapa y obtiene la ubicación.

7.- private void obtenerDatosUsuario()

La función obtenerDatosUsuario() obtiene los datos de usuario de la actividad anterior

8.- private void obtenerFuentes()

La función obtenerFuentes() obtiene los datos, a través de la consulta con la base de datos, para su posterior visionado en pantalla a través del mapa.

*A través de la clase JSon realizaremos consultas a la base de datos, en caso de que ocurra algún error se mostrara un mensaje por pantalla indicándolo, en caso de que todo funcione correctamente, una vez se cierre la activity se añadirá a la cola para su posterior uso.

Directorio Main

```
34 public class AnyadirFuente extends FragmentActivity implements OnMapReadyCallback, OnMarkerDragListener, TimePickerDialog.OnTimeSetListener {
35
36     //Constantes
37     private static final int MAXPROX = 100000;
38     private static final float DEFAULT_ZOOM = 15;
39
40     //Elementos de la pantalla
41     private Button boton_AF;
42     private Switch disponibilidad;
43     private Switch condiciones;
44     private GoogleMap mapa;
45     private EditText nombreFuente;
46     private Button btn_hora_inicio;
47     private TextView apertura;
48     private TextView cierre;
49     private Button botonCierre;
50     private Button botonApertura;
51     private boolean timePicker1;
52
53     //Datos
54     private Double latitud; // fuente a crear
55     private Double longitud; // fuente a crear
56     private Double latUser;
57     private Double lonUser;
58     private Marker fuente;
59     private Usuario thisUser;
60     private String nomUsuario;
```

B) Anyadir_Reporte_Usuario

La clase añadir reporte usuario, es la clase que se encarga de realizar el reporte de un usuario, mandando la información a la base de datos y en caso de que ese usuario llegue a una cantidad de tickets, su cuenta se baneara o inhabilitará de manera automática.

Las variables que se usaran son:

```
//Constantes
private static final int MY_SOCKET_TIMEOUT_MS = 6000;
private String[] items;
//Elementos
private TextInputLayout tipo;
private TextInputLayout descripcion;
private AutoCompleteTextView autocomplete;
private MaterialButton btn_anyadirReporte;

//Datos
private Usuario thisUser;
private Usuario reportado;
```

Donde se encuentran instanciados el tiempo máximo para que la consulta del reporte se realice con éxito, un array de String donde se almacenara las opciones para el reporte. También instanciamos los elementos de la interfaz ligada a esta activitys y dos usuarios correspondiendo al usuario que reporta y el que es reportado.

Métodos:

1.- protected void onCreate(Bundle savedInstanceState)

Directorio Main

La función onCreate() inicializa los elementos de la interfaz y adjunta las opciones a través del array ítems.

2.- private void usarVolley(final String iddescripcion)

La función usarVolley() realiza la parte de consulta a la base de datos, mientras no se haya realizado a este usuario un reporte anteriormente, se haya seleccionado una opción y se rellene el recuadro de informe, o bien, no ocurra ningún error en la conexión o se intente forzar al servidor, se realizará la consulta correctamente y se enviara un correo autogenerado al correo linkeado a la cuenta con información básica.

3. private void obtenerDatos()

La función obtenerDatos() realiza consulta a la base de datos.

C) AnyadirComentario

La clase añadir comentario sirve para como indica su nombre crear la actividad que se realiza a través de pulsar en la información de un objeto del mapa, donde previamente podremos visualizar otros comentarios entre otras opciones.

Para esta clase se usarán las siguientes variables:

```
//Elementos
private TextInputLayout comentario;
private FloatingActionButton btn_aceptar;

//Datos
private Usuario thisUser;
private Fuente thisFuente;
```

Donde se instancian dos objetos, fuente y usuario y los elementos de la interfaz.

Métodos:

```
protected void onCreate(Bundle savedInstanceState) {
```

1.- protected void onCreate(Bundle savedInstanceState)

Directorio Main

La función onCreate inicializa los objetos a través de la función “Obtener datos” a su vez también se inicializan los elementos de la interfaz y se configura qué salte un error cuando se dé al botón salte un error cuando el texto del comentario este vacío.

```
private void crearNuevoComentario(String txt){  
  
    Response.Listener<String> respuesta = (response) -> {  
        try{  
            JSONObject j = new JSONObject(response);  
            if(j.getBoolean( "name: success")){  
                Toast.makeText( context: AnyadirComentario.this, text: "Se ha realizado el comentario correctamente", Toast.LENGTH_SHORT).show();  
                finish();  
            } else {  
                Toast.makeText( context: AnyadirComentario.this, text: "Ha ocurrido un error. Prueba otra vez", Toast.LENGTH_SHORT).show();  
            }  
        } catch (JSONException ex){  
            System.out.println("Error: "+ex.getMessage());  
        }  
    };  
  
    AnyadirComentarioRequest ac = new AnyadirComentarioRequest(thisUser.getNombre_usuario(), thisUser.getPassword(), txt, idfu);  
    SingletonRequestQueue.getInstance(AnyadirComentario.this).addToRequestQueue(ac);  
}
```

2.- private void crearNuevoComentario(String txt)

La función crear Nuevo comentario realiza una consulta a la base de datos y liga el comentario al usuario que lo ha redactado y también a la fuente de la que se realiza el comentario.

3.- private void obtenerDatos()

Pasa los datos de una actividad a otra.

D) AnyadirFuente

En esta clase primero tenemos la declaración de las variables y la unión del xml con el java de esta clase.

```
34 public class AnyadirFuente extends FragmentActivity implements OnMapReadyCallback, OnMarkerDragListener, TimePickerDialog.OnTimeSetListener {
35
36     //Constantes
37     private static final int MAXPROX = 100000;
38     private static final float DEFAULT_ZOOM = 15;
39
40     //Elementos de la pantalla
41     private Button boton_AF;
42     private Switch disponibilidad;
43     private Switch condiciones;
44     private GoogleMap mapa;
45     private EditText nombreFuente;
46     private Button btn_hora_inicio;
47     private TextView apertura;
48     private TextView cierre;
49     private Button botoncierre;
50     private Button botonapertura;
51     private boolean timePicker1;
52
53     //Datos
54     private Double latitud; // fuente a crear
55     private Double longitud; // fuente a crear
56     private Double latUser;
57     private Double lonUser;
58     private Marker fuente;
59     private Usuario thisUser;
60     private String nomUsuario;
```

Métodos:

1.- protected void onCreate(Bundle savedInstanceState)

Este método funciona como un main donde se recogen variables, valores, etc.

```
62 @Override
63 protected void onCreate(Bundle savedInstanceState) {
64     //Default
65     obtenerDatosUsuario();
66     super.onCreate(savedInstanceState);
67     setContentView(R.layout.activity_anyadir_fuente);
68     SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
69         .findFragmentById(R.id.map);
70     mapFragment.getMapAsync(new OnMapReadyCallback() {
71         //final Button button = (Button) findViewById(R.id.button);
72         /*button.setOnClickListener(new View.OnClickListener() {
73             @Override
74             public void onClick(View v) {
75                 DialogFragment timePicker = new TimePicker();
76                 timePicker.show(getSupportFragmentManager(), "time picker");
77             }
78         });
79         */
80         //-----R
81         boton_AF = (Button) findViewById(R.id.button2);
82         nombreFuente = (EditText) findViewById(R.id.editText);
83         disponibilidad = (Switch) findViewById(R.id.switch3);
84         condiciones = (Switch) findViewById(R.id.switch2);
85         botoncierre = (Button) findViewById(R.id.button3);
86         botoncierre.setOnClickListener((v) -> {
87             timePicker1 = false;
88             DialogFragment timePicker2 = new TimePicker();
89             timePicker2.show(getSupportFragmentManager(), "time picker");
```

Directorio Main

```
93     });
94     apertura = (TextView) findViewById(R.id.textView2);
95     cierre = (TextView) findViewById(R.id.textView3);
96     buttonApertura = (Button) findViewById(R.id.button);
97     buttonApertura.setOnClickListener((v) -> {
98         timePicker1 = true;
99         DialogFragment timePicker = new TimePicker();
100         timePicker.show(getSupportFragmentManager(), tag: "time picker");
101     });
102
103     //-----Eventos
104     disponibilidad.setOnCheckedChangeListener((buttonView, isChecked) -> {
105         if(isChecked){
106             Toast.makeText( context: AnyadirFuente.this, text: "Siempre", Toast.LENGTH_SHORT).show();
107             apertura.setVisibility(View.INVISIBLE);
108             cierre.setVisibility(View.INVISIBLE);
109             buttonApertura.setClickable(false);
110             botoncierre.setClickable(false);
111         } else {
112             Toast.makeText( context: AnyadirFuente.this, text: "Seleccione horario", Toast.LENGTH_SHORT).show();
113             apertura.setVisibility(View.VISIBLE);
114             cierre.setVisibility(View.VISIBLE);
115             buttonApertura.setClickable(true);
116             botoncierre.setClickable(true);
117         }
118     });
119
120     });
121
122     });
123
124     });
125
126     });
```

2.- public void onClick(View v)

Este método inicia el timepicker, que usamos para escoger el horario en el que la fuente está disponible.

```
126     boton_AF.setOnClickListener((v) -> {
127         String dispo;
128         if (disponibilidad.isChecked()){
129             dispo = "Siempre";
130         }else{
131             dispo = "Disponible de: "+apertura.getText().toString()+" a "+cierre.getText().toString(); // Disponibilidad de 00:00 a 17:00.
132         }
133         if(condiciones.isChecked()){
134             usarVolley(dispo);
135         }else{
136             Toast.makeText( context: AnyadirFuente.this, text: "Error: Debe aceptar las condiciones", Toast.LENGTH_SHORT).show();
137         }
138     });
139
140     // Se especifica el texto a mostrar cuando se selecciona hora en el TimePicker
141
142     @Override
143     public void onTimeSet(android.widget.TimePicker view, int hourOfDay, int minute) {
144         if(timePicker1){
145             TextView textView = (TextView) findViewById(R.id.textView2);
146             if (minute <= 9) {
147                 textView.setText(hourOfDay + ":0" + minute);
148             } else {
149                 textView.setText(hourOfDay + ":" + minute);
150             }
151         }
152     }
153
154     }
```

3.- public void onMapReady(GoogleMap googleMap)

Este método sirve para poner el mapa y capturar la ubicación de la fuente.

Directorio Main

```
158         }else{
159             TextView textView = (TextView) findViewById(R.id.textView3);
160             if (minute <= 9) {
161                 textView.setText(hourOfDay + ":0" + minute);
162             } else {
163                 textView.setText(hourOfDay + ":" + minute);
164             }
165         }
166     }
167 }
168
169 //-----MAPA
170 @Override
171 public void onMapReady(GoogleMap googleMap) {
172     //Default
173     mapa = googleMap;
174
175     //-----LOCALIZACIÓN DEL USUARIO
176     LatLng indiv = new LatLng(latUser, lonUser);
177
178     //-----MARKER FUENTE
179     fuente = mapa.addMarker(new MarkerOptions().position(indiv).draggable(true));
180     googleMap.setOnMarkerDragListener(this);
181
182     //-----CONFIGURACIONES DEL MAPA
183     CircleOptions circulo = new CircleOptions();
184     circulo.center(indiv);
185     circulo.radius(MAXPROX);
186     circulo.visible(true);
```

4.- private void usarVolley(String dispo)

Este método sirve para pasar los datos al php, desde éste se hacen las consultas con un JSON.

```
187     mapa.addCircle(circulo);
188
189     mapa.getUiSettings().setMyLocationButtonEnabled(false);
190     mapa.setMyLocationEnabled(true); //Se presupone que los permisos ya han sido aceptados
191     mapa.setMapType(2); //Tipo de mapa a satélite
192     mapa.setMinZoomPreference(15);
193     mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(indiv, DEFAULT_ZOOM));
194 }
195
196 //-----VOLLEY(Java -> PHP -> BD) -> (BD -> PHP -> JAVA)
197 private void usarVolley(String dispo){
198     latitud = fuente.getPosition().latitude;
199     longitud = fuente.getPosition().longitude;
200     nomUsuario = thisUser.getNombre_usuario();
201
202     if(checkProximity(latitud, longitud, latUser, lonUser)){
203         Response.Listener<String> respuesta = (respuesta) -> {
204             try{
205                 JSONObject jsonRespuesta = new JSONObject(respuesta);
206                 if(jsonRespuesta.getBoolean("success")){
207                     Toast.makeText(context, AnyadirFuente.this, text: "Insertado correctamente.", Toast.LENGTH_SHORT).show();
208                     finish();
209                 }else{
210                     Toast.makeText(context, AnyadirFuente.this, text: "Error al insertar la fuente.", Toast.LENGTH_SHORT).show();
211                 }
212             } catch (JSONException e){
213                 e.getMessage();
214             }
215         };
216     }
217 }
```

Directorio Main

```
220 String lati = Double.toString(latitud);
221 String lon = Double.toString(longitud);
222
223 AnyadirFuenteRequest afr = new AnyadirFuenteRequest(nomUsuario, lati, lon, dispo, nombreFuente.getText().toString(), respuesta);
224 SingletonRequestQueue.getInstance(AnyadirFuente.this).addToRequestQueue(afr);
225 }else{
226     Toast.makeText(context, AnyadirFuente.this, "Error: Fuera de rango", Toast.LENGTH_SHORT).show();
227 }
228 }
229 }
230 }
231
232
233 //-----CALCULO DE DISTANCIAS ENTRE 2 PUNTOS
234 @ private Boolean checkProximity(double latitud, double longitud, double latUser, double lonUser){
235     int dist = FormulaHaversine(latitud, longitud, latUser, lonUser);
236     Boolean res = dist < MAXPROX;
237     return res;
238 }
239
240 private static int FormulaHaversine(double lon1, double lat1,
241                                     double lon2, double lat2) {
242
243     double earthRadius = 6371; // km
244
245     lati = Math.toRadians(lati);
246     lon1 = Math.toRadians(lon1);
247     lat2 = Math.toRadians(lat2);
```

5.- private Boolean checkProximity(double latitud, double longitud, double latUser, double lonUser)

Este método comprueba que el usuario no se encuentre lejos de la fuente a añadir.

6.- private static int FormulaHaversine(double lon1, double lat1, double lon2, double lat2)

Este método calcula la distancia entre dos puntos de la tierra, fórmula necesaria al ser la tierra redonda y achatada por los polos. Esta fórmula es utilizada por checkProximity.

7.- private void obtenerDatosUsuario()

Este método obtiene los datos del usuario, para poder identificar al creador de la fuente y su ubicación actual.

```
248     lon2 = Math.toRadians(lon2);
249
250     double dlon = (lon2 - lon1);
251     double dlat = (lat2 - lat1);
252
253     double sinlat = Math.sin(dlat / 2);
254     double sinlon = Math.sin(dlon / 2);
255
256     double a = (sinlat * sinlat) + Math.cos(lat1)*Math.cos(lat2)*(sinlon*sinlon);
257     double c = 2 * Math.asin(Math.min(1.0, Math.sqrt(a)));
258
259     double distanceInMeters = earthRadius * c * 1000;
260
261     return (int) distanceInMeters;
262 }
263
264
265 //-----OBTENCIÓN DE DATOS DE LA ACTIVITY ANTERIOR
266 private void obtenerDatosUsuario(){
267     Bundle extras = getIntent().getExtras();
268     thisUser = (Usuario) extras.getSerializable( key: "PasarEsteUsuario");
269     latUser = (Double) extras.getDouble( key: "latitud");
270     lonUser = (Double) extras.getDouble( key: "longitud");
271 }
272
273
274 //-----EVENTOS DEL MARKER (fuente)
275 @Override
276 public void onMarkerDragStart(Marker marker) {
```

Directorio Main

8.- public void onMarkerDrag(Marker marker)

Este método crea el icono que sirve para señalar la ubicación de la fuente.

9.- public void onMarkerDragStart(Marker marker)

Este método inicializa el markerDrag.

10.- public void onMarkerDragEnd(Marker marker)

Este método sitúa el markerDrag donde se encuentra el usuario.

```
277         //Default
278     }
279
280     @Override
281     public void onMarkerDrag(Marker marker) {
282         //Default
283     }
284
285     @Override
286     public void onMarkerDragEnd(Marker marker) {
287         fuente.setPosition(new LatLng(marker.getPosition().latitude, marker.getPosition().longitude));
288     }
289
290 }
291
```

E) AnyadirReporteComentario

La clase contiene una serie de métodos que permite añadir a la base de datos un reporte de un comentario, el funcionamiento de la clase es casi idéntico al de reporte usuario y fuente.

Las variables que usaremos son:

```
//Constantes
private static final int MY_SOCKET_TIMEOUT_MS = 6000;
private String[] items;
//Elementos
private TextInputLayout tipo;
private TextInputLayout descripcion;
private AutoCompleteTextView autocomplete;
private MaterialButton btn_anyadirReporte;
```

Donde tendremos un ticket de tiempo en el cual se finalizará la conexión con el servidor para realizar el reporte, el array items contendrá la información de las pociones predeterminadas por las que se puede reportar a un usuario. El resto de los elementos pertenecen a su interfaz de diseño asociada.

1.- protected void onCreate(Bundle savedInstanceState)

Directorio Main

La función onCreate() inicializan los elementos de la interfaz y se les adjunta las opciones a través del array ítems. También genera un mensaje por pantalla si se intenta enviar un reporte sin haber rellenado los campos necesarios para el envío.

2.- private void usarVolley(String idescription)

La function usarVolley(String idescription) realiza la parte de consulta. Consulta a la base de datos, mientras no se haya realizado a este usuario un reporte anteriormente, sé haya seleccionado una opción y se rellene el recuadro de informe, o bien, no ocurra ningún error en la conexión o se intente forzar al servidor, se realizará la consulta correctamente y se enviara un correo autogenerado al correo linkeado a la cuenta con información básica.

3.- private void obtenerDatos()

Pasa los datos de una actividad a otra.

F) AnyadirReporteFuente

La clase contiene una serie de métodos que permite añadir a la base de datos un reporte de una fuente, el funcionamiento de la clase es casi idéntico al de reporte usuario y comentario.

Las variables que usaremos son:

```
//Constantes
private String[] items;

//Elementos
private TextInputLayout tipo;
private TextInputLayout descripcion;
private AutoCompleteTextView autocomplete;
private MaterialButton btn_anyadirReporte;
```

Donde tendremos un ticket de tiempo en el cual se finalizará la conexión con el servidor para realizar el reporte, el array ítems contendrá la información de las pociones predeterminadas por las que se puede reportar a una fuente. El resto de los elementos pertenecen a su interfaz de diseño asociada.

Métodos:

1.- protected void onCreate(Bundle savedInstanceState)

La función onCreate() inicializan los elementos de la interfaz y se les adjunta las opciones a través del array ítems. También genera un mensaje por pantalla si se intenta enviar un reporte sin haber rellenado los campos necesarios para el envío.

Directorio Main

2.- private void usarVolley(String idescription)

La función usarVolley(String idescription) realiza la parte de consulta. Consulta a la base de datos, mientras no se haya realizado a este usuario un reporte anteriormente, sé haya seleccionado una opción y se rellene el recuadro de informe, o bien, no ocurra ningún error en la conexión o se intente forzar al servidor, se realizará la consulta correctamente y se enviara un correo autogenerado al correo linkeado a la cuenta con información básica.

3.- private void obtenerDatos()

Pasa los datos de una actividad a otra.

G) ComentariosFuentes

Esta clase controlará todos los eventos del fragment que nos muestra todos aquellos comentarios que tiene una fuente en concreto.

Métodos:

1.- public View onCreateView (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)

En este método se encargará de conectarnos con la vista y actualizarla. También se encargará de inicializar el RecyclerView.

2.- public void obtenerComentarios()

Este método se encargará de realizar la petición a partir del ID de al fuente para obtener todos los comentarios.

H) DetallesFuente

Esta clase se encargará de crear las instancias de nuestro menú. Nuestro menú solo constará de 2 secciones: una sección para mostrarnos la información de la fuente y otra sección para mostrarnos los comentarios de dicha fuente.

Directorio Main

```
DetallesFuente.java
1 package com.example.nero;
2
3 import ...
4
20
21 public class DetallesFuente extends AppCompatActivity {
22
23     //Elementos de la pantalla
24     private ViewPager paginas;
25     private TabLayout barra;
26     //Fragmentos
27     private InformacionFuente informacionFuente;
28     private ComentariosFuente comentariosFuente;
29     //Datos
30     private Fuente thisFuente;
31     private Usuario thisUser;
32
33     @Override
34     protected void onCreate(Bundle savedInstanceState) {
35         //Default
36         super.onCreate(savedInstanceState);
37         setContentView(R.layout.activity_detalle_fuente);
38
39         //R
40         paginas = (ViewPager) findViewById(R.id.pagino);
41         barra = (TabLayout) findViewById(R.id.tabs);
42
43         //Obtención de datos de la activity anterior
44         obtenerDatos();
45     }
}
```

Métodos:

1.- protected void onCreate (Bundle savedInstanceState)

Este método se encargará de la llamada de los constructores de las instancias anteriormente mencionadas. También se definirá los eventos necesarios para nuestro menú.

```
46 //Fragments y su configuración
47 informacionFuente = new InformacionFuente(thisFuente, thisUser);
48 comentariosFuente = new ComentariosFuente(thisFuente, thisUser);
49
50 barra.setupWithViewPager(paginas);
51 DetallesFuente.ViewPagerAdapter vpa = new ViewPagerAdapter(getSupportFragmentManager(), behavior: 0);
52 vpa.addFragment(informacionFuente, nombre: "Información");
53 vpa.addFragment(comentariosFuente, nombre: "Comentarios");
54 paginas.setAdapter(vpa);
55
56
57 }
58 //Fragments
59 private class ViewPagerAdapter extends FragmentPagerAdapter {
60
61     private List<Fragment> fragments = new ArrayList<>();
62     private List<String> fragmentNombre = new ArrayList<>();
63
64     public ViewPagerAdapter(@NonNull FragmentManager fm, int behavior) {
65         super(fm, behavior);
66     }
67
68
69     public void addFragment(Fragment fragment, String nombre){
70         fragments.add(fragment);
71         fragmentNombre.add(nombre);
72     }
73
74 }
```

Directorio Main

2.- private class ViewPagerAdapter

Este método se encargará de la inicialización de ambos fragmentos (ComentariosFuentes e InformacionFuente).

3.- void obtenerDatos()

Obtendremos los datos de la Activity anterior.

```
75     @NonNull
76     @Override
77     public Fragment getItem(int position) { return fragments.get(position); }
80
81     @Override
82     public int getCount() { return fragments.size(); }
85
86     @Nullable
87     @Override
88     public CharSequence getPageTitle(int position) { return fragmentNombre.get(position); }
91 }
92
93 //-----OBTENCIÓN DE LOS DATOS DE LA FUENTE DE LA ACTIVITY ANTERIOR-----
94 void obtenerDatos(){
95     Bundle extras = getIntent().getExtras();
96     thisFuente = (Fuente) extras.getSerializable( key: "PasarEstaFuente");
97     thisUser = (Usuario) extras.getSerializable( key: "PasarEsteUsuario");
98 }
99
100 }
```

1) InformacionFuente

Esta clase controlará todos los eventos del fragment que nos muestra la información de una fuente. Guardaremos como atributo los datos de la fuente.

Métodos:

1.- protected void onCreateView (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)

En este método se encargará de conectarnos con la vista y actualizarla. También tendremos el evento para poder realizar un reporte de fuente.

J) MainActivity

Esta clase controlará todos los eventos de la Activity inicial que, en nuestro caso, es el inicio de sesión.

```
MainActivity.java x
1 package com.example.nero;
2
3 import ...
22
23 public class MainActivity extends AppCompatActivity {
24     //Constantes
25     private static final int MY_SOCKET_TIMEOUT_MS = 6000;
26
27     //Campos
28     private Button boton_IC;
29     private EditText camp_usuario;
30     private EditText camp_contra;
31     private ProgressBar circulo_Carga;
32
33     //Datos
34     private String usu;
35     private String cont;
36
37     private Button boton_sign_in;
38     @Override
39     protected void onCreate(Bundle savedInstanceState) {
40         //Default
41         super.onCreate(savedInstanceState);
42         setContentView(R.layout.activity_main);
43
44         //R
45         boton_IC = (Button) findViewById(R.id.boton_is);
46         camp_usuario = (EditText) findViewById(R.id.campoUsuario);
47         camp_contra = (EditText) findViewById(R.id.campoContra);
```

Métodos:

1.- protected void onCreate (Bundle savedInstanceState)

En este método tenemos todos los eventos del inicio sesión.

2.- private void usarVolley()

Este método se encargará de comprobar los datos introducidos por el usuario. En el caso de que los datos son correctos, se procederá a mostrarnos el mapa con todos los datos de nuestro usuario ya cargados. En otro caso, simplemente nos mostrará un mensaje de error.

Directorio Main

```
48     circulo_Carga = (ProgressBar) findViewById(R.id.barraDeCarga);
49     circulo_Carga.setVisibility(View.INVISIBLE);
50     boton_sign_in = (Button) findViewById(R.id.btn_sign_in);
51
52     //Eventos
53     boton_IC.setOnClickListener((v) -> {
54         circulo_Carga.setVisibility(View.VISIBLE);
55         usarVolley();
56     });
57
58
59
60
61
62     boton_sign_in.setOnClickListener((v) -> {
63         Intent intent = new Intent(getApplicationContext(), sign_in.class);
64         startActivity(intent);
65     });
66
67 }
68
69
70
71
72 private void usarVolley(){
73     usu = camp_usuario.getText().toString();
74     cont = camp_contra.getText().toString();
75
76     Response.Listener<String> respuesta = (respuesta) -> {
77         circulo_Carga.setVisibility(View.INVISIBLE);
78         try{
79             JSONObject jsonRespuesta = new JSONObject(respuesta);
80             boolean ok = jsonRespuesta.getBoolean( name: "success");
81             if(ok){
82                 boolean baneado = jsonRespuesta.getBoolean( name: "baneado");
83                 if(baneado){
84                     AlertDialog.Builder alerta = new AlertDialog.Builder( context: MainActivity.this);
85                     alerta.setMessage("Estas baneado").setNegativeButton( text: "Volver", listener: null).create().show();
86                 } else {
87                     //AQUI DEBERIAMOS OBTENER TODOS LOS DATOS DEL USUARIO
88                     Usuario thisUser = new Usuario(usu, cont);
89                     thisUser.setNombre(jsonRespuesta.getString( name: "nombre"));
90                     Intent intent = new Intent(getApplicationContext(), activity_maps.class);
91                     intent.putExtra( name: "PasarEsteUsuario", thisUser);
92                     startActivity(intent);
93                 }
94             } else {
95                 AlertDialog.Builder alerta = new AlertDialog.Builder( context: MainActivity.this);
96                 alerta.setMessage("Datos incorrectos").setNegativeButton( text: "Volver", listener: null).create().show();
97             }
98         } catch (JSONException e){
99             e.getMessage();
100         }
101     };
102
103
104
105
106
107     LoginRequest lr = new LoginRequest(usu, cont, respuesta);
108     lr.setRetryPolicy(new DefaultRetryPolicy(
109         MY_SOCKET_TIMEOUT_MS,
110         DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
111         DefaultRetryPolicy.DEFAULT_BACKOFF_MULT));
112     SingletonRequestQueue.getInstance(MainActivity.this).addToRequestQueue(lr);
113 }
114
115 }
```

K) RVAdaptorComentarios

Esta clase se encargará de mostrarnos y actualizarnos cada elemento del RecyclerView. Cada elemento, es decir, cada comentario tendrá un botón a su derecha que se utilizará para reportar el comentario. También nos mostrará información, por ejemplo, el Usuario que escribió el comentario, la fecha en el que lo escribió, el texto del comentario y la foto de perfil de dicho usuario.

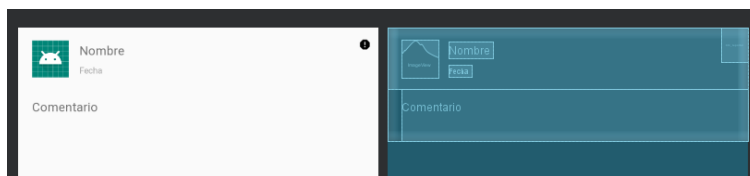
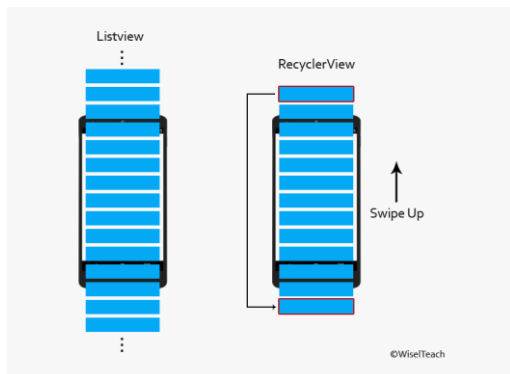
También, si se ha procedido a colocar un evento en el nombre del usuario: si pinchamos en el nombre del usuario podremos realizar un reporte a dicho usuario.

*Esta clase también comprobará si un comentario ha sido escrito por el mismo usuario que ha iniciado sesión. Esto se ha implementado para evitar reportarnos a nosotros mismos.

A nivel de implementación tendremos una lista de comentarios, que contendrá todos los comentarios que mostraremos. También tendremos un objeto de tipo Usuario para conocer el usuario que ha iniciado sesión anteriormente.

Aclaración

A la hora de mostrarnos los comentarios nosotros no solamente buscamos una forma cómoda de hacerlo, sino que también buscamos la máxima eficiencia y flexibilidad posible. Por ello se ha procedido a implementar un **RecyclerView** para poder ver cada comentario. El RecyclerView es mucho más eficiente y flexible que un ListView ya que este reciclará cada elemento de la lista:



Métodos:

1.- public void onBindViewHolder(@NonNull ViewHolder holder, int position)

Directorio Main

Será llamado por RecyclerView automáticamente para colocar un item (un comentario, la imagen anterior) a partir de una posición dada. También podemos ver, en el caso de que el Usuario del comentario que queremos colocar coincide con el Usuario que inició sesión, colocaremos el botón de reporte invisible (Esto quiere decir que, a parte de no verse, no funcionará)

```
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    holder.nombre_usuario.setText(listaComentario.get(position).getCreador().getNombre_usuario());
    holder.fecha.setText(listaComentario.get(position).getFecha());
    //holder.foto.setImageResource(listaComentario.get(position));
    holder.comentario.setText(listaComentario.get(position).getTexto());
    //Para evitar reportarnos a nosotros mismos
    if(listaComentario.get(position).getCreador().getNombre_usuario().equalsIgnoreCase(thisUser.getNombre_usuario())){
        holder.btn_reportar.setVisibility(View.INVISIBLE);
    }
    holder.thisComentario = listaComentario.get(position);
}
```

2.- public static class ViewHolder extends RecyclerView.ViewHolder

Aquí tendremos “la conexión” con la vista y la definición de cada evento a partir de los elementos de la vista.

L) Sign_in

Esta clase será el controlador para el sistema de registro.

Métodos:

1.- protected void onCreate(Bundle savedInstanceState)

Este método “se conectará” con la vista y tendrá todos los eventos necesarios para un sistema de registro robusto:

- Si el usuario deja algún campo vacío (de los que son obligatorios) mostrará un error actualizando la vista.
- Si el usuario excede la longitud o coloca espacios en el nombre de usuario mostrará un error actualizando la vista.

2.- private void usarVolley(String inombre, String inick, String ipassword, String iemail)

Este método se encargará de utilizar la biblioteca de Volley (biblioteca http, que en nuestro caso nos permite enviar o recibir información mediante POST). Una vez comprobado que los puntos anteriores del método OnCreate, enviará una solicitud a la nuestro PHP para comprobar si el nombre de usuario o el email se encuentra ya registrados en la base de datos y una vez el PHP tenga la respuesta nos enviará la información en formato JSON a nuestra aplicación java.

1. Si ya se encuentra en la base de datos, el php nos informará y nuestra aplicación actualizará la vista mostrando que ya existe el nombre de usuario o el email.

Directorio Main

2. Si no se encuentra en la base de datos, se registrará y nos informará nuestra aplicación de un registro correcto.

Nota: Para enviar la información nos ayudamos de la clase sign_inRequest

M) TimePicker

Esta clase será utilizará por AnyadirFuente para darnos la opción de poder seleccionar un horario.

Métodos:

- 1.- public Dialog onCreateDialog(Bundle savedInstanceState)

Esta clase nos mostrará el dialogo para poder seleccionar un horario. Obtendremos la hora del sistema y lo añadiremos como un valor inicial para que a nivel usuario sea más cómodo

```
1 package com.example.nero;
2
3 import ...
4
5
6
7
8 public class TimePicker extends DialogFragment {
9
10     @NonNull
11     Calendar c = Calendar.getInstance();
12     int hour = c.get(Calendar.HOUR_OF_DAY);
13     int minute = c.get(Calendar.MINUTE);
14
15     public Dialog onCreateDialog(Bundle savedInstanceState) {
16         return new TimePickerDialog(getActivity(), (TimePickerDialog.OnTimeSetListener) getActivity(), hour, minute, DateFormat.is24HourFormat(getActivity()));
17     }
18 }
```

2.-Paquete "PaqueteClasesPrincipales"

Este paquete nos enseña el funcionamiento de las principales instancias de la aplicación.

A) CALIFICACION

Esta clase cuenta con 3 atributos, un constructor y 2 métodos públicos que corresponden a los métodos getters y setters de un atributo de la clase.

Métodos

1.- Método setNota(int x)

Este método establece el valor del atributo nota a x.

2.- Método getNota()

Este método devuelve el valor almacenado en el atributo nota.

```
1 package PaqueteClasesPrincipales;
2
3 public class Calificacion {
4     private Usuario thisUser;
5     private Fuente thisFuente;
6     private int nota;
7
8     @
9     public Calificacion(Usuario esteUsuario, Fuente estaFuente){
10         thisFuente = estaFuente;
11         thisUser = esteUsuario;
12     }
13
14     public void setNota(int nota) { this.nota = nota; }
17
18     public int getNota() { return nota; }
21 }
```

B) COMENTARIO

Esta clase emula la instancia comentario. Cuenta con 5 atributos, un constructor y 9 métodos públicos que corresponden a los métodos getters y setters y un método equals propio para comparar objetos de esta clase.

Métodos

1.- Método getCreador()

Devuelve el valor del atributo creador.

Directorio Main

2.- Métodos setTexto y getTexto

Devuelven y establecen el valor del atributo texto.

3.- Métodos setFecha y getFecha

Devuelven y establecen el valor del atributo fecha.

4.- Método getId()

Devuelve el valor del atributo id.

5.- Métodos setFuente y getFuente

Devuelven y establecen el valor del atributo fuente.

6.- Método equals(object o)

Método boolean que devuelve true o false si 2 objetos de esta clase son iguales.

PD: esta clase implementa la interfaz “serializable” que nos permite “pasar” este objeto entre distintas actividades de la aplicación.

```
1 public class Comentario implements Serializable {
2     private int id;
3     private Usuario creador;
4     private String texto;
5     private String fecha;
6     private Fuente suFuente;
7
8     @
9     public Comentario(int id, Usuario creador){
10         this.id = id;
11         creador = creador;
12         suFuente = null;
13         texto = null;
14     }
15
16     public Usuario getCreador() { return creador; }
17
18     public String getTexto() { return texto; }
19
20     public void setTexto(String texto) { this.texto = texto; }
21
22     public String getFecha() { return fecha; }
23
24     public void setFecha(String fecha) { this.fecha = fecha; }
25
26     public int getId() { return id; }
27
28     public Fuente getSuFuente() { return suFuente; }
29
30     public void setSuFuente(Fuente o) { suFuente = o; }
31
32     @Override
33     public boolean equals(Object o) {
34         boolean t = false;
35         if (o instanceof Comentario) {
36             Comentario otroComentario = (Comentario) o;
37             if (this.id == otroComentario.getId()) {
38                 t = true;
39             }
40         }
41         return t;
42     }
43 }
```

C) FUENTE

En esta clase creamos el objeto/instancia fuente que se emulara como una fuente en la aplicación.

Métodos:

1.- Constructores "public Fuente()"

Estos dos métodos son constructores, en el primero se encuentra la ubicación de la fuente y el segundo le asigna una id.

2.- Métodos get y set "public void set() " y "public "tipoVariable" get("tipoVariable" x)"

Todos estos métodos son getters y setters cuyo nombre describe su funcionamiento.

3.- public boolean equals (Object o)

Este método verifica la igualdad del objeto instanciado y la fuente pasada por parámetros.

4.- public String toString()

Este método muestra por pantalla al usuario la información de la fuente.

D) UBICACIÓN

Esta clase emula la instancia ubicación. Cuenta con 2 atributos, un constructor y 7 metodos publicos entre los cuales se encuentran los metodos getters y setters además de un método hashCode, un método equals y un método toString.

Métodos

1.- Métodos getLatitud y setLatitud

Devuelven y establecen el valor del atributo latitud.

2.- Métodos getLongitud y setLongitud

Devuelven y establecen el valor del atributo longitud.

Directorio Main

3.- Método equals(object o)

Método boolean que devuelve true o false si 2 objetos de esta clase son iguales.

4.- Método hashCode()

Crea y devuelve un valor int para el hashCode de cada objeto de la clase.

5.- Método toString()

Crea y devuelve una cadena de texto en representación del objeto.

PD: esta clase implementa la interfaz “serializable” que nos permite “pasar” este objeto entre distintas activities de la aplicación.

```
1 package PaqueteClasesPrincipales;
2
3 import java.io.Serializable;
4
5 public class Ubicacion implements Serializable {
6
7     //Variables de instancia
8     private Double latitud;
9     private Double longitud;
10
11
12
13     @ public Ubicacion (Double latitud, Double longitud) {
14         this.latitud = latitud;
15         this.longitud = longitud;
16     }
17
18
19
20     //Getters para capturar la informacion desde otras clases
21     public Double getLatitud() { return latitud; }
22
23
24     public Double getLongitud() { return longitud; }
25
26
27
28
29
30
31     //Setters para modificar la informacion desde otras clases
32     public void setLatitud(Double latitud) { this.latitud = latitud; }
33
34
35     public void setLongitud(Double longitud) { this.longitud = longitud; }
36
37
38
39
40
41
42     @ public boolean equals (Object o) {
43         boolean t = false;
44         if (o instanceof Ubicacion) {
45             Ubicacion Ubicacion2 = (Ubicacion) o;
46             if (latitud.equals(Ubicacion2.getLatitud()) && (longitud.equals(Ubicacion2.getLongitud()))) {
47                 t = true;
48             }
49         }
50
51         return t;
52     }
```

```
55     @Override
56     public int hashCode() {
57         int suma;
58         suma = latitud.hashCode() + longitud.hashCode();
59         return suma;
60     }
61
62
63     @Override
64     public String toString() {
65         String cadena = "";
66         cadena= "Ubicacion (latitud: " + latitud + ", longitud: " + longitud + ")";
67         return cadena;
68     }
69
70
71 }
```

E) USUARIO

Esta clase emula la instancia Usuario. Tiene 7 atributos, 2 constructores y 16 métodos públicos que corresponden a métodos getters y setters, método equals, método hashCode y método toString. 2 de los atributos son listas de objetos de otras clases del mismo paquete.

Métodos

1.- Métodos getPassword y setPassword

Devuelven y establecen el valor del atributo password.

2.- Métodos getEmail y setEmail

Devuelven y establecen el valor del atributo email.

3.- Métodos getNombre y setNombre

Devuelven y establecen el valor del atributo nombre.

4.- Métodos getNombre_usuario y setNombre_usuario

Devuelven y establecen el valor del atributo Nombre_usuario.

5.- Métodos getBaneado y setBaneado

Devuelven y establecen el valor del atributo email.

Directorio Main

6.- Método equals(object o)

Método boolean que devuelve true o false si 2 objetos de esta clase son iguales.

7.- Método hashCode()

Crea y devuelve un valor int para el hashCode de cada objeto de la clase.

8.- Método toString()

Crea y devuelve una cadena de texto en representación del objeto.

PD: esta clase implementa la interfaz “serializable” que nos permite “pasar” este objeto entre distintas actividades de la aplicación.

Directorio Main

```
1 package PaqueteClasesPrincipales;
2 import ...
5
6 //Se implementa Serializable para que se pueda pasar este objeto entre distintas Activities
7 public class Usuario implements Serializable {
8
9     private String nombre_usuario;
10    private String password;
11    private String email;
12    private String nombre;
13    //...
14    private Boolean baneado;
15    private List<Comentario> comentarios;
16    private List<Fuente> susFuentes;
17
18    //-----CONSTRUCTORES
19    public Usuario(String nombre_usuario, String contra) {...}
20    public Usuario(String nombre_usuario){...}
21
22    //-----GETTERS
23    public String getNombre_usuario() { return nombre_usuario; }
24    public String getPassword() { return password; }
25    public String getEmail() { return email; }
26    public String getNombre() { return nombre; }
27    /*...*/
28    public Boolean getBaneado() { return baneado; }
29
30    //-----SETTERS
31    public void setNombre_usuario(String nombre_usuario) { this.nombre_usuario = nombre_usuario; }
32    public void setPassword(String password) { this.password = password; }
33    public void setEmail(String email) { this.email = email; }
34    public void setNombre(String nombre) { this.nombre = nombre; }
35
36    public void setBaneado(Boolean baneado) { this.baneado = baneado; }
37
38    //-----OVERRIDE
39    @Override
40    public boolean equals(Object o) {...}
41
42    @Override
43    public String toString() {...}
44
45    //-----GETTERS/SETTERS COMENTARIOS
46    public void anadirComentario(Comentario c){
47        comentarios.add(c);
48    }
49    public List<Comentario> getComentarios(){
50        return comentarios;
51    }
52
53    //-----GETTERS/SETTERS FUENTES
54    public void anadirFuente(Fuente f) { susFuentes.add(f); }
55    public List<Fuente> getSusFuentes() { return susFuentes; }
56 }
```

3.-Paquete "Paquete ReporteFactoryMethod"

En este paquete tendremos todas las clases e interfaces necesarias para poder implementar el patrón de diseño Factory Method. El problema, las soluciones y la adaptación de este patrón en el proyecto está completamente descrito en el documento de "Patrón de diseño", por lo que en aquí se explicará directamente el código de este patrón.

A) Factory

Esta interfaz solo contiene el método sin definir `getReporte(String tipoReporte)`. Esta interfaz será implementada por **ReporteFactoryMethod**.

```
public interface Factory {  
    public Reporte getReporte(String tipoReporte);  
}
```

B) Reporte

Esta interfaz definiremos los métodos que serán sobrescritos por las clases `ReporteComentario`, `ReporteFuente` y `ReporteUsuario`. Como podemos observar, tiene un tipo genérico T ya que será usado para el método `getLista()`.

```
public interface Reporte<T>{  
    boolean estaReportado(Object o);  
    void anyadirReporte(Object o);  
    List<T> getLista();  
}
```

C) ReporteComentario

En esta clase, tendremos un atributo de tipo `List<Comentario>` que tendrá una lista de comentarios que han sido reportados.

Métodos:

1.- `Public ReporteComentario()`

La función `ReporteComentario` es el constructor de esta clase que inicializará la lista como un `ArrayList`.

2.- `Public boolean estaReportado(Object o)`

La función `estaReportado` sobrescribirá el método de la interfaz. Aquí se realizará un parseo del parámetro de entrada (de tipo `Object`) a `Comentario`. En el caso de que el objeto no sea de la clase `Comentario` se lanzará una excepción (concretamente un `ReporteException`), en otro caso, se podrá comprobar con un algoritmo de búsqueda si dicho comentario ha sido reportado.

Directorio Main

```
@Override
public boolean estaReportado(Object o){

    boolean encontrado = false;

    if(o instanceof Comentario){

        Comentario obj = (Comentario) o;

        int i=0;
        while(!encontrado && i<comentariosReportados.size()){
            if(comentariosReportados.get(i).equals(obj))
                encontrado = true;
            i++;
        }

    } else {
        throw new ReporteException("Objeto no valido");
    }

    return encontrado;
}
```

3.- public void anyadirReporte(Object o)

Este método se encargará de añadir un comentario que ha sido reportado a la lista. En caso de que el parámetro de entrada no sea una instancia de la clase correspondiente (en este caso Comentario) se lanzará una excepción ReporteException.

```
@Override
public void anyadirReporte(Object o){
    if(o instanceof Comentario){
        Comentario obj = (Comentario) o;
        comentariosReportados.add(obj);
    } else {
        throw new ReporteException("Objeto no valido");
    }
}
```

4.- public List<Comentario> getLista()

Este método simplemente nos devolverá la lista que tenemos en la clase.

D) ReporteException

Esta clase se encargará de lanzar las excepciones cuando se intenta añadir o comprobar un objeto que no sea de la misma clase que la lista. Por ejemplo, si intentamos introducir un objeto que es instancia de la clase Usuario en el método de la clase ReporteComentario lanzará esta excepción.

```
public class ReporteException extends RuntimeException {
    public ReporteException(String mensaje) {
        super(mensaje);
    }
}
```


E) ReporteFactoryMethod

Esta clase se comportará como “la clase de la fábrica” del patrón de diseño.

Métodos:

1.- public Reporte getReporte(String tipoReporte)

Este método se encargará de la llamada de los constructores de los distintos reportes (ReporteComentario, ReporteUsuario, ReporteFuente) dependiendo de la String de entrada.

```
public class ReporteFactoryMethod implements Factory{

    public static final String tipoReporteUsuario = "ReporteUsuario";
    public static final String tipoReporteComentario = "ReporteComentario";
    public static final String tipoReporteFuente = "ReporteFuente";
    public Reporte getReporte(String tipoReporte){

        Reporte devolver = null;

        if(tipoReporte != null){
            if(tipoReporte.equalsIgnoreCase(tipoReporteUsuario)){
                devolver = new ReporteUsuario();
            } else if(tipoReporte.equalsIgnoreCase(tipoReporteComentario)){
                devolver = new ReporteComentario();
            } else if(tipoReporte.equalsIgnoreCase(tipoReporteFuente)){
                devolver = new ReporteFuente();
            }
        }

        return devolver;
    }
}
```

F) ReporteFuente

Esta clase se comportará exactamente de la misma forma que la clase anteriormente mencionada cambiando el tipo de la lista, que será una List<Fuente> y los métodos que en vez de comprobar si objeto o es una instancia de la clase Comentario comprobarán si dicho objeto es una instancia de la clase Fuente. También se lanzará el mismo tipo de excepción mencionada anteriormente cuando sea necesario.

G) ReporteUsuario

Esta clase se comportará exactamente de la misma forma que la clase anteriormente mencionada cambiando el tipo de la lista, que será una List<Usuario> y los métodos que en vez de comprobar si objeto o es una instancia de la clase Comentario comprobarán si dicho objeto es una instancia de la clase Usuario. También se lanzará el mismo tipo de excepción mencionada anteriormente cuando sea necesario.

4.-Paquete "PaqueteRequest"

Este paquete va dirigido a enviar los acciones realizadas en la aplicación al modelo de la aplicación.

A) AnyadirComentarioRequest

Esta clase se encarga de mandar los comentarios al PHP asignado.

Métodos:

1.- public AnyadirComentarioRequest(String nick, String pass, String texto, String idfuente, Response.Listener<String> listener)

Este método se encargará de mandar los comentarios a través de un hashmap al PHP.

```
AnyadirComentarioRequest.java
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class AnyadirComentarioRequest extends StringRequest {
11     private static final String ruta = "https://pruebasnero.000webhostapp.com/anyadircomentario.php";
12     private Map<String, String> val;
13
14     public AnyadirComentarioRequest(String nick, String pass, String texto, String idfuente, Response.Listener<String> listener){
15         super(Request.Method.POST, ruta, listener, errorListener: null);
16         val = new HashMap<>();
17         val.put("user", nick);
18         val.put("pass", pass);
19         val.put("text", texto);
20         val.put("idfuente", idfuente);
21     }
22
23     @Override
24     protected Map<String, String> getParams() { return val; }
25
26 }
27
28
```

B) AnyadirFuenteRequest

Esta clase se encarga de mandar las fuentes al PHP asignado.

Métodos:

1.- public AnyadirFuenteRequest(String nomUsuario, String latitud, String longitud, String dispo, String nomFuente, Response.Listener<String> listener)

Este método se encargará de mandar las fuentes a través de un hashmap al PHP.

```
AnyadirFuenteRequest.java
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class AnyadirFuenteRequest extends StringRequest {
11
12     private static final String ruta = "http://pruebasnero.000webhostapp.com/anyadirFuente.php";
13     private Map<String, String> val;
14
15     public AnyadirFuenteRequest(String nomUsuario, String latitud, String longitud, String dispo, String nomFuente, Response.Listener<String> listener){
16         super(Request.Method.POST, ruta, listener, errorListener: null);
17
18         val = new HashMap<>();
19         val.put("nomUsuario", nomUsuario);
20         val.put("latitud", latitud);
21         val.put("longitud", longitud);
22         val.put("dispo", dispo);
23         val.put("nomFuente", nomFuente);
24     }
25
26     @Override
27     protected Map<String, String> getParams() { return val; }
28
29 }
30
```

C) AnyadirReporteComentarioRequest

Esta clase se encarga de mandar los reportes de comentarios al PHP asignado.

Métodos:

1.- `public AnyadirReporteComentarioRequest(String nick, String pass, String idcomentario, String descripcion, Response.Listener<String> listener)`

Este método se encargará de mandar los reportes de los comentarios a través de un hashmap al PHP.

```
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class AnyadirReporteComentarioRequest extends StringRequest {
11     private static final String ruta = "https://pruebasnoro.000webhostapp.com/anyadirreportecomentario.php";
12     private Map<String, String> val;
13
14     public AnyadirReporteComentarioRequest(String nick, String pass, String idcomentario, String descripcion, Response.Listener<String> listener){
15         super(Request.Method.POST, ruta, listener, errorListener: null);
16         val = new HashMap<>();
17         val.put("user", nick);
18         val.put("pass", pass);
19         val.put("idcomentario", idcomentario);
20         val.put("descripcion", descripcion);
21     }
22
23     @Override
24     protected Map<String, String> getParams() { return val; }
25 }
26
```

D) AnyadirReporteFuenteRequest

Esta clase se encarga de mandar los reportes de las fuentes al PHP asignado.

Métodos:

1.- `public AnyadirReporteFuenteRequest(String nick, String pass, String idfuente, String descripcion, Response.Listener<String> listener)`

Este método se encargará de mandar los reportes de las fuentes a través de un hashmap al PHP.

```
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class AnyadirReporteFuenteRequest extends StringRequest {
11     private static final String ruta = "https://pruebasnoro.000webhostapp.com/anyadirreportefuente.php";
12     private Map<String, String> val;
13
14     public AnyadirReporteFuenteRequest(String nick, String pass, String idfuente, String descripcion, Response.Listener<String> listener){
15         super(Request.Method.POST, ruta, listener, errorListener: null);
16         val = new HashMap<>();
17         val.put("user", nick);
18         val.put("pass", pass);
19         val.put("idfuente", idfuente);
20         val.put("descripcion", descripcion);
21     }
22
23     @Override
24     protected Map<String, String> getParams() { return val; }
25 }
26
```

E) AnyadirReporteUsuarioRequest

Esta clase se encarga de mandar los reportes de los usuarios al PHP asignado.

Métodos:

1.- *public AnyadirReporteUsuarioRequest(String thisUserNick, String thisUserPass, String reportado, String texto, Response.Listener<String> listener)*

Este método se encargará de mandar los reportes de los usuarios a través de un hashmap al PHP.

```
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class AnyadirReporteUsuarioRequest extends StringRequest {
11     private static final String ruta = "https://pruebasnero.000webhostapp.com/anyadirreporteusuario.php";
12     private Map<String, String> val;
13
14     public AnyadirReporteUsuarioRequest(String thisUserNick, String thisUserPass, String reportado, String texto, Response.Listener<String> listener) {
15         super(Request.Method.POST, ruta, listener, errorListener: null);
16         val = new HashMap<>();
17         val.put("thisuser", thisUserNick);
18         val.put("thispass", thisUserPass);
19         val.put("descripcion", texto);
20         val.put("reportado", reportado);
21     }
22
23     @Override
24     protected Map<String, String> getParams() { return val; }
25 }
26
27
28
```

F) ComentarioRequest

Esta clase se encarga de mandar las consultas del usuario al PHP asignado.

Métodos:

1.- *public ComentariosRequest(Response.Listener<String> listener, String fuente_id)*

Este método se encargará de mandar los comentarios de los usuarios a través de un hashmap al PHP.

```
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class ComentariosRequest extends StringRequest {
11     private static final String ruta = "https://pruebasnero.000webhostapp.com/consultascomentarios.php";
12     private Map<String, String> val;
13
14     public ComentariosRequest(Response.Listener<String> listener, String fuente_id){
15         super(Request.Method.POST, ruta, listener, errorListener: null);
16         val = new HashMap<>();
17         val.put("fuenteid", fuente_id);
18     }
19
20     @Override
21     protected Map<String, String> getParams() { return val; }
22 }
23
24
25
```

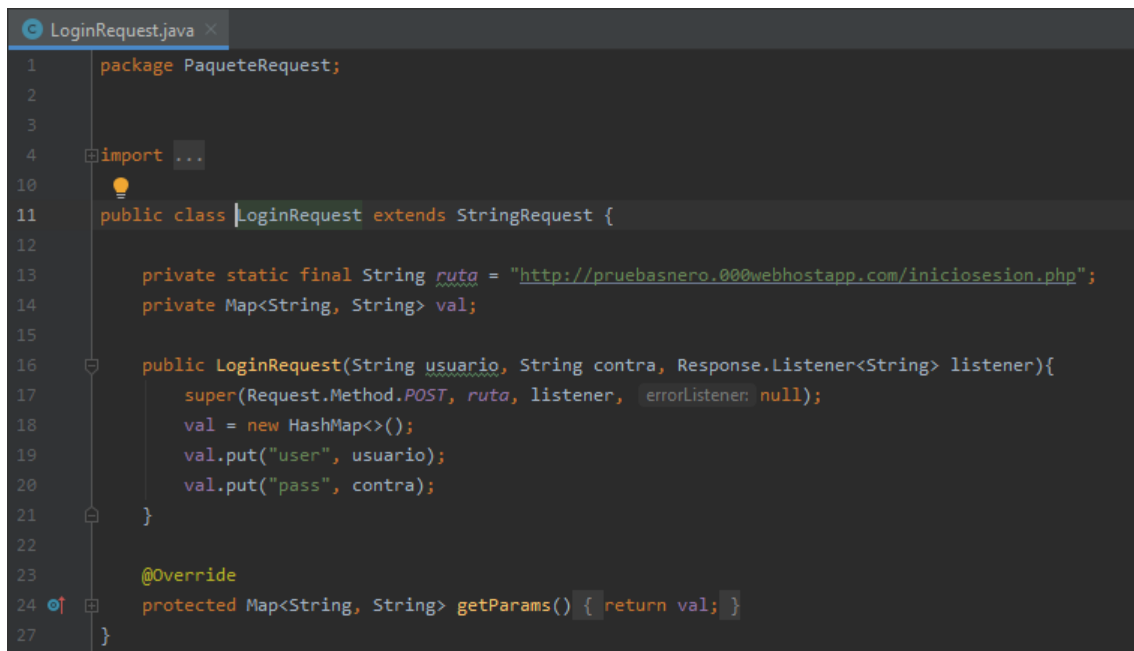
G) LoginRequest

Esta clase se encarga de mandar los datos del login al PHP asignado.

Métodos:

1.- `public LoginRequest(String usuario, String contra, Response.Listener<String> listener)`

Este método se encargará de mandar los login de los usuarios a través de un hashmap al PHP.



```
1 package PaqueteRequest;
2
3
4 import ...
5
10
11 public class LoginRequest extends StringRequest {
12
13     private static final String ruta = "http://pruebasnero.000webhostapp.com/iniciosesion.php";
14     private Map<String, String> val;
15
16     public LoginRequest(String usuario, String contra, Response.Listener<String> listener){
17         super(Request.Method.POST, ruta, listener, errorListener: null);
18         val = new HashMap<>();
19         val.put("user", usuario);
20         val.put("pass", contra);
21     }
22
23     @Override
24     protected Map<String, String> getParams() { return val; }
25
26
27 }
```

H) MapaRequest

Esta clase se encarga de mandar los datos de la ubicación al PHP asignado.

Métodos:

1.- `public LoginRequest(String usuario, String contra, Response.Listener<String> listener)`

Este método se encargará de mandar los login de los usuarios a través de un hashmap al PHP.

Directorio Main

```
MapaRequest.java
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class MapaRequest extends StringRequest {
11     private static final String ruta = "https://pruebasnero.000webhostapp.com/consultasfuente.php";
12     private Map<String, String> val;
13
14     public MapaRequest(Response.Listener<String> listener){
15         super(Request.Method.POST, ruta, listener, errorListener: null);
16         val = new HashMap<>();
17         //val.put("user", usuario);
18         //val.put("pass", contra);
19     }
20
21     @Override
22     protected Map<String, String> getParams() { return val; }
23
24
25 }
```

1) sign_inRequest

Esta clase se encarga de mandar los datos del registro al PHP asignado.

Métodos:

1.- *public sign_inRequest(String usuario, String contra, String email, String nombre, Response.Listener<String> respuesta)*

Este método se encargará de mandar los registros de los usuarios a través de un hashmap al PHP.

```
sign_inRequest.java
1 package PaqueteRequest;
2
3 import ...
4
5
6
7
8
9
10 public class sign_inRequest extends StringRequest {
11     private static final String ruta = "http://pruebasnero.000webhostapp.com/registro.php";
12     private Map<String, String> val;
13
14     public sign_inRequest(String usuario, String contra, String email, String nombre, Response.Listener<String> respuesta) {
15         super(Request.Method.POST, ruta, respuesta, errorListener: null);
16         val = new HashMap<>();
17         val.put("user", usuario);
18         val.put("pass", contra);
19         val.put("correo", email);
20         val.put("name", nombre);
21     }
22
23     @Override
24     protected Map<String, String> getParams() { return val; }
25
26
27 }
28
```

5.-Paquete "Ajustes"

Este paquete se encarga de dirigir los ajustes de la aplicación.

A) Ajustes

La clase se encarga de los ajustes de la aplicación

Las variables son un usuario (Usuario usuario) y tres botones que corresponden a los de la activity.

```
Ajustes.java
1 package Ajustes;
2
3 import android.content.Intent;
4 import android.net.Uri;
5 import android.os.Bundle;
6 import android.view.View;
7 import android.widget.Button;
8
9 import androidx.appcompat.app.AppCompatActivity;
10
11 import com.example.nero.R;
12
13 import PaqueteClasesPrincipales.Usuario;
14
15 public class Ajustes extends AppCompatActivity {
16     private Button btnPermisos, btnEditarPerfil, btnModificarFuente;
17     private Usuario usuario;
18
19     private final int REQUEST_CODE_MODIFICAR_PERFIL = 1;
20
21     @Override
22     protected void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.activity_ajustes);
25
26         /*
27         btnPermisos = (Button) findViewById(R.id.btnModificarPermisos);
28         btnEditarPerfil = (Button) findViewById(R.id.btnEditarPerfil);
29         btnModificarFuente = (Button) findViewById(R.id.btnModificarFuente);*/
30     }
31 }
```

Métodos:

1.- protected void onCreate(Bundle savedInstanceState)

Este método funciona como un main donde se recogen variables, valores, etc.

2.- public void modificarPermisos(View view)

Este método nos envía a la pantalla de ajustes de nuestra aplicación para modificar los permisos.

```
31 obtenerDatosUsuario();
32
33 }
34
35
36 /*Método que nos envía a la pantalla de Ajustes de nuestra aplicación. Para el cambio de los
37 * permisos */
38 public void modificarPermisos(View view){
39
40     Intent i = new Intent(android.provider.Settings.ACTION_APPLICATION_DETAILS_SETTINGS);
41     Uri uri = Uri.fromParts("package", getApplicationContext().getPackageName(), fragment: null);
42     i.setData(uri);
43     startActivity(i);
44 }
45
46 public void editarPerfil(View view){
47
48     Intent i = new Intent(getApplicationContext(), EditarPerfil.class);
49     i.putExtra("name: " + "Usuario", usuario);
50     startActivityForResult(i, REQUEST_CODE_MODIFICAR_PERFIL);
51 }
52
53 public void modificarFuente(View view){
54
55     Intent i = new Intent(getApplicationContext(), ModificarFuente.class);
56     startActivity(i);
57 }
58 }
```

Directorio Main

3.- public void editarPerfil(View view)

Inicia la actividad de editar el perfil.

4.- public void modificarFuente(View view)

Inicia la actividad de modificar una fuente.

5.- protected void onActivityResult(int requestCode, int resultCode, Intent data)

Este es un método necesario para recoger un valor de vuelta en la segunda activity.

6.- private void obtenerDatosUsuario()

Este método sirve para obtener los datos del usuario de la activity anterior.

```
59         startActivity(i);
60     }
61 }
62
63 @Override
64 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
65     super.onActivityResult(requestCode, resultCode, data);
66     if (resultCode == RESULT_OK && requestCode == REQUEST_CODE_MODIFICAR_PERFIL) {
67         if (data.hasExtra("UsuarioModificado")) {
68             usuario = (Usuario) data.getExtras().getSerializable(key: "UsuarioModificado");
69         }
70     }
71 }
72
73
74 //Obtenemos los datos del usuario de la activity anterior
75 private void obtenerDatosUsuario(){
76     Bundle extras = getIntent().getExtras();
77
78     usuario = (Usuario) extras.getSerializable(key: "usuario");
79 }
80 }
81 }
```


B) EditarPerfil

Esta clase se encarga de editar el perfil del usuario. Tiene variables de tipo Usuario y EditText.

Métodos:

1.- protected void

onCreate(Bundle savedInstanceState)

Este método funciona como un main donde se recogen variables, valores, etc.

2.- private void obtenerDatosUsuario()

Este método obtiene los datos del usuario de la activity anterior. Recoge datos del usuario desde la base de datos.

3.- public void guardarCambios(View view)

En esta función se comprueba que los nuevos datos sean diferentes a los anteriores y los actualiza.

4.- public void finish()

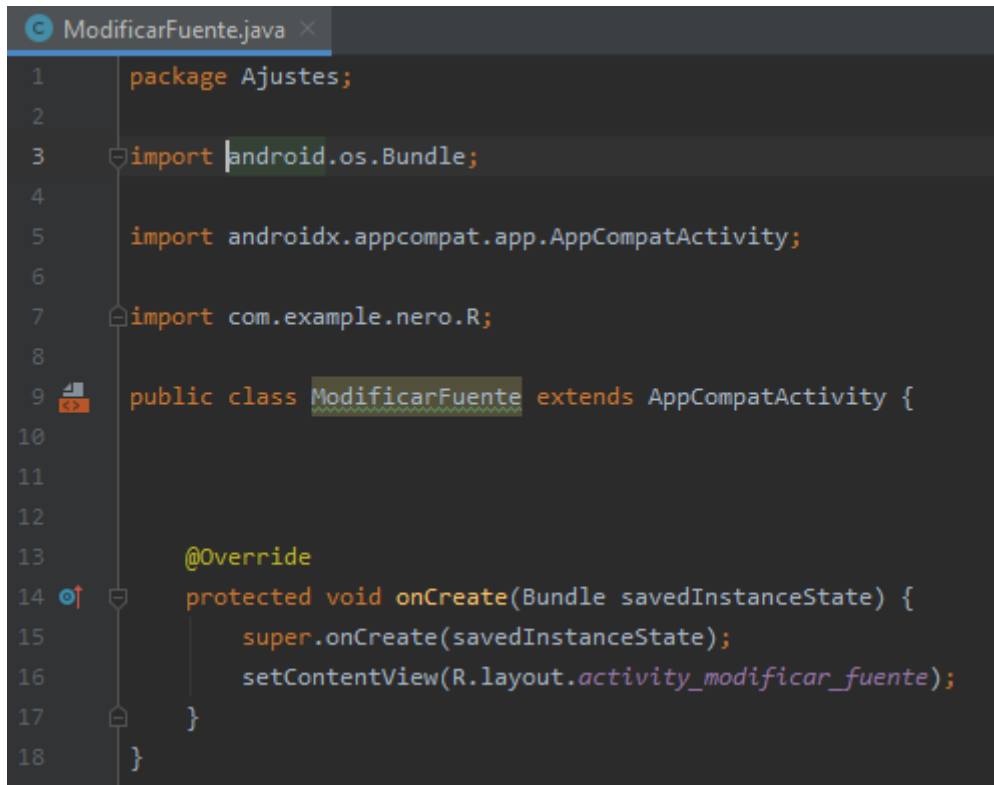
Envía un valor a la primera pestaña mediante el método onActivityResult.

c) ModificarFuente

La clase se encarga de modificar los atributos de las fuentes propias del usuario.

1.- protected void onCreate(Bundle savedInstanceState)

Este método funciona como un main donde se recogen variables, valores, etc.



```
1 package Ajustes;
2
3 import android.os.Bundle;
4
5 import androidx.appcompat.app.AppCompatActivity;
6
7 import com.example.nero.R;
8
9 public class ModificarFuente extends AppCompatActivity {
10
11
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_modificar_fuente);
17     }
18 }
```

6.-Paquete "PaqueteRequestQueueSingleton"

La aplicación tiene un gran número de Activities (pantallas) en la que cada una realiza una solicitud distinta. Para realizar estas solicitudes debemos realizar instancias de RequestQueue, por lo que cada Activity(pantalla) se crea una instancia de RequestQueue cuando no es necesario: Con realizar **una sola instancia de RequestQueue** es suficiente.

Este patrón se ha implementado siguiendo las pautas de implementación en la documentación de Android Studio. Android Studio siempre recomienda usar este patrón si la aplicación realiza un gran número de consultas.

Para más información respecto a Volley usando el patron singleton:

<https://developer.android.com/training/volley/requestqueue>

A) SingletonRequestQueue

Con esta clase podemos asegurar que solo tenemos una sola instancia, por lo que al realizar esto en aplicaciones donde se realizan muchas consultas lo hacemos más eficiente.

Por último, cabe mencionar que unos de los problemas del patrón Singleton es que en un entorno multiproceso, donde varios hilos pueden crear el objeto SingletonRequestQueue varias veces. Para evitar esto hemos utilizado la palabra reservada synchronized, evitando así, un problema bastante grave de este patrón de diseño.

```
1 import ...
2
3
4
5
6
7
8
9 public class SingletonRequestQueue {
10
11     private static SingletonRequestQueue instance;
12     private static Context contexto;
13     private RequestQueue requestQueue;
14
15
16     private SingletonRequestQueue(Context context) {
17         contexto = context;
18         requestQueue = getRequestQueue();
19     }
20
21     public static synchronized SingletonRequestQueue getInstance(Context context) {
22         if (instance == null) {
23             instance = new SingletonRequestQueue(context);
24         }
25         return instance;
26     }
27
28     public RequestQueue getRequestQueue() {
29         if (requestQueue == null) {
30             requestQueue = Volley.newRequestQueue(contexto.getApplicationContext());
31         }
32         return requestQueue;
33     }
34
35     public <T> void addToRequestQueue(Request<T> req) {
36         getRequestQueue().add(req);
37     }
38
39 }
```

Directorio Test

1.-Paquete "com.example.nero"

Este paquete es usado para las técnicas de testeo de la aplicación.

A) ReporteFactoryMethodTest

Esta clase no tiene atributos ya que es una clase para test, en concreto se encarga de hacer el test a la clase ReporteFactoryMethod. Consta de 5 métodos de los cuales dos de ellos son el principio y el final del test.

Métodos

1.- Método init()

Inicializa el test, en este método se crea el objeto de la clase a testear así como cualquier elemento que necesitemos durante toda la ejecución del test.

2.- Método terminate()

Termina el test, en este método se cierran los objetos creados en el método anterior.

3.- Método comprobarFactoryMethodComentario()

Comprueba que la clase ReporteFactoryMethod reconoce un objeto comentario cuando se le pasa una instancia del mismo.

4.- Método comprobarFactoryMethodFuente ()

Comprueba que la clase ReporteFactoryMethod reconoce un objeto Fuente cuando se le pasa una instancia del mismo.

5.- Método comprobarFactoryMethodUsuario()

comprueba que la clase ReporteFactoryMethod reconoce un objeto Usuario cuando se le pasa una instancia de este.

PD: se usa Junit y Mockito.

Directorio Test

```
19 public class ReporteFactoryMethodTest {
20     ReporteFactoryMethod reporte;
21
22     @Before
23     public void init() { reporte = new ReporteFactoryMethod(); }
24
25     @After
26     public void terminate() { reporte = null; }
27
28     @Test
29     public void comprobarFactoryMethodComentario(){
30         //Boolean res;
31         if(!reporte.getReporte(ReporteFactoryMethod.tipoReporteComentario) instanceof ReporteComentario){
32             fail("No es una instancia");
33         }
34
35         /* res = true;
36         }else{
37             res = true;
38         }
39         assertEquals(true, res);*/
40     }
41
42     @Test
43     public void comprobarFactoryMethodFuente(){
44         Boolean res;
45         if(reporte.getReporte( tipoReporte: "ReporteFuente") instanceof ReporteFuente ){
46             res = true;
47         }else{
48             res = true;
49         }
50         assertEquals( expected: true, res);
51     }
52
53     @Test
54     public void comprobarFactoryMethodUsuario(){
55         Boolean res;
56         if(reporte.getReporte( tipoReporte: "ReporteUsuario") instanceof ReporteUsuario){
57             res = true;
58         }else{
59             res = true;
60         }
61         assertEquals( expected: true, res);
62     }
63 }
```

B) TestComentario

Esta clase tiene 2 atributos a pesar de ser una clase Test y en concreto se encarga de hacer el test a la clase Fuente. Consta de 10 métodos de los cuales dos de ellos son el principio y el final del test. Los dos atributos que tiene son un objeto Fuente a evaluar y una ubicación mockeada.

Métodos

1.- Método init()

Inicializa el test, en este método se crea el objeto de la clase a testear así como cualquier elemento que necesitemos durante toda la ejecución del test.

Directorio Test

2.- Método terminate()

Termina el test, en este método se cierran los objetos creados en el método anterior.

3.- Método comprobarID()

Comprueba el método getID().

4.- Método comprobarUbicacion()

Comprueba el método getUbicacion().

5.- Método comprobarDisponibilidad()

Comprueba el método getDisponibilidad ().

6.- Método comprobarHabilitado()

Comprueba que los metodos para habilitar y deshabilitar funcionen.

7.- Método comprobarCreador ()

Comprueba el método getCreador ().

8.- Método comprobarEqual ()

Comprueba el método equal().

9.- Método comprobarToString ()

Comprueba el método toString().

10.- Método comprobarComentariosAsociados()

Comprueba el funcionamiento de la lista de comentarios asociados a los objetos fuentes.

Directorio Test

```
24 @RunWith(MockitoJUnitRunner.class)
25 public class FuenteTest {
26     Fuente fuente;
27     Ubicacion ubi = mock(Ubicacion.class);
28     @Before
29     public void init(){
30         fuente = new Fuente( id: 1, ubi);
31     }
32
33     @After
34     public void terminate(){fuente = null; ubi = null;}
35
36     @Test
37     public void comprobarID() { fuente.getId(); }
38
39     @Test
40     public void comprobarUbicacion() { fuente.getUbicacion(); }
41
42     @Test
43     public void comprobarDisponibilidad(){
44         assertEquals( expected: null, fuente.getDisponibilidad());
45         fuente.setDisponibilidad("DE TARDE");
46         assertEquals( expected: "DE TARDE", fuente.getDisponibilidad());
47     }
48
49     @Test
50     public void comprobarHabilitado(){...}
51
52     @Test
53     public void comprobarCreador(){...}
54
55     @Test
56     public void comprobarEqual(){...}
57
58     @Test
59     public void comprobarToString(){...}
60
61     @Test
62     public void comprobarComentariosAsociados(){
63         Comentario coment = mock(Comentario.class);
64         List<Comentario> misComents = fuente.getComentarios();
65         assertEquals( expected: true, misComents.isEmpty());
66         fuente.anyadirComentario(coment);
67         misComents = fuente.getComentarios();
68         assertEquals( expected: true, misComents.contains(coment));
69     }
70 }
71 }
```

C) TestReporteComentario

Esta clase tiene 1 atributo a pesar de ser una clase Test y en concreto se encarga de hacer el test a la clase ReporteComentario. Consta de 7 métodos de los cuales dos de ellos son el principio y el final del test. El atributo corresponde a un objeto ReporteComentario.

Métodos

1.- Método init()

Inicializa el test, en este método se crea el objeto de la clase a testear así como cualquier elemento que necesitemos durante toda la ejecución del test.

2.- Método terminate()

Termina el test, en este método se cierran los objetos creados en el método anterior.

3.- Método comprobarSiListaVacía ()

Comprueba que la lista de reportes este vacía al iniciar.

4.- Método anyadirObjetodeOtroTipoNoValido()

Comprueba que la clase ReporteComentario funcione correctamente cuando se le introduce un tipo de dato no valido.

5.- Método anyadirObjetoDeEsteTipoValido()

Comprueba que funcione correctamente cuando se introduce un tipo valido.

6.- Método comprobarSiEstaReportadoUnReporteValido();

7.- Método comprobarSiEstaReportadoNoValido();

Directorio Test

```
14 public class TestReporteComentario {
15     ReporteComentario reporteComentario;
16
17     @Before
18     public void init() { reporteComentario = new ReporteComentario(); }
19
20     @After
21     public void terminate() { reporteComentario = null; }
22
23     @Test
24     public void comprobarSiListaVacía() {
25         Comentario comentario = mock(Comentario.class);
26         assertEquals( expected: 0, reporteComentario.getLista().size());
27     }
28
29     @Test(expected = ReportException.class)
30     public void anyadirObjetoDeOtroTipoNoValido() {
31         //probamos con la clase Usuario pero podría probarse también con Fuente
32         Usuario otroTipoDeObjeto = mock(Usuario.class);
33         reporteComentario.anyadirReporte(otroTipoDeObjeto);
34     }
35
36     @Test
37     public void anyadirObjetoDeEsteTipoValido() { ... }
38
39     @Test
40     public void comprobarSiEstaReportadoUnReporteValido() {
41         Comentario comentario = mock(Comentario.class);
42         when(comentario.getId()).thenReturn(200);
43
44         assertEquals( expected: false, reporteComentario.estaReportado(comentario));
45         reporteComentario.anyadirReporte(comentario);
46         assertEquals( expected: true, reporteComentario.estaReportado(comentario));
47     }
48
49     @Test(expected = ReportException.class)
50     public void comprobarSiEstaReportadoUnReporteNoValido() {
51         Usuario usuario = mock(Usuario.class);
52         reporteComentario.anyadirReporte(usuario);
53     }
54 }
```

D) testReportFuentes

Esta clase tiene 1 atributo a pesar de ser una clase Test y en concreto se encarga de hacer el test a la clase ReporteFuente. Consta de 7 métodos de los cuales dos de ellos son el principio y el final del test. El atributo corresponde a un objeto ReporteFuente.

Métodos

1.- Método init()

Inicializa el test, en este método se crea el objeto de la clase a testear así como cualquier elemento que necesitemos durante toda la ejecución del test.

2.- Método terminate()

Termina el test, en este método se cierran los objetos creados en el método anterior.

3.- Método comprobarSiListaVacía ()

Comprueba que la lista de reportes este vacía al iniciar.

4.- Método anyadirObjetodeOtroTipoNoValido()

Comprueba que la clase ReporteFuente funcione correctamente cuando se le introduce un tipo de dato no valido.

Directorio Test

5.- Método `anyadirObjetoDeEsteTipoValido()`

Comprueba que funcione correctamente cuando se introduce un tipo valido.

6.- Método `comprobarSiEstaReportadoUnReporteValidor();`

7.- Método `comprobarSiEstaReportadoNoValido();`

Directorio Test

```
25 public class TestReportFuentes {
26
27     ReporteFuente rf;
28
29     @Before
30     public void init() { rf = new ReporteFuente(); }
31
32     @After
33     public void terminate() { rf = null; }
34
35
36
37
38     @Test
39     public void comprobarSiListaVacía(){...}
40
41
42
43
44     @Test(expected = ReporteException.class)
45     public void anyadirObjetoDeOtroTipoNoValido(){...}
46
47
48
49
50
51     @Test
52     public void anyadirObjetoDeEsteTipoValido(){
53         Fuente fuente = mock(Fuente.class);
54         try{
55             rf.anyadirReporte(fuente);
56         }catch(ReporteException ex){
57             fail("No debería lanzar excepcion");
58         }
59     }
60
61
62     @Test
63     public void comprobarSiEstaReportadoUnReporteValido(){
64         Fuente fuente = mock(Fuente.class);
65         when(fuente.getId()).thenReturn(200);
66
67         assertEquals( expected: false, rf.estaReportado(fuente));
68         rf.anyadirReporte(fuente);
69         assertEquals( expected: true, rf.estaReportado(fuente));
70     }
71
72     @Test(expected = ReporteException.class)
73     public void comprobarSiEstaReportadoUnReporteNoValido(){
74         Usuario usuario = mock(Usuario.class);
75         rf.anyadirReporte(usuario);
76     }
77 }
```

```
activity_maps.java
43 public class activity_maps extends FragmentActivity implements OnMapReadyCallback, OnMyLocationButtonClickListener, GoogleMap.OnMarkerClickListener {
44
45     //Mapa y localización
46     private GoogleMap mMap;
47     private FusedLocationProviderClient fusedLocationClient;
48
49     //Constantes y códigos
50     private static final float DEFAULT_ZOOM = 15f;
51     private final static int CODIGO_DE_PERMISO_LOCALIZACION = 123;
52
53     //Elementos de la pantalla
54     private Button btn_anyadirFuente;
55
56     //Datos
57     private Usuario thisUser;
58     private Double latitudThisUser;
59     private Double longitudThisUser;
60     private List<Fuente> fuentes;
61
62     //-----
63     @Override
64     protected void onCreate(Bundle savedInstanceState) {
65         //Default
66         super.onCreate(savedInstanceState);
67         setContentView(R.layout.activity_maps);
68         SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
69             .findFragmentById(R.id.map);
70         mapFragment.getMapAsync( onMapReadyCallback: this);
71     }
72 }
```

Directorio Test

E) testReportUsuarios

Esta clase tiene 1 atributo a pesar de ser una clase Test y en concreto se encarga de hacer el test a la clase ReporteUsuario. Consta de 7 métodos de los cuales dos de ellos son el principio y el final del test. El atributo corresponde a un objeto ReporteUsuario.

Métodos

1.- Método init()

Inicializa el test, en este método se crea el objeto de la clase a testear así como cualquier elemento que necesitemos durante toda la ejecución del test.

2.- Método terminate()

Termina el test, en este método se cierran los objetos creados en el método anterior.

3.- Método comprobarSiListaVacía ()

Comprueba que la lista de reportes este vacía al iniciar.

4.- Método anyadirObjetodeOtroTipoNoValido()

Comprueba que la clase ReporteUsuario funcione correctamente cuando se le introduce un tipo de dato no válido.

5.- Método anyadirObjetoDeEsteTipoValido()

Comprueba que funcione correctamente cuando se introduce un tipo válido.

6.- Método comprobarSiEstaReportadoUnReporteValido();

7.- Método comprobarSiEstaReportadoNoValido();

Directorio Test

```
25 public class testReportUsuarios {
26
27     ReporteUsuario ru;
28
29     @Before
30     public void init() { ru = new ReporteUsuario(); }
31
32     @After
33     public void terminate() { ru = null; }
34
35
36
37
38     @Test
39     public void comprobarListaVacía(){
40         Usuario usuario = mock(Usuario.class);
41         assertEquals( expected: 0, ru.getLista().size());
42     }
43
44     @Test(expected = ReporteException.class)
45     public void anyadirObjetoDeOtroTipoNoValido(){
46         //Probamos con la clase Comentario pero podria probarse tambien con Fuente
47         Comentario invalido = mock(Comentario.class);
48         ru.anyadirReporte(invalido);
49     }
50
51     @Test
52     public void anyadirObjetoDeEsteTipoValido(){
53         Usuario usuario = mock(Usuario.class);
54         try{
55             ru.anyadirReporte(usuario);
56         }catch(ReporteException ex){
57             fail("No deberia lanzar excepcion");
58         }
59     }
60
61     @Test
62     public void comprobarSiEstaReportadoUnReporteValido(){...}
63
64
65
66
67
68
69
70
71     @Test(expected = ReporteException.class)
72     public void comprobarSiEstaReportadoUnReporteNoValido(){...}
```

F) TestUbicación

Esta clase tiene 1 atributo a pesar de ser una clase Test y en concreto se encarga de hacer el test a la clase Ubicación. Consta de 5 métodos de los cuales dos de ellos son el principio y el final del test. El atributo corresponde a un objeto ReporteUsuario.

Métodos

1.- Método init()

Inicializa el test, en este método se crea el objeto de la clase a testear así como cualquier elemento que necesitemos durante toda la ejecución del test.

2.- Método terminate()

Termina el test, en este método se cierran los objetos creados en el método anterior.

Directorio Test

3.- Método comprobarUbicacion()

Comprueba la ubicación de una fuente.

4.- Método comprobarEqualsHashCode()

Comprueba el método equals() y el hashCode().

5.- Método comprobarToString()

Comprueba el método toString();

```
13  ▶ public class TestUbicación {
14      Ubicacion ubicacion;
15
16      @Before
17      public void init() { ubicacion = new Ubicacion( latitud: 1.1, longitud: 2.2); }
20      @After
21      public void terminate() { ubicacion = null; }
24
25
26      @Test
27      ▶ public void comprobarUbicacion(){
28          Ubicacion ubicacion2 = mock(Ubicacion.class);
29          when(ubicacion2.getLatitud()).thenReturn(1.1);
30          when(ubicacion2.getLongitud()).thenReturn(2.2);
31
32          assertEquals( expected: true, ubicacion.equals(ubicacion2));
33      }
34
35      @Test
36      ▶ public void comprobarEqualsHashCode(){
37          Ubicacion ubi2 = ubicacion;
38          Ubicacion ubi3 = mock(Ubicacion.class);
39          when(ubi3.getLatitud()).thenReturn(1.1);
40          when(ubi3.getLongitud()).thenReturn(2.2);
41
42          assertEquals(ubicacion.hashCode(), ubi2.hashCode(), delta: 0);
43          assertEquals(ubicacion.hashCode(), ubi3.hashCode(), delta: 0);
44      }
45
46      @Test
47      ▶ public void comprobarToString(){
48          String str = ("Ubicación (latitud: 1.1, longitud: 2.2)");
49          assertEquals(str,ubicacion.toString());
50      }
51  }
```

G) UsuarioTest

Esta clase tiene 1 atributo a pesar de ser una clase Test y en concreto se encarga de hacer el test a la clase Usuario. Consta de 9 métodos de los cuales dos de ellos son el principio y el final del test. El atributo corresponde a un objeto Usuario.

Métodos

1.- Método init()

Inicializa el test, en este método se crea el objeto de la clase a testear así como cualquier elemento que necesitemos durante toda la ejecución del test.

2.- Método terminate()

Termina el test, en este método se cierran los objetos creados en el método anterior.

3.- Método comprobarBaneado();

4.- Método comprobarEmail();

5.- Método comprobarToString();

6.- Método comprobarNombre();

7.- Método comprobarUsuarioIgualIgnoreCase();

8.- Método comprobarFuenteAsociado()

Comprueba que la lista de fuentes asociadas al usuario funcione correctamente.

9.- Método comprobarComentariosAsociados()

Comprueba que la lista de comentarios asociadas al usuario funcione correctamente.

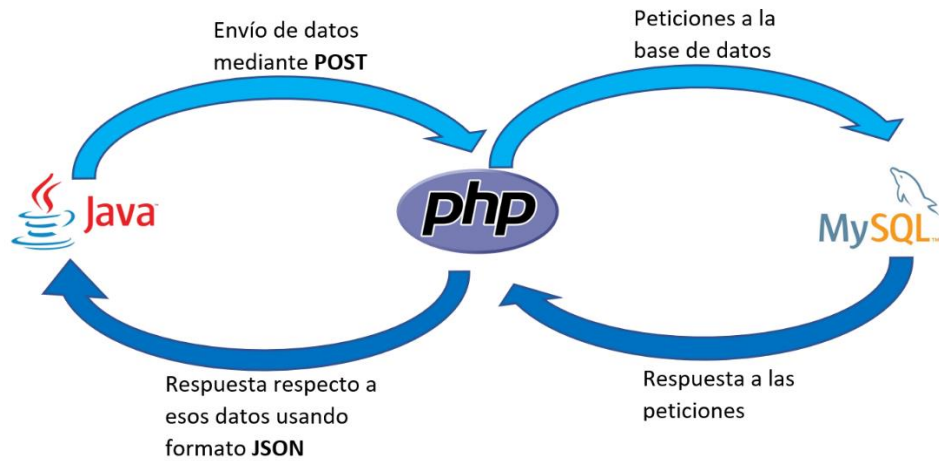
Directorio Test

```
49 @RunWith(MockitoJUnitRunner.class)
50 public class UsuarioTest {
51     Usuario user;
52     @Before
53     public void init(){user = new Usuario( nombre_usuario: "Admin", contrai: "1234");}
54
55     @After
56     public void terminate(){user = null;}
57
58     @Test
59     public void comprobarBaneado(){
60         assertEquals( expected: null, user.getBaneado());
61         user.setBaneado(true);
62         assertEquals( expected: true, user.getBaneado());
63         user.setBaneado(false);
64         assertEquals( expected: false, user.getBaneado());
65     }
66
67     @Test
68     public void comprobarEmail(){
69         String nom = "Juan@prueba.com";
70         user.setEmail(nom);
71         assertEquals( expected: "Juan@prueba.com", user.getEmail());
72     }
73
74     @Test
75     public void comprobarToString(){
76         user.setEmail("Juan@prueba.com");
77         user.setNombre("Juan");
78         user.setBaneado(false);
79         String cadena = ("Usuario con nickname: Admin, contrase@a: 1234, email: Juan@prueba.com, nombre real: Juan y no baneado");
80         assertEquals(cadena, user.toString());
81     }
82
83     @Test
84     public void comprobarNombre(){
85         String nom = "Juan";
86         user.setNombre(nom);
87         assertEquals( expected: "Juan", user.getNombre());
88     }
89 }
```

```
63
64 @Test
65 public void comprobarUsuarioIgualIgnoreCase(){
66     Usuario user2 = new Usuario( nombre_usuario: "admin", contrai: "1234");
67     Boolean res1 = user2.equals(user);
68     assertEquals( expected: true, res1);
69     Usuario user3 = new Usuario( nombre_usuario: "Pepe", contrai: "raul");
70     assertEquals( expected: false, user.equals(user3));
71 }
72
73 @Test
74 public void comprobarFuenteAsociada(){
75     Fuente fuente = mock(Fuente.class);
76     List<Fuente> misFuentes = user.getSusFuentes();
77     assertEquals( expected: true, misFuentes.isEmpty());
78     user.anyadirFuente(fuente);
79     misFuentes = user.getSusFuentes();
80     assertEquals( expected: true, misFuentes.contains(fuente));
81 }
82
83 @Test
84 public void comprobarComentariosAsociados(){
85     Comentario coment = mock(Comentario.class);
86     List<Comentario> misComents = user.getComentarios();
87     assertEquals( expected: true, misComents.isEmpty());
88     user.anyadirComentario(coment);
89     misComents = user.getComentarios();
90     assertEquals( expected: true, misComents.contains(coment));
91 }
92 }
```


I.-Paquetes PHP

El paquete de clases PHP es un paquete de clases implementado dentro de nuestro servidor. Este paquete se encarga de recibir los datos de las clases request de java y enviárselas a la base de datos.



	Name	Size	Date	Permissions
	htaccess	0.2 kB	2020-04-23 15:28:00	-rw-r--r--
	anyedreportecomentario.php	1.2 kB	2020-05-15 17:27:00	-rw-r--r--
	anyedreportefuente.php	0.8 kB	2020-05-03 13:28:00	-rw-r--r--
	anyedreportecomentario.php	2.1 kB	2020-05-22 15:37:00	-rw-r--r--
	anyedreportefuente.php	2.1 kB	2020-05-23 01:56:00	-rw-r--r--
	anyedreportefuente.php	2.4 kB	2020-05-30 04:14:00	-rw-r--r--
	conexion.php	0.3 kB	2020-06-01 23:09:00	-rw-r--r--
	consultacomentarios.php	0.6 kB	2020-05-12 00:17:00	-rw-r--r--
	consultafuente.php	0.4 kB	2020-05-06 03:13:00	-rw-r--r--
	iniciosesion.php	1.0 kB	2020-04-26 19:06:00	-rw-r--r--
	pruebascriptado.php	0.1 kB	2020-04-25 18:14:00	-rw-r--r--
	registro.php	1.0 kB	2020-05-14 01:48:00	-rw-r--r--

```
<?php
include 'conexion.php';

$usuario = $_POST["user"];
$contra = $_POST["pass"];
$descripcion = $_POST["descripcion"];
$idcomentario = $_POST["idcomentario"];

if(strlen($usuario) != 0 && strlen($contra) != 0){
    //Comprobamos el usuario actual
    $consulta = mysqli_prepare($conexion,"SELECT email FROM usuario WHERE nombre_usuario=? AND `password` = ?");

    mysqli_stmt_bind_param($consulta,"ss", $usuario, $contra);
    mysqli_stmt_execute($consulta);
    mysqli_stmt_store_result($consulta);
    mysqli_stmt_bind_result($consulta, $correo);

    $respuesta = array();
    if(mysqli_stmt_num_rows($consulta)>=1){
        $respuesta["mallogado"] = false;
        $consulta2 = mysqli_prepare($conexion,"SELECT comentario_id FROM reportecomentario WHERE nombre_usuario=? AND comentario_id=?");
        mysqli_stmt_bind_param($consulta2,"si", $usuario, $idcomentario);
        mysqli_stmt_execute($consulta2);
        mysqli_stmt_store_result($consulta2);

        if(mysqli_stmt_num_rows($consulta2)==0){
            $respuesta["yaExisteReporte"] = false;

            $consulta3 = mysqli_prepare($conexion,"INSERT INTO reportecomentario (nombre_usuario, comentario_id, descripcion) VALUES (?, ?, ?)");
            mysqli_stmt_bind_param($consulta3,"sis", $usuario, $idcomentario, $descripcion);

            if(mysqli_stmt_execute($consulta3)){
                $respuesta["success"] = true;
                $asunto = "Nero - Reporte de comentario";
                if(mysqli_stmt_fetch($consulta)){
                    mail($correo, $asunto, $descripcion);
                }
            } else {
                $respuesta["success"] = false;
            }
        } else {
            $respuesta["yaExisteReporte"] = true;
        }
    } else {
        $respuesta["mallogado"] = true;
    }
    echo json_encode($respuesta);
} else {
    echo "Entrada no permitida";
}
mysqli_close($conexion);
?>
```