

## DESCRIPCIÓN DEL PROYECTO

El proyecto consiste en una aplicación que recibe la información almacenada en una Base de Datos, donde se registra la información correspondiente de las Emergencias reportadas, desde su ubicación, hasta los heridos de la misma. La aplicación además de mostrar la información, permite la creación de nuevos reportes que son almacenados en la base de datos.

Haciendo uso de un gestor de Servidor, la aplicación se conecta a la base de datos y permite hacer llamadas para obtener y subir información, en este documento se verán las herramientas utilizadas para establecer la conexión con la base de datos así como modificar los datos enviados desde la aplicación.

## HERRAMIENTAS UTILIZADAS PARA LA BASE DE DATOS

- MySQL 8.0
- MySQL Workbench

## HERRAMIENTAS USADAS PARA LA CONEXIÓN AL SERVIDOR

- XAMPP (Servidor Local)
- Amazon Web Services - RDS (Instancia de Base de Datos - Servidor Web)
- Amazon Web Services - EC2 (Instancia de Servidor Web - Servidor Web)
- PuTTY (Transferencia de archivos PHP al servidor web)

## HERRAMIENTAS PARA EL MAPA

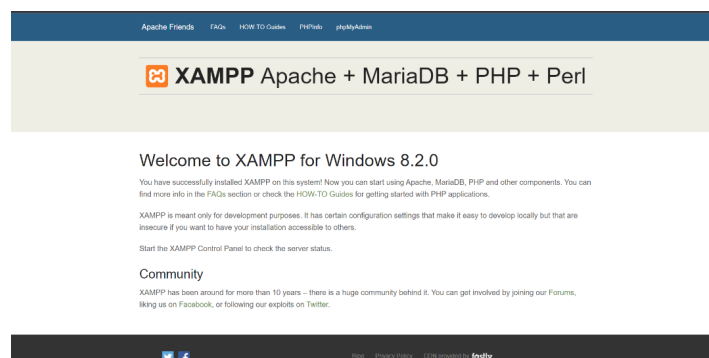
- Cuenta de Google Cloud Platform
  - Google Maps API (Permite el uso del mapa)
  - Geocode API (Permite obtener los nombres de las calles)
  - Directions API (Obtiene la ruta de un punto a otro)

## MANUAL TÉCNICO

### Creación de Servidor Local

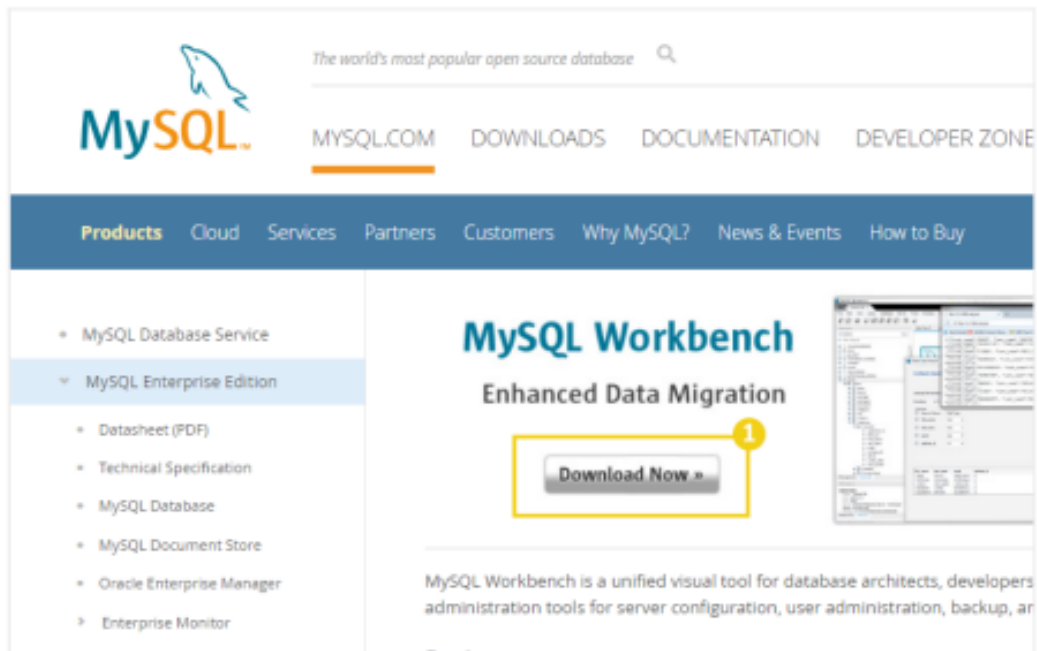
Primero se instala la aplicación de [XAMPP](#) donde se prepara la base de datos como también el servidor local que almacena la base de datos.

Una vez instalada la aplicación, se inician los módulos de Apache y MySQL para controlar el servidor local y la base de datos respectivamente, para verificar la correcta activación del servidor ingresando la dirección IP del Localhost nos aparecerá por default esta pantalla.

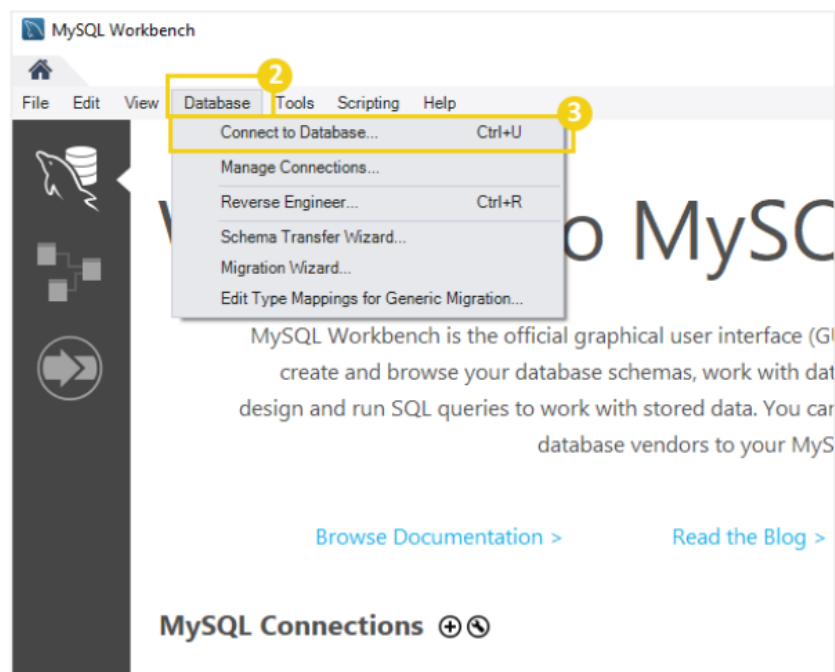


## Conexión a Base de datos

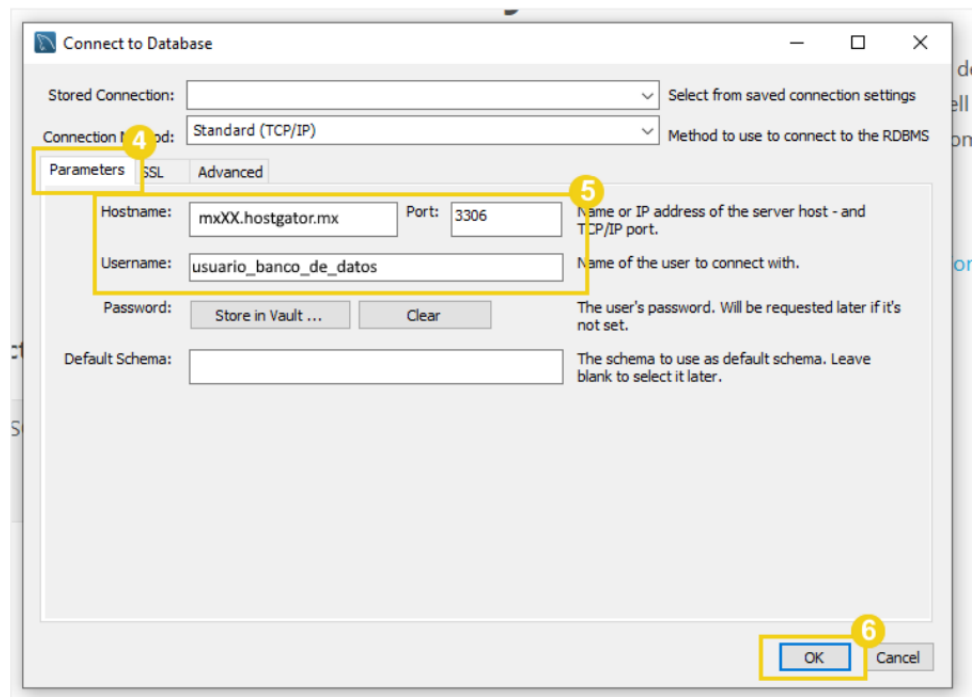
1. Procedemos a instalar la plataforma [MySQL Workbench](#) con un botón en la página oficial que nos permitirá la descarga.



2. Luego de abrir MySQL Workbench, en el menú superior, haga clic en la opción "Database"
3. Seguido, en la opción de "Connect to Database"



4. Una vez abierta la pestaña, se rellenan los siguientes campos con la información correspondiente.
5. Campos:
  - *Hostname*: Nombre del servidor (Si se usa un servidor local, se ingresa la **IP**)
  - *Port*: Se usa el estándar **3306**
  - *Username*: Nombre del usuario de la base de datos (Si se ingresa el usuario por defecto, escriba **root**)
6. Después de haber ingresado los datos, haga clic en “OK”



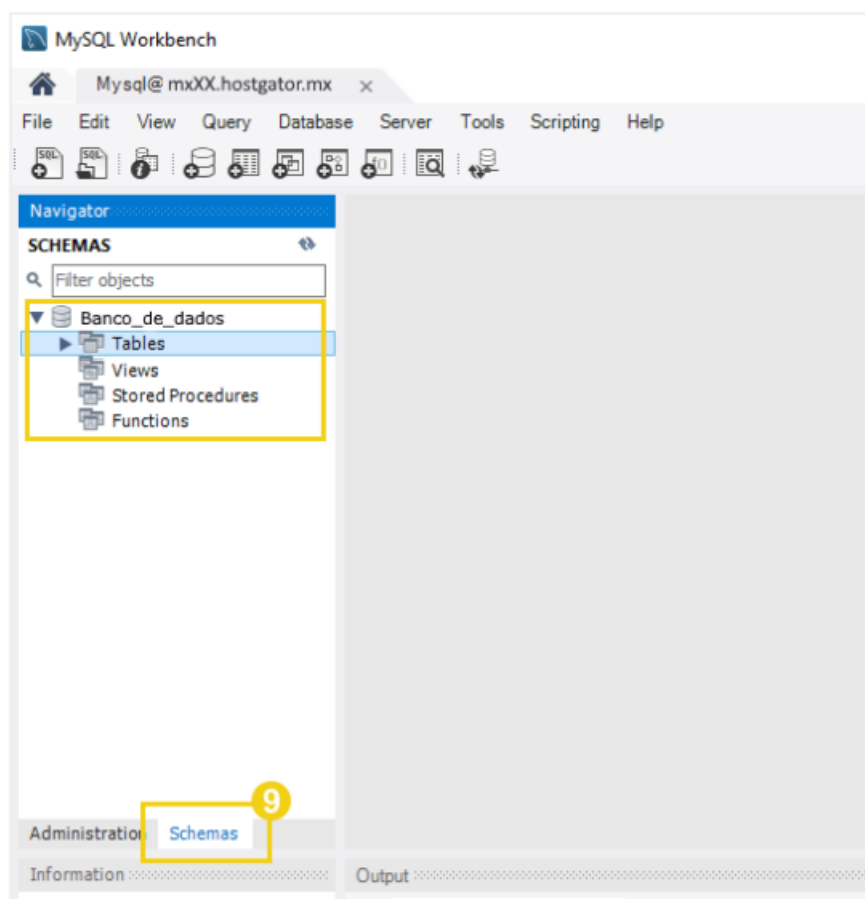
7. Al dar clic en “OK” aparecerá otra ventana en donde deberá introducir la contraseña de su base de datos (En caso de ser el usuario por defecto, dejar el espacio en blanco).

**NOTA:** En caso de ingresar con un usuario diferente, los valores deben de reasignarse en los correspondientes archivos PHP.

8. Hacer clic en “OK” luego de ingresar la contraseña



9. Una vez se haya verificado la conexión, se abrirá la ventana del editor, de lado izquierdo aparecerá el “Navigator”, en la parte inferior aparece la pestaña de ‘Schemas’ donde podrá observar todas las bases de datos creadas dentro del servidor.

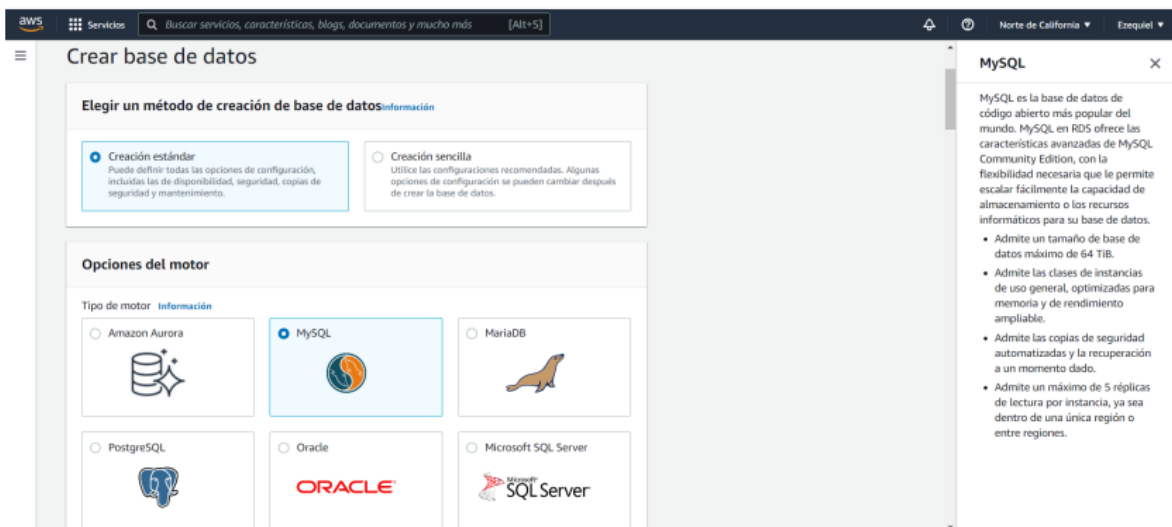


### Creacion de Servidor Web

Con el fin de mejorar el acceso al servidor de la aplicación y haciendo uso de las herramientas proporcionadas por Amazon Web Services (AWS), se creó una instancia de un servidor web que permitirá el acceso desde la nube para poder ingresar sin necesidad de pertenecer a la misma red local.

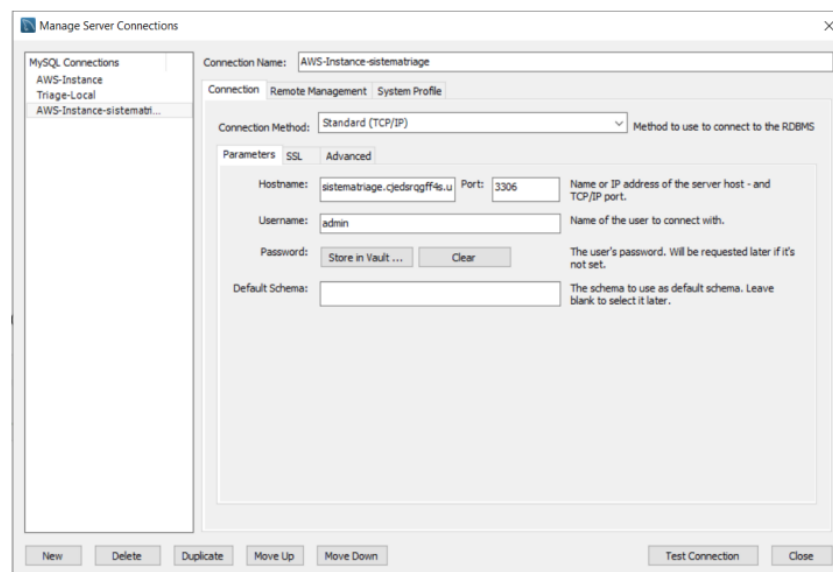
Dentro de la plataforma de AWS se haría uso de dos servicios: El servicio de Bases de Datos Relacionales (RDS) y el servicio Amazon Elastic Compute Cloud (EC2), los cuales trabajan en conjunto para gestionar la base de datos y el servidor en la nube respectivamente.

Para empezar se debe crear la instancia de base de datos, buscamos el servicio RDS en el panel de control de AWS, se elige el método de creación estándar de base de datos al igual que la capa gratuita y para el motor de la base de datos, elegimos MySQL.



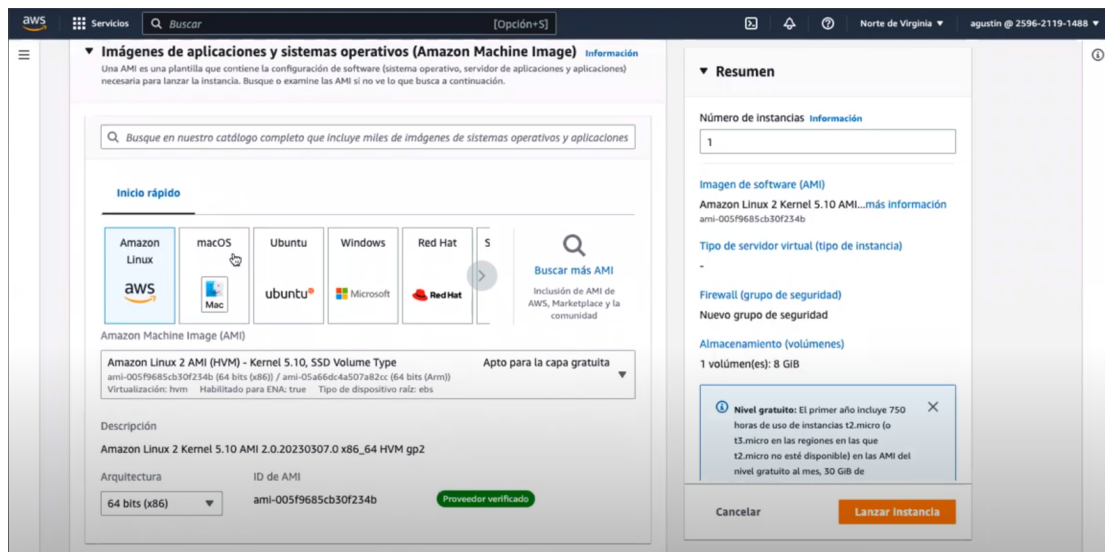
Para conectar esta instancia con MySQL Workbench, se realiza el mismo proceso que con el Servidor Local, con la diferencia de que en este caso usaremos el Hostname o dirección IP de la instancia RDS así como el usuario y contraseña creado para la misma.

**NOTA:** En caso de haber problemas con la conexión de la instancia, revisar la configuración de seguridad para permitir el tráfico a la base de datos.



Una vez hecha la instancia de la base de datos, se crea el servidor en la nube con EC2, este servidor permitirá instalar las herramientas y archivos necesarios para comunicar la aplicación con la base de datos, aquí es donde se almacenarán los archivos PHP que realizan las consultas y retornan la información que la aplicación necesite.

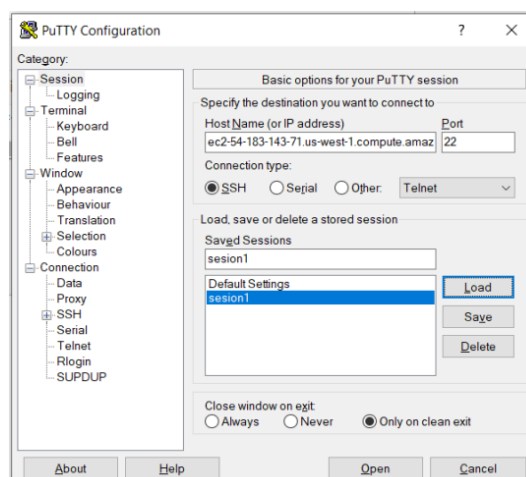
Lo primero que debemos hacer es elegir el sistema operativo que tendrá nuestro servidor, para este caso, se optó por Amazon Linux 2, que forma parte de los Kernel más recientes de Linux que Amazon proporciona, toda la demás configuración fue realizada de manera predeterminada con la capa gratuita.



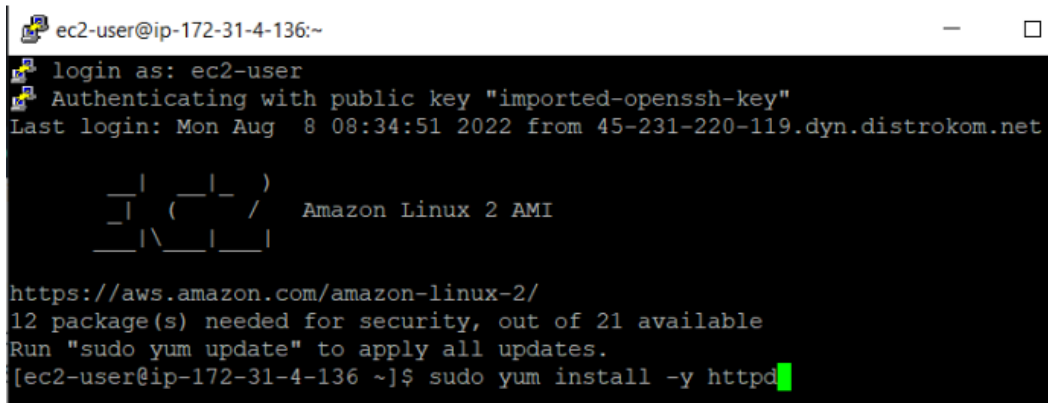
Selección del sistema operativo utilizado en el servidor virtual EC2 dentro de la plataforma de AWS

Una vez creada la instancia del servidor, se hace uso de la herramienta de PuTTY para poder instalar las librerías necesarias para que el servidor pueda conectarse con nuestra base de datos, esta herramienta también permite transferir los archivos PHP que realizan las consultas a la base de datos.

Con una llave SSH (Creada en paralelo con la instancia del servidor), creamos una sesión dentro de la herramienta de PuTTY, ingresando el Hostname o IP del servidor para poder acceder a nuestra instancia.



Al conectarse correctamente, aparecerá una terminal de nuestra instancia donde se debe iniciar sesión, luego de esto, se prepara el servidor para instalar las herramientas que se necesitarán, desde la línea de comandos instalaremos los paquetes más recientes de PHP, APACHE y MariaDB los cuales deberán ser configurados para responder a la instancia de base de datos.



```
ec2-user@ip-172-31-4-136:~  
login as: ec2-user  
Authenticating with public key "imported-openssh-key"  
Last login: Mon Aug  8 08:34:51 2022 from 45-231-220-119.dyn.distrokom.net  
  
  _ | _ | _ )  
  _ | ( _ - /  Amazon Linux 2 AMI  
  _ | \ _ | _ |  
  
https://aws.amazon.com/amazon-linux-2/  
12 package(s) needed for security, out of 21 available  
Run "sudo yum update" to apply all updates.  
[ec2-user@ip-172-31-4-136 ~]$ sudo yum install -y httpd
```

Una vez instalados los paquetes dentro del servidor, se implementan los archivos PHP dentro del mismo, si la instalación y configuración del servidor fue correcta, al ingresar al Hostname y realizar la llamada del archivo, se retorna el resultado correspondiente de la base de datos.

### Conexión de la aplicación

Con el fin de mejorar la accesibilidad y configuración del servidor dentro de la aplicación, esta cuenta con una clase llamada *“ServerURL.java”* misma que almacena una variable estática que sirve como la URL universal de todas las llamadas al servidor, todos los métodos de la aplicación hacen uso de esta variable, lo que permite cambiar fácilmente la URL sin tener que modificar a cada uno de los métodos.

**NOTA:** En caso de querer agregar un método nuevo que haga referencia al servidor, se recomienda usar esta variable.



```
package com.example.registroincidentes;  
  
public class ServerURL {  
  
    //Aqui se administra la direccion URL del servidor, cada clase accede al php que necesita  
    public static String url = "http://192.168.1.67/registroIncidentes/";  
    //public static String url = "http://192.168.0.123//registroIncidentes/";  
}
```

Clase **ServerURL** que almacena la dirección del servidor

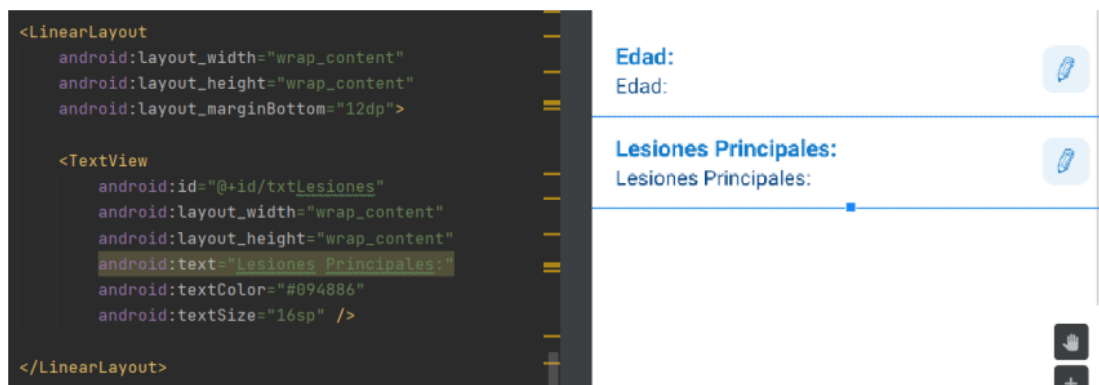
### Adición de más datos para capturar

Dentro de la información tenemos dos pantallas que muestran la información:

- *EmDetailsActivity*
- *InjuredProfileActivity*

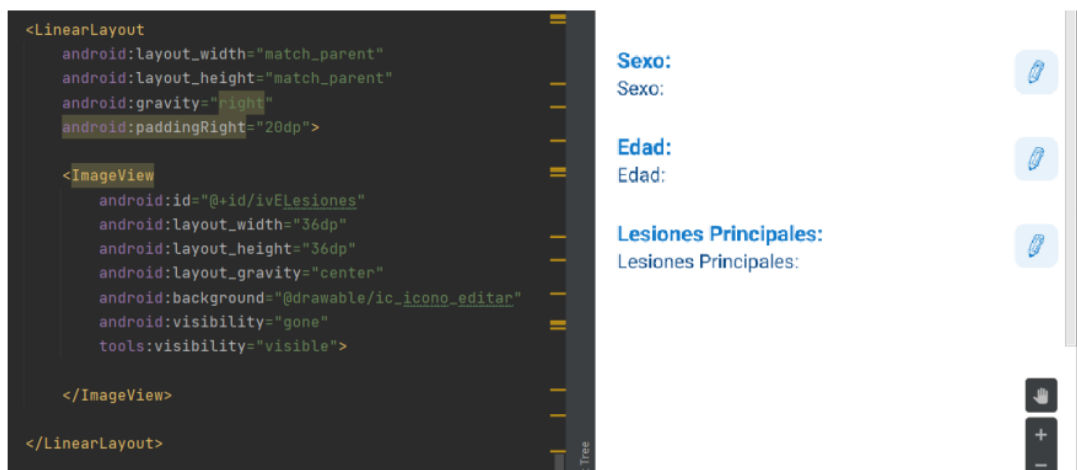
Estos Activity contienen varios fragment que muestran la información completa tanto de las emergencias como los heridos de cada una de estas, siendo *DetailsFragment* el que se encarga de esto en *EmDetailsActivity* y *PersonalDataFragment* como *TriageFragment* en *InjuredProfileActivity*.

Cada uno de estos fragments almacena la información correspondiente de las emergencias y heridos. No obstante, antes de poder crear las variables de estos en sus respectivas clases, los componentes deben ser declarados en sus correspondientes archivos *xml*.



Código de los campos y su representación gráfica.

Además del Textview que almacena la información, se declara un respectivo botón, mismo que permite la edición por parte del usuario, dicho botón es declarado con un `ImageView` mismo que nos da la posibilidad de insertar un recurso gráfico personalizado.



Código para crear un ImageView

Una vez creados los elementos, dentro del método *onCreateView* de su fragment correspondiente se asigna una variable de su mismo tipo, donde se asignan los valores obtenidos por la llamada al servidor.



```
//Se hacer referencia a los elementos del archivo fragment_details.xml
ivEmName = (ImageView) view.findViewById(R.id.ivEditName);
ivEmLocation = (ImageView) view.findViewById(R.id.ivEditLocation);
ivRegDate = (ImageView) view.findViewById(R.id.ivEditDate);
ivEmType = (ImageView) view.findViewById(R.id.ivEditType);
ivClosedDate = (ImageView) view.findViewById(R.id.ivEditCierre);
ivEmSecType = (ImageView) view.findViewById(R.id.ivEditTypeSec);

tvEmName = (TextView) view.findViewById(R.id.txtEmergencyName);
tvEmLocation = (TextView) view.findViewById(R.id.txtEmLocation);
tvRegDate = (TextView) view.findViewById(R.id.txtRegisterDate);
tvEmType = (TextView) view.findViewById(R.id.txtEmType);
tvEmSecType = (TextView) view.findViewById(R.id.txtEmType2);
tvClosedDate = (TextView) view.findViewById(R.id.txtFechaCierre);
```

Dentro de la clase fragment, se declaran también los métodos correspondientes a la edición de la información, los cuales hacen uso de los *alertDialog* con un *editText* para ingresar los valores manualmente, una vez ingresado se queda almacenado en su respectiva variable.

```
1 usage
private void OverwriteEmNameDialog(){
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    builder.setTitle("Nombre de Emergencia");
    final EditText input = new EditText(getActivity());
    input.setInputType(InputType.TYPE_CLASS_TEXT);
    input.setHint("Escribe el nombre de la Emergencia");
    builder.setView(input);
    builder.setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            EmergencyDetailsActivity.isModified = true;
            name = input.getText().toString().trim();
            tvEmName.setText(name);
        }
    });
    builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(getActivity(), "Cancelado", Toast.LENGTH_SHORT).show();
        }
    });
    builder.create().show();
}
```

Otra opción que se presenta para no introducir los datos de manera manual, es a través de definir una lista de opciones predeterminadas, las cuales al ser seleccionadas por el usuario son asignadas automáticamente.

```

alertOpciones.setItems(opciones, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        if (opciones[i].equals("Incendio")){
            EmergencyDetailsActivity.isModified = true;
            tvEmType.setText("Incendio");
            //PerfilHerido.cambio = true;
        }
        else if (opciones[i].equals("Choque")) {
            EmergencyDetailsActivity.isModified = true;
            tvEmType.setText("Choque");
            //PerfilHerido.cambio = true;
        }
        else if (opciones[i].equals("Derrumbe")){
            EmergencyDetailsActivity.isModified = true;
            tvEmType.setText("Derrumbe");
            //PerfilHerido.cambio = true;
        }
    }
});
alertOpciones.show();

```

Luego de que los valores de los `textViews` son modificados en cualquiera de los fragment correspondientes, mediante un botón flotante permite guardar los datos modificados para ser enviados a la base de datos.

Los valores son enviados en una cadena de caracteres determinada la cual se envía junto con la petición del servidor a la base de datos, si este responde correctamente, los datos se modifican en la base de datos.

```

// se crea un HashMap y se añaden los valores con el método put, con la forma ("Clave", valor);
@Override
protected Map<String, String> getParams() throws AuthFailureError {

    String idEm = EmergencyDetailsActivity.idEmergency;
    String emName = tvEmName.getText().toString();
    String emType1 = tvEmType.getText().toString();
    String emType2 = tvEmSecType.getText().toString();

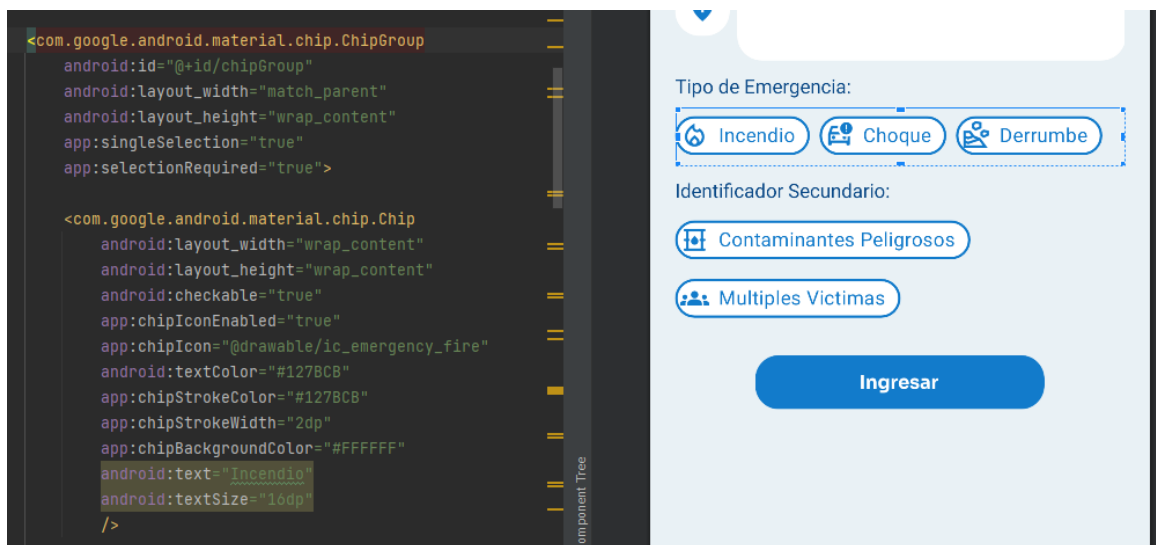
    Map<String, String> parametros = new HashMap<>();
    parametros.put("idEmergency", idEm);
    parametros.put("em_Name", emName);
    parametros.put("em_Type1", emType1);
    parametros.put("em_Type2", emType2);
    return parametros;
}

```

## Implementación de nuevos tipos de emergencias e iconos

La aplicación cuenta con identificadores de emergencia que permiten una guía visual sobre cuál es la emergencia, por lo que, en caso de querer crear más emergencias, se deben implementar de manera manual en la aplicación, así como los iconos correspondientes.

Para poder agregar más identificadores de emergencias, lo primero que debemos hacer es agregarlos dentro del layout *activity\_emergency\_report.xml*, dentro de este se encuentran dos *ChipGroups* que contienen tanto el tipo de emergencia como los identificadores secundarios. En caso de querer agregar más identificadores, se debe agregar un *Chip* dentro de su respectivo *ChipGroup*.



Código xml del *ChipGroup* y *Chip*, y su representación gráfica

Una vez agregados al xml, si estos son agregados dentro de su *ChipGroup* correspondiente, la aplicación tomará el texto de estos y de manera automática guarda la información al ser seleccionado, por lo que no hay necesidad de definirlo manualmente en la clase java.

**NOTA:** En caso de querer agregar un icono nuevo ver *página 13*.

## Cambiar Información mostrada

En caso de querer agregar más información dentro de las pestañas de emergencias, heridos y personal, cada uno de estos elementos cuenta con su propio layout xml, mismo que puede ser identificado con la terminación *“\_card.xml”*.

Toda la información que se recibe de estos objetos es almacenada en una clase objeto correspondiente, la cual permite que pueda ser accedida por cualquier activity de la aplicación. Cada objeto tiene el mismo nombre que su correspondiente tabla de la base de datos.

```

public class Emergency {

    2 usages
    private Integer idEmergency;
    2 usages
    private String Name;
    2 usages
    private String Location;
    2 usages
    private String RegisterDate;
    2 usages
    private Double Latitude;
    2 usages
    private Double Longitude;
    2 usages
    private Double Altitude;
    2 usages
    private String EmergencyType;

    2 usages
    private String EmergencyType2;
    2 usages
    private Boolean IsClosed;
    2 usages
    private String ClosedDate;
    2 usages
    private String EmergencyCol;
}

```

Clase objeto Emergency

Para que el objeto pueda recibir la información de cada elemento obtenido a través de la base de datos, se prepara la clase Adapter, la cual permite la asignación de la información a su respectivo archivo “\_card.xml”, este archivo inflar el card para generar un objeto por cada emergencia, herido o personal dentro de la aplicación.

Primero dentro del método ViewHolder se declaran las variables públicas.

```

1 usage
public EmergencyViewHolder(View itemView) {
    super(itemView);
    personCardView = (LinearLayout) itemView.findViewById(R.id.emergency_card);
    icon1 = (ImageView) itemView.findViewById(R.id.imagen);
    icon2 = (ImageView) itemView.findViewById(R.id.imagen2);
    txtName= (TextView) itemView.findViewById(R.id.txtEmergencyName);
    txtLocation = (TextView) itemView.findViewById(R.id.txtEmergencyLocation);
    txtId = (TextView) itemView.findViewById(R.id.tvID);
    addStaff = (ImageView) itemView.findViewById(R.id.ivAddStaff);

    addStaff.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(activity, UsersListActivity.class);
            intent.putExtra( name: "idEmergency", txtId.getText());
            activity.startActivity(intent);
        }
    });
}
}

```

Dentro del método *onBindViewHolder* se asignan los valores de la lista a cada uno de los elementos correspondientes.

```
@Override
public void onBindViewHolder(EmergencyViewHolder holder, final int position) {
    holder.txtName.setText(EmergenciesList.get(position).getName());
    holder.txtLocation.setText(String.valueOf(EmergenciesList.get(position).getLocation()));
    holder.txtId.setText(String.valueOf(EmergenciesList.get(position).getIdEmergency()));
    setEmergencyIcon(holder.icon1, String.valueOf(EmergenciesList.get(position).getEmergencyType()));
    setEmergencyIcon(holder.icon2, String.valueOf(EmergenciesList.get(position).getEmergencyType2()));
}
```

Para poder recibir los datos a través de la consulta, una vez la aplicación llama al servidor, este responde con una cadena JSON donde envía todos los parámetros obtenidos, el trabajo de la aplicación es asignar estos valores de manera individual con los códigos de cada dato.

```
emergency = new Emergency();
jsonObject=null;
jsonObject=json.getJSONObject(i);

emergency.setIdEmergency(jsonObject.optInt( name: "idEmergency"));
emergency.setName(jsonObject.optString( name: "em_Name"));
emergency.setLocation(jsonObject.optString( name: "em_Location"));
emergency.setRegisterDate(jsonObject.optString( name: "em_RegDate"));
emergency.setLatitude(jsonObject.optDouble( name: "em_Lat"));
emergency.setLongitude(jsonObject.optDouble( name: "em_Lon"));
emergency.setAltitude(jsonObject.optDouble( name: "em_Alt"));
emergency.setEmergencyType(jsonObject.optString( name: "em_Type"));
emergency.setEmergencyType2(jsonObject.optString( name: "em_Type2"));
emergencies.add(emergency); // Cada objeto se añade a la listaHeridos
```

### Agregar iconos a las emergencias

En el caso específico de las emergencias, debido a que estas cuentan con un icono para identificarlas, el adapter cuenta con un método extra que permite asignar un icono tomando el valor obtenido por el tipo de emergencia que retorne la base de datos, por lo que para poder agregar un nuevo icono debe ser agregado desde el método *setEmergencyIcon*.

```

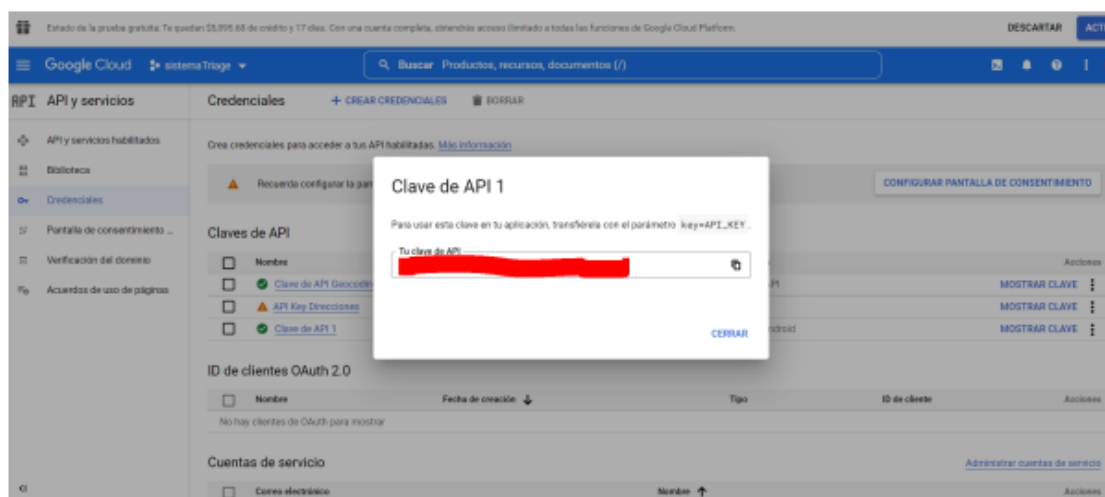
@
public void setEmergencyIcon(ImageView icon, String em_type){
    switch (em_type){
        case "Incendio":
            icon.setImageResource(R.drawable.ic_emergency_fire);
            break;
        case "Choque":
            icon.setImageResource(R.drawable.ic_emergency_car_crash);
            break;
        case "Derrumbe":
            icon.setImageResource(R.drawable.ic_emergency_landslide);
            break;
        case "Contaminantes Peligrosos":
            icon.setImageResource(R.drawable.ic_emergency_toxic);
            break;
        case "Multiples Victimas":
            icon.setImageResource(R.drawable.ic_emergency_multiple_victims);
            break;
        default:
            icon.setVisibility(View.GONE);
            break;
    }
}

```

### Configurar la Clave de la API de Google Maps

Luego de crear una cuenta en la Plataforma de Google Cloud, se crea una clave para poder hacer consultas con la API de Google Maps. La clave puede crearse y ser copiada desde la pestaña de Credenciales del menú izquierdo.

**NOTA:** Para que todas las funciones se ejecuten correctamente, se deben activar los servicios mencionados anteriormente en los requisitos de la aplicación.



Esta llave es utilizada para generar el mapa dentro del Fragment del mapa, al igual que en los activity EmReportActivity y RegisterInjuredActivity en donde se usa para obtener los nombres de las calles a través de las coordenadas.

```

HttpGet httpGet = new HttpGet(
    uri: "https://maps.googleapis.com/maps/api/geocode/json?latlng="
        + lat + ", " + lng + "&sensor=true&key=AIzaSyDTN-fVZkUS5FD0cVWwMVuJAI09xF5f19A"); // clave de acceso a la API
HttpClient client = new DefaultHttpClient();
HttpResponse response;
StringBuilder stringBuilder = new StringBuilder();

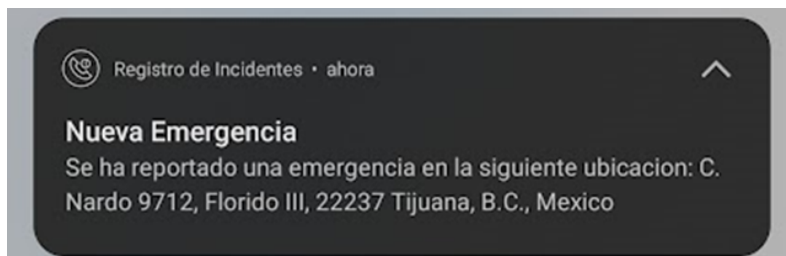
```

Ejemplo de uso en el método getLocationInfo de EmReportActivity

Se procede a sustituir la llave por la llave propia para su correcto funcionamiento.

## Configuración de Notificaciones Push

Al reportar una emergencia dentro de la aplicación, se crea de la misma forma una notificación que aparece en la barra de tareas.



Esta se encuentra como un método dentro de la clase EmReportActivity, en caso de querer modificar su contenido, estas modificaciones se deben hacer en el método sendNotification dentro de la clase antes mencionada.

```

//Se definen los detalles Generales de la notificacion, como el texto, icono y titulo
//Bitmap iconoNotifica = BitmapFactory.decodeResource(context.getResources(), R.drawable.notifica_icon);
int iconoSmall = R.drawable.ic_noti_alert;
mBuilder.setSmallIcon(iconoSmall);
//mBuilder.setLargeIcon(iconoNotifica);
mBuilder.setTitle("Nueva Emergencia");
mBuilder.setText("Se ha reportado una emergencia");
mBuilder.setStyle(new NotificationCompat.BigTextStyle().bigText(message));
mBuilder.setContentIntent(pendingIntent);
mBuilder.setChannelId(channelID);
notificationManager.notify(id: 1, mBuilder.build());

```

Configuración general de la Notificación

Al dar clic en la notificación, ésta envía al usuario a la pantalla de listas de emergencias de la aplicación, esto se hace creando un Intent primero, que luego es relacionado a un PendingIntent con una Flag que actualiza la aplicación. En caso de querer redireccionar a la notificación, esto se hace en el Intent que se encuentra en el método antes mencionado.