ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“DESARROLLO DE UN SISTEMA DE LOCALIZACIÓN PARA ESPOL BASADO EN DISPOSITIVOS MÓVILES”

INFORME DE MATERIA INTEGRADORA

Previa a la obtención del Título de:

INGENIERO EN TELEMÁTICA

EMILIO JOSÉ PIEDRAHITA ICAZA

BRYAN LEONEL BASANTES ESPARZA

GUAYAQUIL-ECUADOR

AÑO: 2017

AGRADECIMIENTOS

Agradezco a mi familia por siempre ser el pilar de mis esfuerzos, su apoyo incondicional siempre fue importante en mi desarrollo académico, a mis amigos con los que siempre cuando fueron necesarios me brindaron una mano. A mis profesores que inculcaron el conocimiento necesario para poder desarrollar este proyecto.

Emilio Piedrahita

Agradezco en primer lugar a Dios por haberme dado salud y haberme protegido en todo momento y por tener la dicha de estar con mi hermosa familia. A mi madre que falleció el año pasado y es a quien amo mucho con todo mi corazón, aunque ya no esté presente, está en mi corazón. A mi hermano que siempre está ahí para darme ánimos y a mi padre por ser incondicional conmigo.

De manera especial agradezco a la ESPOL por abrirme las puertas y a mi estimado Tutor, el ing. Benjamín Flament y nuestro guía durante todo este proceso para poder finalizar esta presente tesis.

Bryan Basantes

TRIBUNAL DE EVALUACIÓN

Ing. Néstor Arreaga	**Ing. Benjamín Flament**
PROFESOR EVALUADOR	PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

“La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual”

.....

Bryan
Basantes

.....

Emilio
Piedra
hita

RESUMEN

ESPOL MAPS es el sistema localización que nosotros implementaremos y su finalidad es llevar al usuario a su destino dentro de la ESPOL. El destino puede ser un bloque, aula, laboratorio, oficina, auditorio, etc.

Los tiempos que le toma al usuario llegar a su destino serán calculados según la ubicación actual, la distancia del camino o rutas sugeridas por nuestro ESPOL MAPS y el medio de transporte que elija el usuario (A pie o en auto).

Estos valores serán obtenidos a medida que el usuario haga el recorrido sugerido por el ESPOL MAPS. Al mostrar inicialmente el tiempo que tomará realizar la ruta y la hora registrada en el celular, se podrá determinar a qué hora aproximadamente llegará el usuario al destino.

El ESPOL MAPS ayudara a la persona a ser puntual y encontrar cualquier destino dentro de la ESPOL.

ÍNDICE GENERAL

CAPÍTULO 1

1. MARCO GENERAL

Las universidades del mundo más famosas y especializadas, a la vez de dar una educación y preparación de excelencia, ofrecen una infraestructura y áreas extensas de terreno para una mejor interacción de las personas que estudian y vistan, con su entorno. Tienen diferentes Facultades, oficinas administrativas, salas de conferencias, complejos deportivos, etc.

Ecuador cuenta con la Espol. Ubicada en las primeras posiciones en rankings de mejores universidades del país, demostrando que esta Universidad da una educación de excelencia siendo categoría A [1].

El Campus Gustavo Galindo abarca 690 hectáreas, de las cuales 40 están urbanizadas, 40 se utilizarán para expansión futura y 600 han sido declaradas bosque protector que la ESPOL reforestará como una muestra de su preocupación por la naturaleza [2].

La Espol ofrece conferencias, eventos, postgrado de alto nivel, etc. Existen diferentes eventos que se desarrollan en la Universidad además de las clases repartidas para estudiantes de pregrado lo cual hace que diferentes clases de personas asistan a la Universidad.

1.1 Descripción del problema

Un problema muy común presentado a los estudiantes de la ESPOL, mayormente novatos, visitantes, etc. Es el desconocer la ubicación de los sitios a los que debe asistir, tales como aulas y/o laboratorios, oficinas administrativas, sala de eventos, etc.

A la vez el tamaño de la universidad da como resultados el tener las

aulas distantes a la ubicación actual de la persona, lo que produciría un atraso a la asistencia de las mismas o llegar tarde a la hora de las lecciones u entrega de proyectos.

Algunas oficinas administrativas son desconocidas por los estudiantes, por lo que la desorientación es un problema muy común y una manera mucho más fácil de poder conocer el lugar, donde quieren llegar es con la aplicación de Espol Maps. Una Aplicación capaz de conocer su ubicación actual y dirigirlo a su destino.

1.2 Justificación

En la actualidad la mayoría de empresas, profesores o autoridades requieren de su personal, estudiantes o corresponsales puntualidad a los diferentes eventos, clases o exposiciones que se vayan a impartir. La Espol, al tener un tamaño significativo tiene diferentes caminos para llegar a un destino.

Para este proyecto se procederá a ofrecer el recorrido, y el tiempo que le tomara llegar al destino se lo mostrará a la persona que use la aplicación.

La importancia de la Aplicación es que los estudiantes puedan llegar con seguridad a su lugar destino a tiempo y no se pierdan al llegar a su ubicación.

.

1.3 Objetivos

1.3.1 Objetivo General

Disminuir el tiempo que le toma a una persona llegar a un bloque, aula, oficina, laboratorio, etc. Dependiendo de la ubicación actual del usuario y la ubicación del destino dentro de la ESPOL.

1.3.2 Objetivos Específicos

- Tomar las diferentes coordenadas GPS de los distintos Bloques y edificios administrativos de la ESPOL.
- Formar Base de Datos con las coordenadas registradas de bloques, aulas, oficinas, etc.
- Realizar la aplicación de navegación GPS y la programación para la elección de la ruta origen y destino medida en km o metros dentro de la Espol.

1.4 Resultados esperados

Inicialmente se tomarán muestras estadísticas para determinar cuánto un alumno demora en llegar a un lugar desconocido sin hacer uso de nuestro sistema. Así mismo se tomarán muestras del tiempo en que demoran en llegar a su destino usando el sistema. Como resultado se espera que el tiempo promedio mejore considerablemente.

CAPÍTULO 2

2. DISEÑO Y DESARROLLO

En esta sección se describe el desarrollo y creación de la aplicación ESPOL MAPS y la base de datos que trabaja tanto con la aplicación como con la pagina WEB.

2.1 Metodología.

El proyecto está dividido en las siguientes actividades:

- Recopilación de información correspondiente a coordenadas, bloques, aulas, oficinas, Laboratorios, fotos, etc.
- Almacenar información en servidor, y levantar servicio Web en el mismo.
- Desarrollo de la App ESPOLMAPS.
- Desarrollo de sitio Web para actualizar cualquier dato subido a la base de datos del Servidor.

2.2 Herramientas de Hardware.

Haciendo el estudio del Proyecto que se está realizando hemos determinado que el Hardware que se usó es el siguiente:

- Laptop Personal
- Dispositivo Móvil
- Servidor

2.2.1 Laptop Personal

Se requiere un ordenador que soporte los programas a utilizar, virtualización y conexiones VPN. Los requisitos mínimos que para un desarrollo óptimo fueron: [3]

- 2 GB de memoria RAM.
- 2 GB de espacio libre en Disco
- Resolución mínima de 1280x800 para virtualización en Android Studio.
- Java 8
- Procesador Intel.

2.2.2 Dispositivo Móvil

En este Proyecto hemos decidido que la App a desarrollar será para dispositivos Android. Este dispositivo nos ayudó a realizar pruebas de campo para comprobar su funcionalidad. Las características del móvil que se ha utilizado son las siguientes:

- Versión Android 4.1 (API 16) en adelante.
- Debe tener GPS (Obligatorio).
- Conexión a internet (Por Wifi o Datos Móviles).

2.2.3 Servidor

Hemos analizado las formas de almacenamiento, la actualización de datos y el consumo de recursos del dispositivo. Hemos determinado 2 formas de almacenamiento:

- De manera Local.
- Por medio de un Servidor.

Ambas opciones tienen ventajas y desventajas, pero la ventaja que es más relevante es la forma en la que se actualizarán los datos. El servidor es la herramienta que permitirá actualizar los datos sin tener que actualizar la App en sí y será escalable para la parte administrativa de cada facultad.

Al utilizar el servidor, mediante un sitio Web, la parte administrativa de cada facultad, podrá modificar de manera

sencilla, la información que necesitará la App mostrar a los usuarios y no será necesario que solo los desarrolladores modifiquen dicha información.

Los requerimientos del servidor a utilizar son los siguientes:

- S.O Linux Distribución Ubuntu 14.06
- 80 GB de Disco Duro
- 512 MB de RAM

2.3 Herramientas de Software.

2.3.1 Android Studio

Android Studio es la herramienta fundamental del proyecto. Es el IDE oficial de Android, herramienta que facilita el diseño y desarrollo de Aplicaciones de buena calidad.

Ofrece distintas herramientas de depuración, pruebas, edición, etc. El lenguaje de programación base del IDE es JAVA, aunque también es compatible con lenguaje C, C++ y NDK.

La IDE cuenta con un Emulador para poder probar las aplicaciones creadas desde tu mismo ordenador. Se verifica su funcionabilidad y errores a corregir [4].

2.3.2 MYSQL

Hemos optado por la funcionabilidad de almacenar la información mediante un servidor, por lo que tenemos que elegir un motor de base de datos. El motor de base de datos que hemos elegido es MYSQL por el conocimiento que hemos adquirido a lo largo de nuestra carrera.

Muchas empresas de gran nivel y de mayor crecimiento del mundo como son Facebook, Google, Adobe, etc. Confían en MYSQL [5].

2.3.3 PuTTY

El servidor que usaremos se encuentra en la Espol, por lo que requerimos una comunicación remota al mismo. Para eso optamos con la herramienta PuTTY.

PuTTY es un cliente SSH y telnet para plataformas Windows desarrollado por Simon Tatham y es un software de código abierto [6].

2.3.4 Cisco System AnyConnect Mobility Client

Para conectarnos a un servidor de la Espol debemos conectarnos a la red privada de la Universidad por medio de una VPN.

Para poder conectarnos a la VPN de la Espol es necesaria la herramienta **Cisco System AnyConnect Mobility Client.**

2.4 Desarrollo del Sistema.

2.4.1 Recopilación de información

En lo que se refiere a recolección de datos e información, se recorrió los diferentes lugares de la Espol tomando datos como:

- Facultades
- Bloques
- Aulas
- Oficinas
- Coordenadas

Los denominados bloques son los edificios de la universidad. Cada Bloque tiene una numeración o una identificación. No todos los Bloques tienen aulas, por lo que también se recolecto la información de las oficinas dentro de esos bloques. También

se tomaron fotos correspondientes de la entrada a cada Bloque.

Las coordenadas tomadas son de los diferentes bloques de la universidad. La aplicación no determina el aula u oficina, sino los bloques en donde se encuentran. La forma de identificar donde están las aulas u oficinas fue dada por una pequeña descripción del piso del bloque y en la dirección con respecto a la entrada al bloque.

Nuestra aplicación al estar basada en Google Maps tiene un inconveniente, que son la falta de caminos peatonales en la App de Google. Para esto hemos tomado la iniciativa de crear los caminos faltantes. Para eso tomamos las diferentes coordenadas de los caminos faltantes.

2.4.2 Desarrollo base de Datos MYSQL.

Se ha optado en el desarrollo de una Base de datos MySQL para el almacenamiento de información correspondiente a los datos requeridos para la aplicación. Para esto se han creado la Base de datos espolmaps dentro de un servidor proporcionado por la Espol y las siguientes tablas relacionadas como indica la **Figura 2.1**:

- Facultad.
- Bloque
- Piso
- Aula
- Profesores

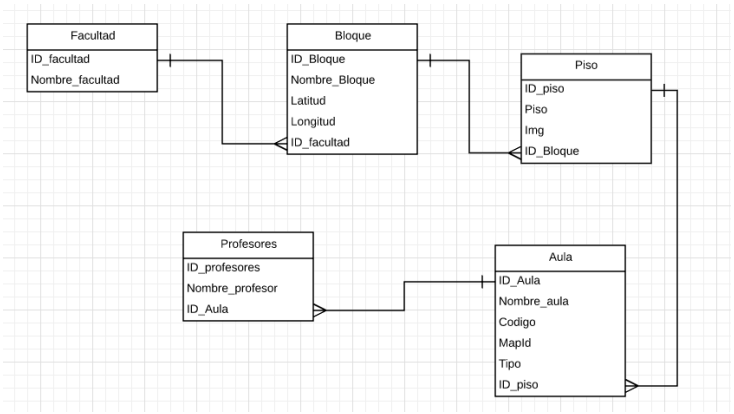

Figura 2.1: Diagrama de bloques de base de datos.

Se debe crear usuarios para cada facultad, con el motivo de que personal administrativos de las mismas puedan acceder a la base de datos y agregar información relevante como los profesores y en qué oficina se encuentran. Estos podrán modificar información por una página web.

Para la comunicación de nuestra App con la base de datos es fundamental levantar el servicio Web de nuestro servidor, y la instalación de paquetes para la codificación PHP, ya que se usará dicho lenguaje de programación para el envío de datos a nuestro dispositivo móvil.

Para la comunicación y configuración de nuestro servidor, al ser este perteneciente a la ESPOL, nos conectamos a la VPN de la ESPOL mediante el software **CISCO AnyConnect** proporcionado por la universidad.

Para ingresar entorno de línea de comando de nuestro servidor, estando conectado a la VPN, lo hicimos mediante SSH. Para esto utilizaremos la aplicación **PuTTy**.

Antes de descargar e instalar cualquier paquete debemos actualizar nuestro repositorio Linux de nuestro servidor. Para esto dentro de la línea de comando digitamos el siguiente

comando:

sudo apt-get update

Procedimos a descargar los paquetes mysql-server desde la línea de comandos con el siguiente comando:

sudo apt-get install mysql-server

Apareció una ventana para poner una contraseña al usuario root de MYSQL, tal y como indica la **Figura 2.2**, le colocamos la contraseña correspondiente

Figura 2.2: Configuración inicial MySQL.

Ingresamos al mysql con el siguiente comando:

mysql –u root –p

Al hacer eso ingresamos como usuario root dentro del mysql. Al terminar quedo listo para ingresar los diferentes comandos SQL en la línea de comandos como indica la **Figura 2.3**:

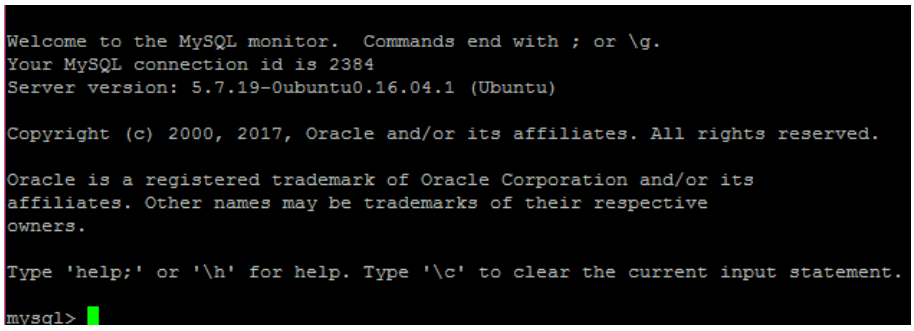

Figura 2.3: Ingreso a MySQL.

Creamos un usuario que tenga todos los privilegios con el siguiente código mysql:

CREATE USER 'nombre_usuario'@'localhost' IDENTIFIED BY 'tu_contrasena';

GRANT ALL PRIVILEGES ON *.* TO 'nombre_usuario'@'localhost';

Creamos la base de datos espolmaps con el siguiente comando mysql:

CREATE DATABASE espolmaps;

Procedemos a crear las tablas, aquí un ejemplo de una tabla creada:

CREATE TABLE Facultad (

ID_facultad int primary key auto incremental,

Nombre_facultad varchar (100));

La tabla fue creada como indica la **Figura 2.4.**

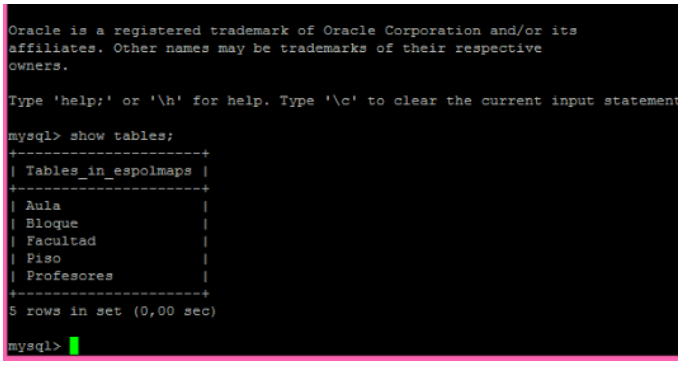

Figura 2.4: Creación de Tablas.

Con el comando INSERT INTO llenamos las tablas con los datos recolectados:

INSERT INTO Facultad VALUES(1,'FIEC');

Con el commando *Select* se pudo ver los valores ingresados en la table como indica la **Figura 2.5**

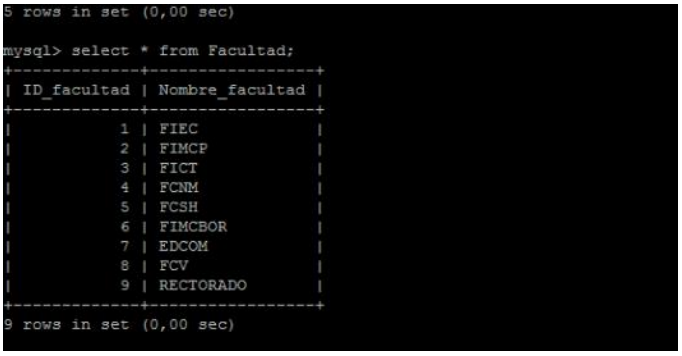

Figura 2.5: Ingreso de Datos en Tablas.

2.4.3 Levantamiento de Servidor Web.

Para nuestro servidor hemos optado por el levantamiento del servicio web APACHE2 al tener excelente compatibilidad a nuestro servidor con distribución UBUNTU (Linux).

Procedemos a descargar los paquetes de APACHE2, PHP y Phpmyadmin desde la línea de comandos con los siguientes comandos:

sudo apt-get install apache2
sudo apt-get install libapache2-mod-auth-mysql php5-mysql phpmyadmin

Al instalar Phpmyadmin nos salió una ventana para saber en qué servidor deseamos configurar el mismo. Elegimos apache2 que es el servidor que hemos levantado. La ventana de configuración se puede observar en la **Figura 2.6 y 2.7**:

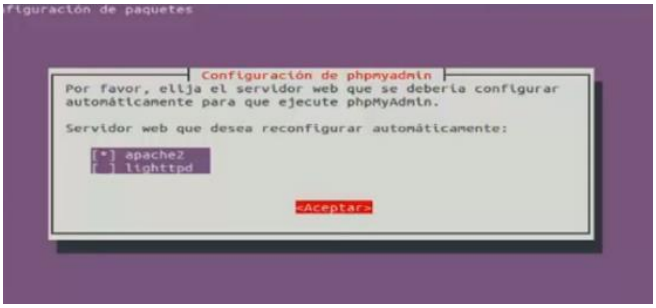

Figura 2.6: Configuración Servidor PhpMyAdmin.

Después salió una ventana donde nos pidió la contraseña que agregamos a nuestro Mysql:

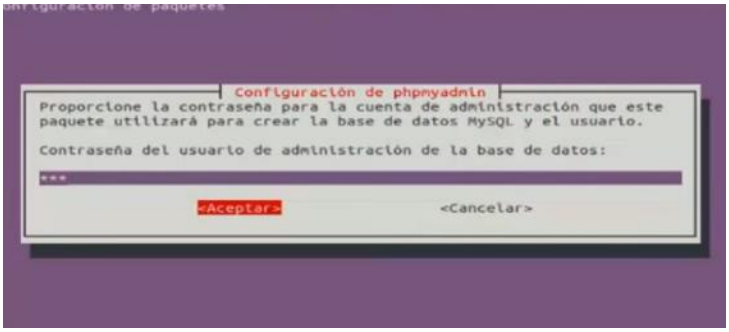

Figura 2.7: Configuración de enlace a base de Datos.

Con esto ya podemos acceder a nuestro servidor y a nuestra base de datos por el navegador. Se puede observar en la **Figura 2.8 y 2.9**

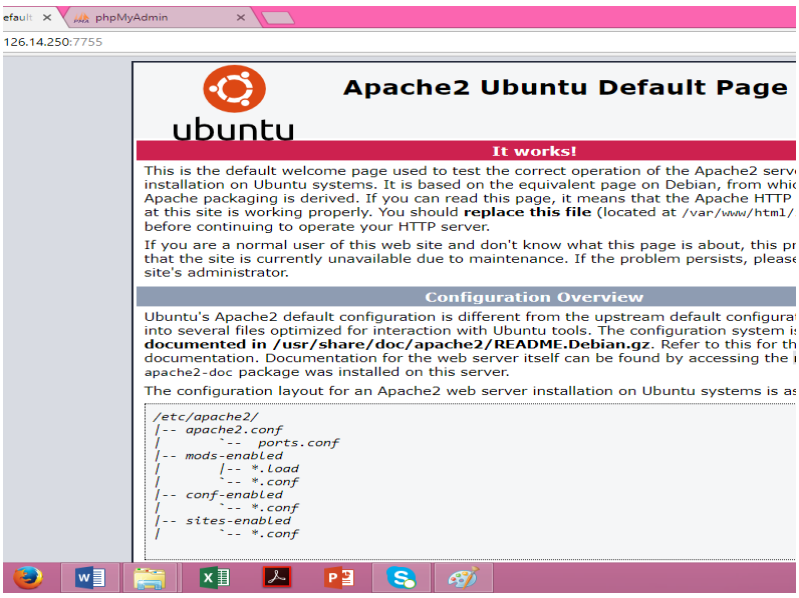

Figura 2.8: Comprobación de levantamiento de servidor Web.

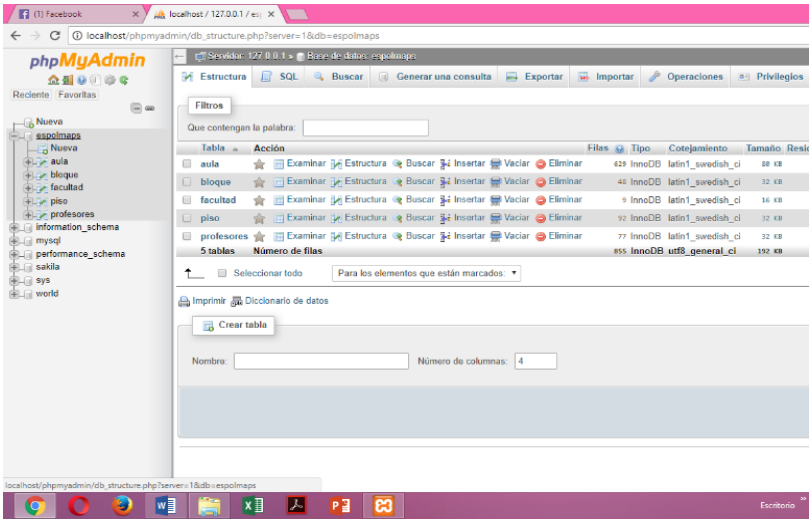

Figura 2.9: Funcionamiento de PhpMyAdmin.

2.4.4 Desarrollo de la APP.

Figura 2.10: Wireframe de la aplicación.

CODIFICACIÓN

La aplicación cuenta con 6 Activity o ventanas:

- Main
- Menu
- Menu_Block
- Busqueda
- Busqueda_Profesor
- Map

MAIN

Figura 2.11: Layout MainActivity

Cuenta con la función **onCreate** la cual es la que crea el Layout asociado a la **Activity Main**. Además de que indica la acción que realizará el botón con un ClickListener, el cual activa primero la función de la ProgressBar, **AsyncTask_load()**, y luego nos enviará a la **Activity Menu** mencionada anteriormente.

```
public class MainActivity extends AppCompatActivity {
    //Tiempo que demora la ProgressBar en llenarse completamente en milisegundos
    private static final long SPLASH_TTIME_OUT = 3000;
    //Se definenVariables de tipo ProgressBar y boton
    ProgressBar barra_horizontal;
    Button Aceptar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Se Instancian el ProgressBar y el boton.
        barra_horizontal = (ProgressBar) findViewById(R.id.progressBar);
        Aceptar = (Button) findViewById(R.id.Ingresar);
        //Se agrega un clickListener al boton
        Aceptar.setOnClickListener((view) → {
                //Llama a la funcion que llenara la progressBar
                new AsyncTask_load().execute();
                //Se agrega el Delay con duracion de 3000 milisegundos y se envía a la siguiente actividad
                new Handler().postDelayed(() → {
                        Intent home = new Intent(MainActivity.this, menu.class);
                        startActivity(home);
                        finish();
                }, SPLASH_TTIME_OUT);
        });
    }
```

Aquí se muestra la función **AsyncTask_load()** y la función **onBackPressed(),** esta última es iniciada cuando se presiona la tecla back del móvil, en este caso finaliza la App**:**

```
private class AsyncTask_load extends AsyncTask<Void, Integer, Void> {
    //se define un entero que representara el porcentaje de carga en la ProgressBar
    int progreso=0;

    @Override
    protected Void doInBackground(Void... params) {
        //Se hace un lazo que aumente nuestra variable entera simulando la carga de progressBar
        while (progreso < 100) {
            progreso++;
            publishProgress(progreso);
            //se hace una pausa de 20 milisegundos cada carga.
            SystemClock.sleep(20);
        }
        return null;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        //se agrega la carga a la progressBar
        barra_horizontal.setProgress(values[0]);
    }
}
//Si presionan el BackBoton finaliza la apl
public void onBackPressed() { finish(); }
```

MENU

Figura 2.12: Layout MenuActivity

Esta Activity nos mostrara las diferentes Facultades de la universidad y un icono que al presionarlo despliega un mini menú para que el usuario decida si busca un Aula o Profesor. El usuario puede aplastar una facultad y esto nos enviaría a la **Activity Menu_Block.** En caso de que el usuario opte por elegir dentro de mini menú Buscar Aula, lo enviará a la **Activity Busqueda**. Si elige Buscar profesor lo enviará a la **Activity Busqueda_profesor.**

Como toda Activity cuenta con una función **onCreate** donde inicializa el Layout relacionado a esta activity y los elementos de

la misma, además de que se agregó una solicitud de permisos para que el usuario permita a la App tener acceso al GPS del dispositivo móvil:

```
public class menu extends AppCompatActivity {
    private static final int MI_PERMISO_GPS = 1;
    //Variable tipo String que indicará el ID_facultad a buscar en Base de Datos
    String facultad="1";
    //Se definen los botones iniciales ya diseñados
    Button fiec,fcsh,fcnm,fimcp,fimcbor,fict,fcv,edcom,administracion;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        //Se avisa al usuario que la app requiere permisos GPS
        ActivityCompat.requestPermissions(menu.this,
                new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                MI_PERMISO_GPS);
        //Se instancian todos los botones
        fiec = (Button) findViewById(R.id.b_fiec);
        fcsh = (Button) findViewById(R.id.b_fcsh);
        fcnm = (Button) findViewById(R.id.b_fcnm);
        fimcp = (Button) findViewById(R.id.b_fimcp);
        fimcbor = (Button) findViewById(R.id.b_fimcbor);
        fict = (Button) findViewById(R.id.b_fict);
        fcv = (Button) findViewById(R.id.b_fcv);
        edcom = (Button) findViewById(R.id.b_edcom);
        administracion = (Button) findViewById(R.id.b_Administracion);
        //se llama a la funcion que dara clickListener a todos los botones.
        this.facultades();
```

La función **facultades()** permitirá agregar la acción a los botones que representan cada Facultad, enviando un valor numérico que corresponde al **ID_facultad guardado en la base de datos** a la **Activity Menu_Block.** También inicializa dicha Activity.

```
public void facultades(){
    fiec.setOnClickListener((view) -> {
            Intent intent = new Intent(menu.this, menu_block.class);
            //ID perteneciente a la FIEC
            facultad = "1";
            intent.putExtra("facultad", facultad);
            startActivity(intent);
            finish();
    });

    fcsh.setOnClickListener((view) -> {
            Intent intent = new Intent(menu.this, menu_block.class);
            //ID perteneciente a la FCSH
            facultad = "5";
            intent.putExtra("facultad", facultad);
            startActivity(intent);
            finish();
    });

    fcnm.setOnClickListener((view) -> {
            Intent intent = new Intent(menu.this, menu_block.class);
            //ID perteneciente a la FCNM
            facultad = "4";
            intent.putExtra("facultad", facultad);
            startActivity(intent);
            finish();
    });
    fimcp.setOnClickListener((view) -> {
```

Además, cuenta con la función **onCreateOptionMenu,** el cual agrega a la ToolBar (Barra superior donde sale el nombre de la App) el mini menú. La función **onOptionsItemSelected** da acción en cuanto presionamos un ítem del mini menú. Los ítems son: Buscar Aula – Buscar Profesor. Cuando presionamos Buscar Aula nos enviará a la **Activity Busqueda**, y si presionamos Buscar profesor nos enviará a la **Activity Busqueda_profesor**.

```
public boolean onCreateOptionsMenu(Menu menu) {
    //Se instancia los iconos que estaran en la toolbar de esta ventana
    //En este caso un menu el cual tiene la opción Buscar Aula o Buscar Profesor
    getMenuInflater().inflate(R.menu.menu_main,menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    //Se determinará la accion en caso de escoger un item del menu del toolbar
    int SelectID=item.getItemId();
    switch (SelectID){
        case R.id.Aula:
            //Se iniciará la activity Busqueda
            Intent intent=new Intent(menu.this,Busqueda.class);
            startActivity(intent);
            finish();
            break;
        case R.id.Profesor:
            //Se iniciará la activity Busqueda_profesor
            Intent intent2=new Intent(menu.this,busqueda_profesor.class);
            startActivity(intent2);
            finish();
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

MENU_BLOCK

Figura 2.13: Layout Menu_blockActivity

Esta Activity es dinámica y solo es llamada desde la **Activity Menu**. Permite crear botones dependiendo de un valor numérico enviado desde la **Activity Menu.** Estos botones serán los Bloques correspondientes a la facultad seleccionada en **Activity Menu** y serán creados dinámicamente dependiendo de la información recibida por la **base de datos**.

Como es una Activity que nos mostrara botones de los bloques correspondientes a una facultad se crea una clase llamada Bloques:

```
//Clase que contiene todos los atributos obtenidos en la base de Datos
class bloques{
    public int ID_bloque;
    public String Nombre;
    public double lat;
    public double lng;
    public int ID_facultad;
    public bloques(int ID_bloque, String nombre, double lat, double lng, int ID_facultad) {
        this.ID_bloque = ID_bloque;
        this.Nombre = nombre;
        this.lat = lat;
        this.lng = lng;
        this.ID_facultad = ID_facultad;
    }
}
```

Antes de la funcion **onCreate** declaramos algunas variables globales. Una de ellas es un ArrayList tipo Bloque. Este

ArrayList es el que sera llenado con la informacion descargada de la base de datos. En la funcion **onCreate** se llama a ejecutar una clase llamada ConsultarDatos de tipo AsynTask, es decir que sera ejecutada en segundo plano. Esta ejecutara enviando un URL que corresponde a la de nuestro servidor Web el cual tiene un archivo llamado **InfoBloque_facultad.php**, el cual recibe como parametro un entero el cual es el **ID_facultad** enviada por la Activity anteriormente mencionada:

```
public class menu_block extends Activity {
    //Entero que representara el recurso del piso correspondiente
    int piso=0;
    //El LinearLayout contenedor de los botones dinamicos
    LinearLayout contenedor;
    //String que determinara el modo de viaje elegido por el usuario
    String parameter="driving";
    //Variables utilizadas para la comprobacion de que se esta conectado al internet
    ConnectivityManager cm;
    NetworkInfo ni;
    //ArrayList Tipo Bloque, el cual sera llenado con la información de la base de Datos
    ArrayList<bloques> list= new ArrayList<>();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu_block);
        contenedor=(LinearLayout)findViewById(R.id.contenedor2);
        //Se llama una función que realizará la obtención de datos en segundo plano
        //Se envía el url del servidor con el codigo que Php que retornará los bloque dependiendo
        //de la variable ID_facultad recibida de la activity anterior (menu).
        new ConsultarDatos().execute("http://192.168.1.148/espolmaps/InfoBloque_facultad.php?ID_facultad="+getIntent().getE
    }
```

Código del archivo **InfoBloque_facultad.php.** Este código nos envía un JSONArray con la información de los bloques. Podemos ver un ejemplo de este codigo en la **Figura 2.14**

```
<?php
include('functions.php');
$facultad=$_GET["ID_facultad"];
$retunr_arr=array();


if($resultset=getSQLResultSet("Select * from Bloque where ID_facultad='$facultad'")){
        while ($row = $resultset->fetch_array(MYSQLI_ASSOC)){
                $row_array['ID_Bloque']=$row['ID_Bloque'];
                $row_array['Nombre_bloque']=$row['Nombre_bloque'];
                $row_array['Latitud']=$row['Latitud'];
                $row_array['Longitud']=$row['Longitud'];
                $row_array['ID_facultad']=$row['ID_facultad'];

                array_push($retunr_arr,$row_array);
        }
}
echo json_encode($retunr_arr);
?>
```

Figura 2.14: Código Php para recibir bloques por facultad

Resultado de URL enviando como **ID_facultad** el valor de 5, correspondiente a la facultad FCSH indicado en la **Figura 2.15**:

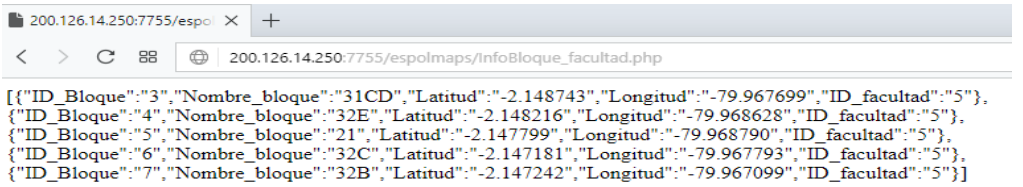

Figura 2.15: Respuesta del servidor y del código Php

La clase **ConsultarDatos** al ser una clase AsynTask posee métodos propios de esa clase como **doInBackground** el cual recibe el URL y llama a una función llamada **downloadUrl**:

```
private class ConsultarDatos extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... urls) {

        // params comes from the execute() call: params[0] is the url.
        try {
            //funcion que retorna un String recibido del url
            return downloadUrl(urls[0]);
        } catch (IOException e) {
            return "Unable to retrieve web page. URL may be invalid.";
        }
    }
    // onPostExecute displays the results of the AsyncTask.
```

downloadUrl es la función que realiza la conexiona internet, abre el URL y lee el contenido de respuesta del navegador mediante una función llamada **readIt**:

```
private String downloadUrl(String myurl) throws IOException {
    Log.i("URL",""+myurl);
    //reemplaza los espacios con %20
    myurl = myurl.replace(" ","%20");
    //variable que tomara todo lo que contiene la url
    InputStream is = null;
    // Only display the first 1500 characters of the retrieved
    // web page content.
    int len = 1500;

    try {
        //crea una variable url con el string recibido
        URL url = new URL(myurl);
        //realiza la conexion hacia dicha url
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
```

```
        conn.setReadTimeout(10000 /* milliseconds */);
        conn.setConnectTimeout(15000 /* milliseconds */);
        conn.setRequestMethod("GET");
        conn.setDoInput(true);
        // Starts the query
        conn.connect();
        int response = conn.getResponseCode();
        Log.d("respuesta", "The response is: " + response);
        //llena la variable con la respuesta de la url
        is = conn.getInputStream();

        // Convert the InputStream into a string
        String contentAsString = readIt(is, len);
```

```
        return contentAsString;

        // Makes sure that the InputStream is closed after the app is
        // finished using it.
    } finally {
        if (is != null) {
            is.close();
        }
    }
}
//funcion que convierte InputStream a String
public String readIt(InputStream stream, int len) throws IOException, Unsupported
    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    //lo lee mediandte cadena de caracteres
    char[] buffer = new char[len];
    reader.read(buffer);
    //retorna el String que sera convertido en JSONArray
    return new String(buffer);
}
```

Otra de las funciones de la clase **ConsultarDatos** es la función **onPostExcecute** el cual recibe el String que fue sacado después de leer la página resultado de la URL y es transformado en JSONarray. De aquí se decodifica dicho código y se agrega cada elemento al **ArrayList tipo Bloque creado inicialmente**. Luego se crean botones dependiendo del contenido del **ArrayList**. Además de que se le agrega la acción de cada botón mediante la función **dialogolista** que recibe información de cada botón:

```
protected void onPostExecute(String result) {
    //variable JSONArray que almacenara la información recibida de la URL en formato
    //String
    JSONArray ja = null;
    try {
        ja = new JSONArray(result);
        for(int i=0;i<ja.length();i++) {
            //Agrega de uno en uno el bloque correspondiente en el ArrayList creado inicialmente (list)
            list.add(new bloques(Integer.parseInt(ja.getJSONObject(i).getString("ID_Bloque").trim()),ja.getJSONObject(i).getString("Nombre_bloque"),Double.parseDouble(j
        }
        //se crea un boton dependiendo de los elementos dentro del ArrayList list tipo Bloque
        for(final bloques c: list){
```

```
        Button cb=new Button(menu_block.this);
        //Se agregan los parametros Width and Heigth
        cb.setLayoutParams(new LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.WRAP_CONTENT));
        //pone el ID del boton con el ID_Bloque
        cb.setId(c.ID_bloque);
        //Se pone la imagen correspondiente como Background al boton dependiendo del nombre
        //del bloque guardado en los recursos
        cb.setBackgroundResource(getResources().getIdentifier("bloque_"+ c.Nombre.toLowerCase(),"drawable",getPackageName()));
        cb.setHeight(5);
        //se agrega el boton al contenedor
        contenedor.addView(cb);
        //se crea un espacio de 20dp entre cada boton
        Space h=new Space(menu_block.this);
        h.setMinimumHeight(20);
        contenedor.addView(h);
```

```
        Button cb=new Button(menu_block.this);
        //Se agregan los parametros Width and Heigth
        cb.setLayoutParams(new LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.WRAP_CONTENT));
        //pone el ID del boton con el ID_Bloque
        cb.setId(c.ID_bloque);
        //Se pone la imagen correspondiente como Background al boton dependiendo del nombre
        //del bloque guardado en los recursos
        cb.setBackgroundResource(getResources().getIdentifier("bloque_"+ c.Nombre.toLowerCase(),"drawable",getPackageName()));
        cb.setHeight(5);
        //se agrega el boton al contenedor
        contenedor.addView(cb);
        //se crea un espacio de 20dp entre cada boton
        Space h=new Space(menu_block.this);
        h.setMinimumHeight(20);
        contenedor.addView(h);
        //se agrega el clickListener a cada boton
        cb.setOnClickListener((view) -> {
                //llama a funcion que hara una lista de dialogo para que el usuario elija si
                //camina o maneja al destino seleccionado.
                dialogolista(c.lat,c.lng,"Bloque "+c.Nombre,null,"bloque_"+ c.Nombre.toLowerCase());
        });
    }
} catch (JSONException e) {
    e.printStackTrace();
}
}
```

La función **dialogolista** me permite crear una pequeña ventana de opciones las cuales son los métodos de movilización que tendrá el usuario en el recorrido del mapa. Las opciones son Caminando o Manejando. Además, envía los datos recibidos junto a un String llamado **Parameter** que tiene un valor dependiendo de la opción seleccionada a un método llamado **mapas:**

```
//Metodo que muestra una lista de dialogo con opciones de manejando o caminando
public void dialogolista(final double lat, final double lng, final String descrip,String op,String block){
    //Se definen los items de la lista de dialogo
    final CharSequence[] items = {"Caminando","Manejando"};
    //Se saca el entero correspondiente al recurso de la foto seleccionada
    final int bloque=getResources().getIdentifier(block,"drawable",getPackageName());
    if(op!=null){
        //se saca el recurso del piso correspondiente
        piso=this.getResources().getIdentifier(op,"drawable",this.getPackageName());
    }
```

```
    //se crea el constructor de AlertDialog con el titulo correspondiente a Metodo de movilizacion
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Metodo de Movilizacion");
    //se agrega la accion en caso de elegir un item del AlertDialog
    builder.setSingleChoiceItems(items, 0, (dialog, item) → {
            if(item==0){
                //en caso de elegir caminando la variable parameter cambia a walking
                parameter="walking";
            }
            //llama a la funcion mapas
            mapas(lat,lng,descrip,parameter,piso,bloque,"");
    });
    //se crea el alertDialog dependiendo del constructor y se Muestra
    AlertDialog alert = builder.create();
    alert.show();
}
```

El método Mapas es la encargada de enviar la información hacia la Activity Maps. También llama a los métodos validacionGps y conectividad, los cuales verificaran que el móvil tenga conexión a internet y el GPS activo:

```
public void mapas(double latitud, double longuitud, String lugar, String parametro,int op,int bloque,String piso){
    //Metodo que nos enviará al MapActivity con ciertos datos que seran usados en dicha Activity
    Intent intent = new Intent(this, MapsActivity.class);
    intent.putExtra("Latitud",latitud);
    intent.putExtra("Longitud",longuitud);
    intent.putExtra("Lugar",lugar);
    intent.putExtra("parametro",parametro);
    intent.putExtra("op",op);
    intent.putExtra("bloque",bloque);
    intent.putExtra("Piso",piso);
    intent.putExtra("Tipo",0);
    //Se valida si hay conexion internet y GPS
    this.conectividad(intent);
    this.validacionGps();
}
public void validacionGps(){
    LocationManager locManager=(LocationManager)getSystemService(Context.LOCATION_SERVICE);
    if (!locManager.isProviderEnabled(LocationManager.GPS_PROVIDER))
    {
        //en caso de no tener activo el gps saldra un mensaje de error
        Toast toast1 = Toast.makeText(getApplicationContext(), "ERROR: Conecta tu Gps", Toast.LENGTH_SHORT);
        toast1.setGravity(Gravity.CENTER,0,0);

        toast1.show();
    }
}
```

```
public void conectividad(Intent intent){
    cm = (ConnectivityManager) this.getSystemService(Context.CONNECTIVITY_SERVICE);
    ni = cm.getActiveNetworkInfo();
    //se definen dos booleans que indicaran si existe conexion wifi o de datos moviles
    boolean tipoConexion1 = false;
    boolean tipoConexion2 = false;
    if (ni != null) {
        //se definen variables para validación de conexion wifi y por datos moviles
        ConnectivityManager connManager1 = (ConnectivityManager) this.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo mWifi = connManager1.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
```

```
        ConnectivityManager connManager2 = (ConnectivityManager) this.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo mMobile = connManager2.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);

        if (mWifi.isConnected()) {
            //si existe conexion wifi
            tipoConexion1 = true;
        }
        if (mMobile.isConnected()) {
            //si existe conexion de datos moviles
            tipoConexion2 = true;
        }

        if (tipoConexion1  || tipoConexion2 ) {
           /* Estas conectado a internet usando wifi o redes moviles, puedes seguir a la
            * MapActivity */
            startActivity(intent);
            finish();
```

```
        }
    }
    else {
   /* No estas conectado a internet */
        Toast toast1 = Toast.makeText(getApplicationContext(), "ERROR: Comprueba tu conexion a internet", Toast.LENGTH_SHOR
        toast1.setGravity(Gravity.CENTER,0,0);

        toast1.show();
    }
}
```

Por ultimo tenemos el método **onBackPressed**, que en este caso finalizara esta Activity, pero nos enviara de regreso a la **Activity Menu:**

```
public void onBackPressed() {
    //Al presionar el BackBoton regresaras a la activity anterior (menu)
    Intent intent = new Intent(this, menu.class);
    startActivity(intent);
    finish();
}
```

BUSQUEDA

Figura 2.16: Layout busquedaActivity

Es una Activity dinámica, es decir que su contenido depende de los datos recibidos por la **base de datos**. Estos datos son

recibidos dependiendo de la información ingresada por medio de un searchView o buscador. El contenido que tendrá esta Activity será un RecyclerView el cual será creado en segundo plano y tendrá la información de las Aulas que tengan similitud con los datos ingresados en el buscador. En la función **onCreate** agrega el searchView en el ToolBar, y una acción al searchView dependiendo del texto que esté recibiendo. La acción ingresada es de ejecutar una clase llamada **backgroundTask** de tipo Asynctask la cual recibe el texto ingresado y el contexto de esta Activity:

```
public class Busqueda extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_busqueda);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);


    }
    public boolean onCreateOptionsMenu(Menu menu) {
        //Se instancia el menu con los items correspondientes, en nuestro caso sera un
        //searchView
        getMenuInflater().inflate(R.menu.menu_aula_profesor,menu);
        final MenuItem searchItem = menu.findItem(R.id.BuscarAula);
        final SearchView searchView=(SearchView) MenuItemCompat.getActionView(searchItem);
        //Se coloca los Listener de lo ingresado en el searhView
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String query) {

                return false;
            }

            @Override
            public boolean onQueryTextChange(String newText) {
                //Iniciamos la clase BackgroundTask y ejecutamos lo que se ingresa en el searchView

                backgroundTask prueba=new backgroundTask(Busqueda.this);
                prueba.execute("%25"+newText+"%25");
                return true;
            }
        });
        return true;
    }
```

El método **onBackPressed** finalizara esta Activity y nos enviara a la **Activity Menu**:

```
public void onBackPressed() {
    //si presionamos el BackBoton nos enviara a la Activity anterior(menu)
    Intent intent = new Intent(this, menu.class);
    startActivity(intent);
    finish();
}
```

La clase **backgroundTask** tiene los métodos correspondientes a una clase AsynTask como **doInBackground, onPreExecute**

y onProgressUpdate. Antes de entrar a estos métodos se crea el **constructor** de la clase, el cual recibe el contexto de la Activity que llama a esta clase. Además, se hace un String con el URL correspondiente a la Busqueda de Aulas:

```
public class backgroundTask extends AsyncTask<String,Aulas,String> {
    //variables que instancian a la activity de la que esta clase fue llamada.
    Context con;
    Activity activity;
    //RecyclerView en los cuales seran llenados con los datos obtenidos de un ArrayList
    // tipo Aula
    RecyclerView recyclerView;
    //Adaptador que ayudara a llenar el recyclerView
    RecyclerView.Adapter adapter;
    //herramienta que obtendra el formato layout de cada fila del recyclerView
    RecyclerView.LayoutManager layoutManager;
    //ArrayList tipo aula que sera llenada con datos de la base de datos
    ArrayList<Aulas> list=new ArrayList<>();
    //constructor de la Clase recibe el contexto de la activity busqueda
    public backgroundTask(Context con){
        this.con=con;
        activity=(Activity)con;
    }
    //URL que contiene el codigo php para busqueda de la información de Aulas
    String Json_atring="http://192.168.1.148/espolmaps/InfoAula.php?Nombre=";
```

También se crea una clase **Aula** para guardar los distintos datos recibidos por URL:

```
public class Aulas{
    public String Nombre_aula;
    public String Codigo;
    public String Tipo;
    public String MapId;
    public String Piso;
    public String Img;
    public String Nombre_bloque;
    public double lat;
    public double lng;
    public String Nombre_facultad;
    public Aulas( String Nombre_aula,String Codigo,String Tipo,String MapId,String Piso,String Img,String Nombre_bloque, double lat, double lng,String Nombre_facultad)
        this.Nombre_aula = Nombre_aula;
        this.Codigo=Codigo;
        this.Tipo=Tipo;
        this.MapId=MapId;
        this.Piso=Piso;
        this.Img=Img;
        this.Nombre_bloque=Nombre_bloque;
        this.lat = lat;
        this.lng = lng;
        this.Nombre_facultad=Nombre_facultad;
    }
}
```

El método **onPreExecute** crea el recyclerView y lo pone en la Activity que envió su contexto. Además de eso al Adaptador de ReciyclerView lo inicializa con una clase que hemos creado llamada **RecyclerAdapter** la cual envía un ArrayList de tipo Aula y el contexto recibido.

```
@Override
protected void onPreExecute() {
    //se instancia el recyclerView y se lo agrega a la activity busqueda
    recyclerView=(RecyclerView)activity.findViewById(R.id.recycler);
    //se instancia el layout del contexto de la activity busqueda
    layoutManager=new LinearLayoutManager(con);
    recyclerView.setLayoutManager(layoutManager);
    recyclerView.setHasFixedSize(true);
    //se inicializa la clase RecyclerAdapter enviando el contexto y el ArrayList
    adapter=new RecyclerAdapter(con,list);
    //se agrega el adaptador al recyclerView
    recyclerView.setAdapter(adapter);
}
```

En el método doInBackground se realizará la conexión de internet y la recepción de la respuesta URL concatenada al texto recibido en el buscador de la **Activity Busqueda.** También se decodificará el código JSON recibido y se ingresara los datos en un elemento tipo Aula el cual será enviado a un método llamado **publishProgress**, el cual enviara dicho elemento al método OnProgressUpdate el cual añadirá ese elemento al **ArrayList creado inicialmente** y será enviado al **RecyclerAdapter**:

```
protected String doInBackground(String... aula) {
    try {
        //string recibido por el searchView de la activity busqueda
        //reemplazamos los espacios con %20
        aula[0]=aula[0].replace(" ","%20");
        //se concatena el URL con el dato recibido del searchView
        URL url=new URL(Json_atring+aula[0]);
        //Se realiza la conexion al URL
        HttpURLConnection httpURLConnection=(HttpURLConnection)url.openConnection();
        //Se agrega lo optenido de la URL a un InputStream
        InputStream inputStream=httpURLConnection.getInputStream();
        //Se crea un bufferReader que leera cada linea obtenida del inputStream
        BufferedReader bufferedReader=new BufferedReader(new InputStreamReader(inputStream));
        //se crea un StringBuilder donde se guardaea lo leido por el buffer para ser transformado en
        //String
        StringBuilder stringBuilder=new StringBuilder();
        String line;
        //se lee linea por linea y se agrega al StringBilder
        while ((line=bufferedReader.readLine())!=null){

            stringBuilder.append(line+"\n");
        }

        //se desconecta
        httpURLConnection.disconnect();
        String json_string=stringBuilder.toString().trim();
        Log.d("JSON STRING",json_string);
        //Se transforma el string en JSONObjet
```

```
            JSONObject jsonObject=new JSONObject(json_string);
            //Se crea el JSONArray
            JSONArray jsonArray=jsonObject.getJSONArray("server_response");
            int count=0;
            while (count<jsonArray.length())
            {
                JSONObject jo=jsonArray.getJSONObject(count);
                count++;
                //se crea un aula con cada elemento leido del JSONArray
                Aulas aulas=new Aulas(jo.getString("Nombre_aula"),jo.getString("Codigo"),jo.getStrir
                //Se publica el aula obtenida para agregarlo al ArrayList creado inicialmente
                publishProgress(aulas);
            }

        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onProgressUpdate(Aulas... values) {
        //Se agrega un elemento al ArrayList tipo aula
        list.add(values[0]);
        //Se agrega elemento al RecyclerView
        adapter.notifyDataSetChanged();
    }
```

La clase **RecyclerAdapter** es la que nos permitirá llenar un RecyclerView al recibir un ArrayList de un tipo de elemento, en este caso es de tipo Aula y el contexto de la Activity que la requiere, en este caso es la **Activity Busqueda**. Antes de describir las funciones de la clase tenemos el constructor, el cual recibe el ArrayList tipo Aula y el contexto:

```
public class RecyclerAdapter extends Adapter<RecyclerAdapter.MyViewHolder>{
    //ArrayList que sera llenada con el arrayList del BackGroudTask
    ArrayList<Aulas> list=new ArrayList<>();
    //Contexto de la Activity Busqueda
    Context con;
    //parametro que sera enviado a la activity Map
    String parameter="driving";
    //constructor de la clase
    RecyclerAdapter(Context con,ArrayList<Aulas> arrayList){
        this.con=con;
        this.list=arrayList;
    }
```

La primera función que tenemos es **onCreateViewHolder** de tipo **MyViewHolder** que es una clase creada de manera predeterminada. En esta función llamamos al Layout correspondiente a la fila de nuestro recyclerView, el layout es mostrado en la **Figura 2.17**:

```
public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    //se instancia la columna del recyclerView
    View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.row_layout_aula,parent,false);
    return new MyViewHolder(view);
}
```

Figura 2.17: Layout row_Layout_aula

La clase interna **MyViewHolder** es llenada para instanciar los elementos de nuestro Layout:

```
public static class MyViewHolder extends RecyclerView.ViewHolder{
    //clase que instanciara cada elemento de la fila instanciada inicialmente
    TextView Nombre_aula,Codigo_aula,Bloque_aula,Piso_aula,Facultad_aula;
    Button boton;

    public MyViewHolder(View itemView) {
        super(itemView);
        //instancia los elementos de la fila
        Nombre_aula=(TextView)itemView.findViewById(R.id.Nombre_aula);
        Codigo_aula=(TextView)itemView.findViewById(R.id.Codigo_aula);
        Bloque_aula=(TextView)itemView.findViewById(R.id.Bloque_aula);
        Piso_aula=(TextView)itemView.findViewById(R.id.Piso_aula);
        Facultad_aula=(TextView)itemView.findViewById(R.id.Facultad_aula);
        boton=(Button) itemView.findViewById(R.id.irMapa);
    }
}
```

El método más importante de la clase **RecyclerAdapter** es el método **onBindViewHolder** el cual recibe el **Holder** el cual es una fila del RecyclerView y la posición. Esta posición recorrerá el **ArrayList tipo Aula recibido** y sacará los datos para ponerlos en los elementos de nuestro Holder. A la vez se le da una acción al elemento clickeable del holder, el cual es de mandar información y a la **Activity Maps** y enviarnos a dicha activity, finalizando la Activity que envió su contexto, el cual es la **Activity Busqueda**.

```
public void onBindViewHolder(MyViewHolder holder, final int position) {
    //se agrega los datos recolectados del arrayList y se agraga a cada elemento
    holder.Nombre_aula.setText("Aula: "+list.get(position).Nombre_aula);
    holder.Codigo_aula.setText("Codigo: "+list.get(position).Codigo);
    holder.Bloque_aula.setText("Bloque: "+list.get(position).Nombre_bloque);
    holder.Piso_aula.setText("Piso: "+list.get(position).Piso);
    holder.Facultad_aula.setText("Facultad: "+list.get(position).Nombre_facultad);
    //se agrega la imagen de fondo del boton con el recurso correspondiente al nombre del bloque
    holder.boton.setBackgroundResource(con.getResources().getIdentifier("bloque_"+list.get(position).Nombre_bloque.toLowerCase(),"drawable
    //se agrega el ClickListener al boton
    holder.boton.setOnClickListener((view) -> {
            //Items que apareceran en el AlertDialog
            final CharSequence[] items = {"Caminando","Manejando"};
            //entero correspondiente al recurso del bloque correspondiente del aula seleccionada
            final int bloque=con.getResources().getIdentifier("bloque_"+list.get(position).Nombre_bloque.toLowerCase(),"drawable",con.get
            //entero correspondiente al recuro del piso correspondiente del aula seleccionada
            final int floor=con.getResources().getIdentifier(list.get(position).Img,"drawable",con.getPackageName());
            //constructor del alertDialog y sera colocado contexto de la Activity Busqueda
            AlertDialog.Builder builder = new AlertDialog.Builder(con);
            builder.setTitle("Metodo de Movilizacion");
            builder.setSingleChoiceItems(items, 0, (dialog, item) -> {
                    if(item==0){
                        //si selecciona el item caminando cambiara el valor parameter a walking
                        parameter="walking";
                    }
                    //iniciara la Activity Maps y se enviaran valores que seran usados en dicha activity
                    Intent intent=new Intent(con,MapsActivity.class);
                    intent.putExtra("Latitud",list.get(position).lat);
                    intent.putExtra("Longitud",list.get(position).lng);
                    intent.putExtra("Lugar",list.get(position).Nombre_aula);
                    intent.putExtra("parametro",parameter);
                    intent.putExtra("op",floor);
                    intent.putExtra("Edificio",list.get(position).Nombre_bloque);
                    intent.putExtra("MapId",list.get(position).MapId);
                    intent.putExtra("Piso",list.get(position).Piso);
                    intent.putExtra("Codigo",list.get(position).Codigo);
                    intent.putExtra("bloque",bloque);
                    intent.putExtra("Tipo",1);
                    con.startActivity(intent);
                    //finalizara la activity busqueda
                    ((Activity)con).finish();
            });
            //crea el AlertDialog con ayuda del constructor y la muestra
            AlertDialog alert = builder.create();
            alert.show();

    });

}
```

BUSQUEDA_PROFESOR

Figura 2.18: Layout búsqueda_profesroActivity

La **Activity Busqueda_profesor** es muy semejante a la **Activity Busqueda**. La diferencia es que se busca profesores, por lo que se crea una Clase llamada **Profesor** que es idéntica a la clase Aula, pero con un String adicional llamado Nombre_profesor:

```
public class Profesor {
    public String Nombre_profesor;
    public String Nombre_aula;
    public String Codigo;
    public String Tipo;
    public String MapId;
    public String Piso;
    public String Img;
    public String Nombre_bloque;
    public double lat;
    public double lng;
    public String Nombre_facultad;

    public Profesor(String nombre_profesor, String nombre_aula, String codigo, String tipo, String mapId, String piso, String img, String nombre_bloque, double lat, dou
        Nombre_profesor = nombre_profesor;
        Nombre_aula = nombre_aula;
        Codigo = codigo;
        Tipo = tipo;
        MapId = mapId;
        Piso = piso;
        Img = img;
        Nombre_bloque = nombre_bloque;
        this.lat = lat;
        this.lng = lng;
        Nombre_facultad = nombre_facultad;
    }
}
```

Otra diferencia es el de la URL que se enviará. En este caso será la URL correspondiente a la búsqueda de Profesores:

```
public backgroundTask2(Context con){
    this.con=con;
    activity=(Activity)con;
}
String Json_atring="http://200.126.14.250:7755/espolmaps/OficinaProfesor.php?Nombre=";
@Override
protected String doInBackground(String... profesores) {
    try {
```

El **RecyclerAdapter** recibirá **Arraylist de tipo Profesor**. De ahí todos los métodos son iguales:

```
public class RecyclerAdapter2 extends RecyclerView.Adapter<RecyclerAdapter2.MyViewHolder> {
    ArrayList<Profesor> list=new ArrayList<>();
    Context con;
    String parameter="driving";

    RecyclerAdapter2(Context con,ArrayList<Profesor> arrayList){
        this.con=con;
        this.list=arrayList;
    }

    @Override
```

Activity búsqueda_profesor:

```
public class busqueda_profesor extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_busqueda_profesor);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

    }
    public boolean onCreateOptionsMenu(Menu menu) {
        //Se instancia el menu con los items correspondientes, en nuestro caso sera un
        //searchView
        getMenuInflater().inflate(R.menu.menu_aula_profesor,menu);
        final MenuItem searchItem = menu.findItem(R.id.BuscarAula);
        final SearchView searchView=(SearchView) MenuItemCompat.getActionView(searchItem);
        //Se coloca los Listener de lo ingresado en el searhView
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String query) {
              /*  backgroundTask prueba=new backgroundTask(Busqueda.this);
                prueba.execute("%25"+query+"%25");*/
                return false;
            }

            @Override
            public boolean onQueryTextChange(String newText) {
                //Iniciamos la clase BackgroundTask y ejecutamos lo que se ingresa en el searchView
                backgroundTask2 prueba=new backgroundTask2(busqueda_profesor.this);
                prueba.execute("%25"+newText+"%25");
                return true;
```

```
        }
    });
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) { return true; }
public void onBackPressed() {
    //si presionamos el BackBoton nos enviara a la Activity anterior(menu)
    Intent intent = new Intent(this, menu.class);
    startActivity(intent);
    finish();
}
}
```

backgroundTask2:

```
public class backgroundTask2 extends AsyncTask<String,Profesor,String> {
    //variables que instancian a la activity de la que esta clase fue llamada.
    Context con;
    Activity activity;
    //RecyclerView en los cuales seran llenados con los datos obtenidos de un ArrayList
    // tipo Profesor
    RecyclerView recyclerView;
    //Adaptador que ayudara a llenar el recyclerView
    RecyclerView.Adapter adapter;
    //herramienta que obtendra el formato layout de cada fila del recyclerView
    RecyclerView.LayoutManager layoutManager;
    //ArrayList tipo profesor que sera llenada con datos de la base de datos
    ArrayList<Profesor> list=new ArrayList<>();
    //constructor de la Clase recibe el contexto de la activity busqueda
    public backgroundTask2(Context con){
        this.con=con;
        activity=(Activity)con;
    }
    //URL que contiene el codigo php para busqueda de la información de profesores
    String Json_atring="http://192.168.0.101/espolmaps/OficinaProfesor.php?Nombre=";

    protected String doInBackground(String... profesores) {
        try {
            //string recibido por el searchView de la activity busqueda_profesor
            //reemplazamos los espacios con %20
            profesores[0] = profesores[0].replace(" ","%20");
            //se concatena el URL con el dato recibido del searchView
            URL url=new URL(Json_atring+profesores[0]);
            //Se realiza la conexion al URL
            HttpURLConnection httpURLConnection=(HttpURLConnection)url.openConnection();
            //Se agrega lo optenido de la URL a un InputStream
            InputStream inputStream=httpURLConnection.getInputStream();
            //Se crea un bufferReader que leera cada linea obtenida del inputStream
            BufferedReader bufferedReader=new BufferedReader(new InputStreamReader(inputStream));
            //se crea un StringBuilder donde se guarda lo leido por el buffer para ser transformado en
            //String
            StringBuilder stringBuilder=new StringBuilder();
            String line;
            //se lee linea por linea y se agrega al StringBilder
```

```
        while ((line=bufferedReader.readLine())!=null){

            stringBuilder.append(line+"\n");
        }
        //se desconecta
        httpURLConnection.disconnect();
        String json_string=stringBuilder.toString().trim();
        Log.d("JSON STRING",json_string);
        //Se transforma el string en JSONObjet
        JSONObject jsonObject=new JSONObject(json_string);
        //Se crea el JSONArray
        JSONArray jsonArray=jsonObject.getJSONArray("server_response");
        int count=0;

        while (count<jsonArray.length())
        {
            JSONObject jo=jsonArray.getJSONObject(count);
            count++;
            //se crea un profesor con cada elemento leido del JSONArray
            Profesor profesor=new Profesor(jo.getString("Nombre_profesor"),jo.getString("Nombre_aula"),
            //Se publica el profesor obtenida para agregarlo al ArrayList creado inicialmente
            publishProgress(profesor);
        }

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}

@Override
protected void onPreExecute() {
    //se instancia el recyclerView y se lo agrega a la activity busqueda
    recyclerView=(RecyclerView)activity.findViewById(R.id.recycler2);
    //se instancia el layout del contexto de la activity busqueda
    layoutManager=new LinearLayoutManager(con);
    recyclerView.setLayoutManager(layoutManager);
    recyclerView.setHasFixedSize(true);
    //se inicializa la clase RecyclerAdapter enviando el contexto y el ArrayList
    adapter=new RecyclerAdapter2(con,list);
    //se agrega el adaptador al recyclerView
    recyclerView.setAdapter(adapter);
}
@Override
protected void onProgressUpdate(Profesor... values) {
    //Se agrega un elemento al ArrayList tipo profesor
    list.add(values[0]);
    //Se agrega elemento al RecyclerView
    adapter.notifyDataSetChanged();
}

@Override
protected void onPostExecute(String result) { super.onPostExecute(result); }
}
```

RecyclerAdapter2:

```
public class RecyclerAdapter2 extends RecyclerView.Adapter<RecyclerAdapter2.MyViewHolder> {
    //ArrayList que sera llenada con el arrayList del BackGroudTask
    ArrayList<Profesor> list=new ArrayList<>();
    //Contexto de la Activity Busqueda
    Context con;
    //parametro que sera enviado a la activity Map
    String parameter="driving";
    //constructor de la clase
    RecyclerAdapter2(Context con,ArrayList<Profesor> arrayList){
        this.con=con;
        this.list=arrayList;
    }

    @Override
    public RecyclerAdapter2.MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        //se instancia la columna del recyclerView
        View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.row_layout_profesor,parent,fals
        return new RecyclerAdapter2.MyViewHolder(view);
    }

    @Override
    public void onBindViewHolder(RecyclerAdapter2.MyViewHolder holder, final int position) {
        //se agrega los datos recolectados del arrayList y se agraga a cada elemento
        holder.Profesor.setText("Profesor: "+list.get(position).Nombre_profesor);
        holder.Oficina.setText("Oficina: "+list.get(position).Codigo);
        holder.Bloque_aula.setText("Bloque: "+list.get(position).Nombre_bloque);
        holder.Piso_aula.setText("Piso: "+list.get(position).Piso);
        holder.Facultad_aula.setText("Facultad: "+list.get(position).Nombre_facultad);
        //se agrega la imagen de fondo del boton con el recurso correspondiente al nombre del bloque
        holder.foto.setImageResource(con.getResources().getIdentifier("bloque_"+list.get(position).Nombre_bl
```

```
    //se agrega el ClickListener al boton
    holder.foto.setOnClickListener((view) -> {
            //Items que apareceran en el AlertDialog
            final CharSequence[] items = {"Caminando","Manejando"};
            //entero correspondiente al recurso del bloque correspondiente del profesor seleccionado
            final int bloque=con.getResources().getIdentifier("bloque_"+list.get(position).Nombre_bloque
            //entero correspondiente al recuro del piso correspondiente del profesor seleccionado
            final int floor=con.getResources().getIdentifier(list.get(position).Img,"drawable",con.getPa
            //constructor del alertDialog y sera colocado contexto de la Activity Busqueda_profesor
            AlertDialog.Builder builder = new AlertDialog.Builder(con);
            builder.setTitle("Metodo de Movilizacion");
            builder.setSingleChoiceItems(items, 0, (dialog, item) -> {
                    if(item==0){
                        //si selecciona el item caminando cambiara el valor parameter a walking
                        parameter="walking";
                    }
                    //iniciara la Activity Maps y se enviaran valores que seran usados en dicha activity
                    Intent intent=new Intent(con,MapsActivity.class);
                    intent.putExtra("Latitud",list.get(position).lat);
                    intent.putExtra("Longitud",list.get(position).lng);
                    intent.putExtra("Lugar",list.get(position).Nombre_profesor);
                    intent.putExtra("parametro",parameter);
                    intent.putExtra("op",floor);
                    intent.putExtra("Edificio",list.get(position).Nombre_bloque);
                    intent.putExtra("Codigo",list.get(position).Codigo);
                    intent.putExtra("Piso",list.get(position).Piso);
                    intent.putExtra("Facultad",list.get(position).Nombre_facultad);
                    intent.putExtra("MapId",list.get(position).MapId);
                    intent.putExtra("bloque",bloque);
```

```
                intent.putExtra("Tipo",2);
                con.startActivity(intent);
                //finalizara la activity busqueda_profesor
                ((Activity)con).finish();
            });
            //crea el AlertDialog con ayuda del constructor y la muestra
            AlertDialog alert = builder.create();
            alert.show();

        });
    }
```

```
    @Override
    public int getItemCount() {
        //retorna el tamaño del recyclerView que en este caso es el
        //tamaño del ArrayList
        return list.size();
    }

    public static class MyViewHolder extends RecyclerView.ViewHolder{
        //clase que instanciara cada elemento de la fila instanciada inicialmente
        TextView Profesor,Oficina,Bloque_aula,Piso_aula,Facultad_aula;
        ImageView foto;

        public MyViewHolder(View itemView) {
            super(itemView);
            //instancia los elementos de la fila
            Profesor=(TextView)itemView.findViewById(R.id.Nombre_profesor);
            Oficina=(TextView)itemView.findViewById(R.id.Oficina_profesor);
            Bloque_aula=(TextView)itemView.findViewById(R.id.Bloque_profesor);
            Piso_aula=(TextView)itemView.findViewById(R.id.Piso_profesor);
            Facultad_aula=(TextView)itemView.findViewById(R.id.Facultad_profesor);
            foto=(ImageView) itemView.findViewById(R.id.img_profesor);
        }
    }
}
```

MAPS

Figura 2.19: Layout MapsActivity

Es la Activity que nos mostrara el mapa y el camino que deberá recorrer el usuario hasta su bloque, aula, oficina o profesor elegido en las Activities anteriores, ya sea que su opción haya

sido elegida en la **Activity Menu_Block** (Elige el bloque), **Activity Busqueda** (Elige el Aula) o la **Activity Busqueda_profesor** (Elige el profesor). En cada una de esas Activities se envió diferente tipo de información, y una de ellas es un entero que determina el tipo de búsqueda que realizo el usuario. Dependiendo de ese tipo se mostrará la información correspondiente al marcador destino.

La Activity Maps al ser de tipo Fragmento de GoogleMaps tiene dos métodos por defecto: **onCreate y onMapReady.** En **onCreate** se instancia el Layout correspondiente de la Activity

```
public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback {
    private static int VELOCIDAD_MIN_CAMINANDO = 83;
    private static int VELOCIDAD_MIN_MANEJANDO = 1000;
    private GoogleMap mMap;
    private Marker marcador;
    private Marker desti;
    double lat = -2.147614;
    double lng = -79.968466;
    ArrayList<LatLng> points = null;
    TextView tiempotext;
    TextView distanciaText;
    ImageView foto;
    MediaPlayer mp;
    int cont=0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
        //Se instancia elementos donde se llenara con el tiempo de destino, la distancia y la foto
        tiempotext = (TextView) findViewById(R.id.tiempo1);
        distanciaText = (TextView) findViewById(R.id.distancia);
        foto = (ImageView) findViewById(R.id.direccion);
        //Se reproduce audio de inicio del recorrido
        mp = MediaPlayer.create(this, R.raw.comenzamos);
        mp.start();
    }
```

onMapReady es el método que permitirá dibujar el mapa y sus elementos, además de que llamara a una serie de funciones que realizaran tareas de Geolocalización, insertar imágenes en el mapa, etc. La primera función que llamara este método es la función **miUbicacion()** la cual es la que nos dibujara en el mapa un marcador con nuestra posición actual, y cambiar automáticamente si el usuario se va moviendo. La primera

información recibida por las Activities pasadas es la Latitud y Longitud del destino, estos son colocados en una variable tipo LatLng destino. La segunda información recibida será el tipo de destino. El cual tendrá valor 0 si el destino es un Bloque, valor 1 si el destino es un Aula y valor 2 si el destino es un Profesor. Se inicia un marcador y se llena la información del mismo dependiendo del Tipo del destino y de su Latitud y longitud:

```
@Override
public void onMapReady(GoogleMap googleMap) {
    boolean validar = true;
    //inicializa el mapa de googlemaps
    mMap = googleMap;
    //Se verifica el tipo de información recibida de Activitys Anteriores:
    //0 si es bloque, 1 si es aula, 2 si es profesor
    int opcion = getIntent().getExtras().getInt("Tipo");
    //Se obtiene la foto correspondiente al destino
    foto.setImageResource(getIntent().getExtras().getInt("bloque"));
    //Metodo que hará la geolocalización
    miUbicacion();
    //se coloca la posicion inicial en la variable origen
    //las variable globales lat y lng cambiaran en el transcurso del camino.
    LatLng origen = new LatLng(lat, lng);
    //Información recibida desde la activitys anteriores
    LatLng destino = new LatLng(getIntent().getExtras().getDouble("Latitud"), getIntent().getExtras().ge
    //Se agregara marcador en el destino con la información dependiendo del tipo recibido
    switch (opcion) {
        //en caso de bloque se agregara el nombre, latitud y longitud
        case 0:
            desti = mMap.addMarker(new MarkerOptions()
                    .position(destino)
                    .title("Destino: " + getIntent().getExtras().getString("Lugar"))
                    .snippet("Latitud: " + destino.latitude + "\nLongitud: " + destino.longitude)
                    .icon(BitmapDescriptorFactory.defaultMarker()));
            break;
        //en caso de ser aula, se mostrara nombre, codigo, bloque, piso y facultad
        case 1:
            desti = mMap.addMarker(new MarkerOptions()
                    .position(destino)
                    .title("Destino: " + getIntent().getExtras().getString("Lugar"))
                    .snippet("Codigo: " + getIntent().getExtras().getString("Codigo") + "\nBloque: " + g
                    .icon(BitmapDescriptorFactory.defaultMarker()));
            break;
        //en caso de ser profesor, se mostrara nombre, oficina, bloque, piso y facultad.
        case 2:
            desti = mMap.addMarker(new MarkerOptions()
                    .position(destino)
                    .title("Destino: " + getIntent().getExtras().getString("Lugar"))
                    .snippet("Oficina: " + getIntent().getExtras().getString("Codigo") + "\nBloque: " + getI
                    .icon(BitmapDescriptorFactory.defaultMarker()));
            break;
    }
```

Después de haber agregado nuestro marcador destino le agregamos una **Ventana de Información**. Esta será desplegada al dar un clic en el marcador. La ventana de información nos mostrara la información ingresada en el marcador, colocada en el paso anterior, más una imagen del edificio en donde se apunta. Para eso se crea una Clase interna llamada **UserInfoWindowAdapter**, la imagen obtenida depende del valor recibido por las Activities anteriores llamado **"Bloque":**

```
class UserInfoWindowAdapter implements GoogleMap.InfoWindowAdapter{
    int bloque=0;
    LayoutInflater inflater=null;
    UserInfoWindowAdapter(LayoutInflater inflater) { this.inflater=inflater; }
    @Override
    public View getInfoWindow(Marker marker) { return null; }

    @Override
    public View getInfoContents(Marker marker) {
        //Se instancia un layout llamado info_vindow con el modelo de la ventada de información
        View marca = inflater.inflate(R.layout.info_window,null,false);
        //Se instancian cada elemento del Layout
        TextView Nombre=(TextView)marca.findViewById(R.id.txtNombre);
        TextView Sniped=(TextView)marca.findViewById(R.id.txtSniped);
        ImageView foto=(ImageView)marca.findViewById(R.id.img);
        //se agrega la información de los marcadores en los textview
        Nombre.setText(marker.getTitle());
        Sniped.setText(marker.getSnippet());
        //solo se agregara la imagen al destino
        if(!Nombre.getText().equals("Mi localizacion")){
            bloque=getIntent().getExtras().getInt("bloque");
            foto.setBackgroundResource(getIntent().getExtras().getInt("bloque"));
            return marca;
        }
        else
            return null;
    }
```

El Layout al cual hace referencia es de este modelo mostrado en la **Figura 2.20**:

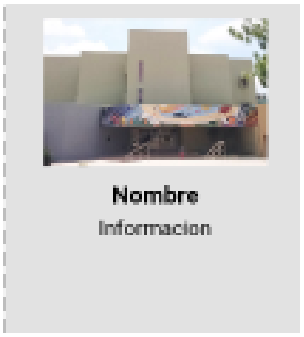

Figura 2.20: Layout infowindows

Otra función que es llamada dentro de **onMapReady** es el método **insertarPiso**, el cual recibe las coordenadas destino y su función es colocar las divisiones de los edificios si estas existen. Para nuestro proyecto solo existen las divisiones de los Bloques que pertenecen a la FIEC. Antes de llamar a las funciones que dibujen los caminos de manera dinámica, validamos la posición de origen y destino para dibujar caminos estáticos, ya que Google no cuenta con todos los caminos de la ESPOL. Para eso llamamos a la función **validacionOrigen()** la cual retorna un boolean y caminos dibujados dinámicamente en el mapa dependiendo si aplica. Después se valida ese boolean y se cambia las ubicaciones de origen y destino con la función **newLatln()**, la cual cambia el origen y destino para que se creen los caminos dinámicos con las funciones siguientes.

También se llama a la función tipo String **obtenerDireccionesUrl** el cual recibe las coordenadas origen y destino. Esta función es la que escribirá la dirección URL de los servidores de Google maps que nos entregará las rutas y caminos que deberá tomar el usuario. Se elige el tipo de mapa que se desea dibujar, en nuestro caso es de tipo **HYBRID** el cual es satelital con información de los caminos. Y por último se llama a una Clase interna llamada **DownloadTask** que es de tipo AsynTask que se encargara de ejecutar el URL obtenido de la función **obtenerDireccionesURL** en segundo plano:

```
mMap.setInfoWindowAdapter(new UserInfoWindowAdapter(getLayoutInflater()));
//funcion que permite colocar un piso dividido en el mapa si este existe
insertarPiso(destino);
//se coloca un mapa de tipo satelital
mMap.setMapType(MAP_TYPE_SATELLITE);
//se verifica el metodo de transporte para ver si se colocan caminos estáticos
if(getIntent().getExtras().getString("parametro").equals("walking")){
    //primero se verifica si el origen y destino se encuentran en el mismo cuadrante
    //estático y se dibuja en caso de ser cierto una ruta estatica
    validar=validacionOrigen(origen,destino);
    //En caso de no estar se verifica si el origen o el destino estan en esos cuadrantes
    if (validar) {
        //se dibuja rutas estaticas si se encuentran en los cuadrantes en los que no
        // existen caminos dinamicos de google y se dan
        //nuevas coordenadas para dibujar rutas dinamicas
        origen=newLatln(origen);
        destino=newLatln(destino);
```

```
            //se dibujan rutas dinamicas con las nuevas coordenadas
            //se crea la url
            String url = obtenerDireccionesURL(origen, destino);
            //se ejecuta en segundo plano la url y se dibuja las rutas dinamicas
            DownloadTask downloadTask = new DownloadTask();
            downloadTask.execute(url);
        }
    }
    //en caso de no ser a pie el recorrido se aplica el metodo normal
    else{
        //se crea la url con el origen y destino normales
        String url = obtenerDireccionesURL(origen, destino);
        //se ejecuta el url y se dibuja los caminos dinamicos
        DownloadTask downloadTask = new DownloadTask();
        downloadTask.execute(url);
    }
}
```

El método **miUbicacion** es el método que usara el GPS de nuestro teléfono y cambiara la ubicación dependiendo de que nos movemos, pero esta función no es la que colocara el marcador. Mediante un **LocationManeger y un LocationListener** llamaremos a la función **actualizaUbicacion** que es la que nos permitirá colocar el marcador y recibe como parámetro la localización actual del usuario:

```
private void miUbicacion() {
    //se inicia el servicio GPS
    LocationManager locManeger = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    //Se crea una variable criteria con atributos para conocer el mejor provedor de localización
    Criteria criteria=new Criteria();
    criteria.setPowerRequirement(Criteria.POWER_LOW);
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    criteria.setAltitudeRequired(false);
    criteria.setBearingRequired(false);
    criteria.setSpeedRequired(false);

    criteria.setCostAllowed(false);
    //se elije el provedor con la variable criteria seleccionada
    String Provedor=locManeger.getBestProvider(criteria,true);
    int dist,temp;
    if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.P

        return;
    }
    //Se concede la ultima localización conocida por el movil
    Location location = locManeger.getLastKnownLocation(Provedor);
    actualizaUbicacion(location);
    //se crea una variable con las coordenadas del destino de las activitys anteriores
    Location location2 = new Location("Location2");
    location2.setLatitude(getIntent().getExtras().getDouble("Latitud"));
    location2.setLongitude(getIntent().getExtras().getDouble("Longitud"));
    //se calcula la distancia inicial de la ubicación actual con el destino
    dist=(int)(location.distanceTo(location2));
    //se calcula el tiempo inicial dependiendo del modo de transporte dividiendo la distancia por la velocidad
    if(getIntent().getExtras().getString("parametro").equals("walking"))
        temp=dist/VELOCIDAD_MIN_CAMINANDO;
    else
        temp=dist/VELOCIDAD_MIN_MANEJANDO;
    //se coloca el tiempo y la distancia inicial en el cuadro de dialogo debajo de la pantalla
    tiempotext.setText(temp+" min");
```

```
    distanciaText.setText(dist+ " metros");
    //se inicia la busqueda de updates con el listener
    locManeger.requestLocationUpdates(Provedor,0,0,loclistener);
}
```

El **LocationListener** es una clase que realizara una acción en cuando el proveedor GPS mande una notificación, en nuestro caso es el cambio de ubicación. Esta clase tiene métodos predeterminados. El que nos va a interesar es **onLocationChanged**. En este método lo que haremos es llamar a nuestro método **actualizaUbicacion** enviando la nueva ubicación recibida por el proveedor GPS. Además de actualizar la Ubicación calcula la distancia y el tiempo que tiene el usuario antes llegar a su destino. Llama al método **voces(int dist), que** recibe como parámetro la distancia que está el usuario hacia su destino.

```
//se crea un listener el cual recibira las notificaciones de localizacion y realizaran acciones al
//tener un cambio de localizacion
LocationListener loclistener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        //realiza la actualización de ubicacion
        actualizaUbicacion(location);
        int dist=0;
        int temp=0;
        Location location2 = new Location("Location2");
        //se crea una variable de localizacion con las coordenadas de destino recibidas de Activitys anteriores
        location2.setLatitude(getIntent().getExtras().getDouble("Latitud"));
        location2.setLongitude(getIntent().getExtras().getDouble("Longitud"));
        //se realiza el siguiente codigo para calcular distancia actual con el destino

        dist=(int)(location.distanceTo(location2));
        //se calcula el tiempo dividiendo el tiempo para la velocidad dependiendo del modo de transporte
        if(getIntent().getExtras().getString("parametro").equals("walking"))
            temp=dist/VELOCIDAD_MIN_CAMINANDO;
        else
            temp=dist/VELOCIDAD_MIN_MANEJANDO;
        //Se agrega el tiempo y la distancia en el cuadro de dialogo debajo de la pantalla, estos cambiaran
        //a medida que cambien los valores de temp y dist
        tiempotext.setText(temp+" min");
        distanciaText.setText(dist+ " metros");
        //se llama al metodo que reproducirá voz dependiendo de la distancia
        voces(dist);
    }
```

El método **actualizarUbicacion** recibirá una localización y lo que hará es darle el valor de Latitud y Longitud de la localización recibida a nuestras variables globales **Lat y Lng** en caso de que la Localización recibida no sea Null. Después llama al método **agregarMarcador** y manda como parámetros el valor de nuestras variables globales y la inclinación de dirección del usuario:

```
private void actualizaUbicacion(Location localition) {
    if (localition != null) {
        //metodo recibe la localizacion actual y cambia los valores globales
        lat = localition.getLatitude();
        lng = localition.getLongitude();
        //se llama al metodo que agrega el marcador con las variables globales e inclinación
        agregarMarcador(lat, lng,localition.getBearing());
    }
}
```

El método **agregarMarcador** recibe las coordenadas de la posición actual del usuario y dibuja el marcador en esa posición. Al ser esta llamada siempre en el método **actualizarUbicacion** cambiará la ubicación del marcador siempre y cuando la localización del usuario cambie, además recibe la inclinación de dirección, esta se coloca en una variable CameraPosition, la cual hará girar el mapa automáticamente hacia la dirección en donde el usuario se está moviendo:

```
private void agregarMarcador(double lat, double lng,float bearing) {
    //variable que tendra la posicion actual
    LatLng localizacion = new LatLng(lat, lng);
    //posicion de la camara con inclinacion actual, esto movera el mapa dependiendo de la posicion actual
    CameraPosition cam=CameraPosition.builder().bearing(bearing).tilt(45).target(localizacion).zoom(20).build();
    //removera marcador actual
    if (marcador != null) marcador.remove();
    //coloca un nuevo marcador, esto simulara que el marcador se esta moviendo con el cambio de posicion

    marcador = mMap.addMarker(new MarkerOptions()
            .position(localizacion)
            .title("Mi localizacion")
            .icon(BitmapDescriptorFactory.fromResource(R.drawable.marcador_origen2))
    );
    //mueve la camara del mapa dependiendo de la configuración anterior.
    mMap.animateCamera(CameraUpdateFactory.newCameraPosition(cam));
}
```

La función **insertarPiso** recibe como parámetro las coordenadas del destino y también recibe de las Activities anteriores un estero llamado **“op”** el cual representa la imagen del piso dividido. Si este valor es 0 no agrega el piso dividido, caso contrario lo coloca en las coordenadas recibidas. La función **obtenerDireccionesURL** recibe como parámetros las coordenadas de origen y destino, además de un valor enviado de las Activities anteriores llamado **“parametro”,** el cual tiene un valor walking o driving dependiendo de lo seleccionado en dichas Activities. Los caminos varían dependiendo de dicho parámetro:

```
private void insertarPiso(LatLng destino) {
    //busca en los recursos el piso dividido enviado desde la activity anterior
    if(getIntent().getExtras().getInt("op")!=0){
        GroundOverlayOptions newarkMap = new GroundOverlayOptions()
                .image(BitmapDescriptorFactory.fromResource(getIntent().getExtras().getInt("op")))
                .position(destino,81, 100);
        //agrega el piso en el mapa
        mMap.addGroundOverlay(newarkMap);
    }
}

private String obtenerDireccionesURL(LatLng origin, LatLng dest) {
    //crea la url con la informacion de coordenada de origen y destino
    String str_origin = "origin=" + origin.latitude + "," + origin.longitude;
    String str_dest = "destination=" + dest.latitude + "," + dest.longitude;
    String sensor = "sensor=false";
    //se elige el modo de transporte tomado desde las activitys anteriores
    String mode = "mode="+ getIntent().getExtras().getString("parametro");
    String parameters = str_origin + "&" + str_dest + "&" + mode + "&" + sensor;
    //La información recibida sera en formato JSON
    String output = "json";
    String url = "https://maps.googleapis.com/maps/api/directions/" + output + "?" + parameters;
    return url;
}
```

La clase **DowloadTask** es de tipo Asynctask. Se realizará en segundo plano y su proceso es el de descargar le resultado de lo obtenido en la URL que se creó en funciones ya explicada anteriormente en esta Activity y colocarlo en un String llamado data para luego enviarlo a otro proceso en segundo plano llamado **ParseTask**. La primera función tipo String que se analiza es doInBackground, llama a la función **downloadUrl** enviando como parámetro la URL recibida en esta clase:

```
private class DownloadTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... url) {
        String data = "";
        try {
            //Se llama al metodo downloadUrl con la url creada
            data = downloadUrl(url[0]);
        } catch (Exception e) {
            Log.d("ERROR AL OBTENER INFO S", e.toString());
        }
        return data;
    }
```

La función **downloadUrl** es la función que se conectara a internet y cargara el URL recibido. Como resultado leerá lo obtenido de esa página y lo guardará en una variable string llamada data. Esta variable es la que se enviara al método **onPostExecute** de nuestra clase **DownloadTask**:

```
private String downloadUrl(String strUrl) throws IOException {
    String data = "";
    InputStream iStream = null;
    HttpURLConnection urlConnection = null;
    try {
        URL url = new URL(strUrl);
        // Creamos una conexion http
        urlConnection = (HttpURLConnection) url.openConnection();
        // Conectamos
        urlConnection.connect();
        // Leemos desde URL
        iStream = urlConnection.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(iStream));
        StringBuffer sb = new StringBuffer();
        String line = "";
        while ((line = br.readLine()) != null) {
            sb.append(line);
        }
        data = sb.toString();
        br.close();
    } catch (Exception e) {
        Log.d("Exception", e.toString());
    } finally {
        iStream.close();
        urlConnection.disconnect();
    }
    return data;
}
```

En el método **onPostExecute de DownloadTask** se enviará nuestro string a la clase **ParserTask** que será la encargada de decodificar el código JSON de la URL y dibujará las Polylines en nuestro mapa:

```
@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);
    //Se decodifica lo recibido de la url llamando a una clase
    //ejecutara en segundo plano esta acción
    ParserTask parserTask = new ParserTask();
    parserTask.execute(result);
}
```

La clase **ParserTask** será la encargada de dibujar el camino de nuestro mapa. Para eso primero debe decodificar el código JSON de la URL y de ahí decodificar el código de las polylines y agregarlos a ArrayList de tipo LatLng y de ahí sacar los datos y dibujarlas en el mapa. Para eso usaremos los métodos predeterminados de una clase Asynctask que son **doInBackground y onPostExecute**. En **doInBackground** llamaremos a la clase **DirectionsJSONParser** la cual tiene una función llamada **parse** que recibe como parámetro un JSONObject y retorna una matriz de String. Los cuales contienen los diferentes elementos de la ruta como pasos, coordenadas origen, coordenada destinos, indicaciones, **Polylines,** etc.

```
private class ParserTask extends AsyncTask<String, Integer, List<List<HashMap<String, String>>>> {
    @Override
    protected List<List<HashMap<String, String>>> doInBackground(String... jsonData) {
        JSONObject jObject;
        //lista de rutas creadas para almacenar datos referentes a polilineas
        List<List<HashMap<String, String>>> routes = null;
        try {
            //se crea un Json Objetct y se lo inicia con el string leido del URL
            jObject = new JSONObject(jsonData[0]);
            //se llama a una clase para llenar la lista de rutas
            DirectionsJSONParser parser = new DirectionsJSONParser();
            routes = parser.parse(jObject);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return routes;
    }
```

El método **onPostExcecute** de la clase **ParserTask** es la encargada de dibujar las polylines sacadas de la matriz de String resultado de la función **doInBackground.** Se crea un ArrayList de tipo LatLng y se agrega los puntos de la Polilyne a dibujar. Luego se envía dicho ArrayList a un **PolylineOptions** el cual se le configura el color y ancho de la polyline a dibujar. Por último, se agrega la PolylineOptions al mapa:

```
@Override
protected void onPostExecute(List<List<HashMap<String, String>>> result) {
    PolylineOptions lineOptions = null;
    MarkerOptions markerOptions = new MarkerOptions();
    //se recorre la lista y se toma los valores de coordenadas y se agrega a una lista de latlng
    for (int i = 0; i < result.size(); i++) {
        points = new ArrayList<LatLng>();
        lineOptions = new PolylineOptions();
        List<HashMap<String, String>> path = result.get(i);
        for (int j = 0; j < path.size(); j++) {
            HashMap<String, String> point = path.get(j);
            double lat = Double.parseDouble(point.get("lat"));
            double lng = Double.parseDouble(point.get("lng"));
            LatLng position = new LatLng(lat, lng);
            points.add(position);
        }
        //se agrega la lista de Latlng al lineOptions, se configura grosor y color
        lineOptions.addAll(points);
        lineOptions.width(12);
        lineOptions.color(Color.rgb(0, 0, 255));
    }
    if (lineOptions != null) {
        //se agregan las polilineas al mapa
        mMap.addPolyline(lineOptions);
    }
}
```

La clase **DirectionsJSONParser** es la encargada de decodificar un código JSON. Esta clase tiene una función

llamada **parse**, la cual fue convocada en el **doInBackground** de la clase **ParserTask**. Esta función regresa la matriz decodificada del JSON recibido, para eso dentro de su lectura invoca una función llamada **decodePoly** la cual recibe como parámetro un string codificado de la polyline y retorna una lista de LatLng que corresponde a una polyline:

```
public class DirectionsJSONParser {
    //metodo que recibe un jsonObject y retorna una lista de hasmap de strings con las rutas decodificadas
    public List<List<HashMap<String, String>>> parse(JSONObject jObject) {
        List<List<HashMap<String, String>>> routes = new ArrayList<List<HashMap<String, String>>>();
        JSONArray jRoutes = null;
        JSONArray jLegs = null;
        JSONArray jSteps = null;

        try {
            //se separan las rutas obtenidas
            //se recorre el JsonArray para sacar las polilineas
            jRoutes = jObject.getJSONArray("routes");
            for (int i = 0; i < jRoutes.length(); i++) {
                jLegs = ((JSONObject) jRoutes.get(i)).getJSONArray("legs");
                List path = new ArrayList<HashMap<String, String>>();

                for (int j = 0; j < jLegs.length(); j++) {
                    jSteps = ((JSONObject) jLegs.get(j)).getJSONArray("steps");

                    for (int k = 0; k < jSteps.length(); k++) {
                        //se obtienen todas las polilineas en forma de String
                        String polyline = "";
                        polyline = (String) ((JSONObject) ((JSONObject) jSteps.get(k)).get("polyline")).get("points
                        //se decodifica las lineas y se agrega a una lista de LatLng
                        List<LatLng> list = decodePoly(polyline);

                        for (int l = 0; l < list.size(); l++) {
                            HashMap<String, String> hm = new HashMap<String, String>();
                            //se agregan en el hasmap las coordenadas de la lista LatLng creada
                            hm.put("lat", Double.toString(((LatLng) list.get(l)).latitude));
                            hm.put("lng", Double.toString(((LatLng) list.get(l)).longitude));
                            path.add(hm);
                        }
                    }

                    routes.add(path);
                }
            }

        } catch (JSONException e) {
            e.printStackTrace();
        } catch (Exception e) {
        }

        return routes;
    }
```

La función **decodePoly** es una función dada por los servidores de Google para decodificar las polylines dadas por el URL. Estas vienen codificadas en un String y es necesaria esta función. Retorna una lista de LatLng:

```
private List<LatLng> decodePoly(String encoded) {

    List<LatLng> poly = new ArrayList<LatLng>();
    int index = 0, len = encoded.length();
    int lat = 0, lng = 0;
    //decodifica un string y crea un elemento LatLng
    while (index < len) {
        int b, shift = 0, result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lat += dlat;

        shift = 0;
        result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;

            shift += 5;
        } while (b >= 0x20);
        int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lng += dlng;

        LatLng p = new LatLng((((double) lat / 1E5)),
                (((double) lng / 1E5)));
        //se agrega el elemento LatLng a una lista que sera retornada al finalizar el proceso
        poly.add(p);
    }

    return poly;
}
```

La Función tipo Boolean **validacionOrigen** es usada cuando tanto el origen como el destino se encuentran en un cuadrante en el cual no existen caminos dinámicos, si ese es el caso, dibuja el camino estatico correspondiente y retorna False a la funcion principal el cual indicara que ya no debe crear mas rutas. En caso contrario mandara True y se validará despues en otra funcion si existen otros caminos estaticos.

```
//función que verifica si tanto el origen y destino se encuentran en un cuadrante donde no existen caminos dinamic
public Boolean validacionOrigen(LatLng origen, LatLng destino)
{
    PolylineOptions lineOptions = new PolylineOptions();
    double latOrigin=origen.latitude;
    double lngOrigin=origen.longitude;
    double latDestin=destino.latitude;
    double lngDestin=destino.longitude;
    //valor que retornara true si ambos, el origen y destino, no se encuentran en un cuadrante estatico
    Boolean validar=true;
    //se crea un arraylist de LatLng donde se cargaran coordenadas estaticas
    ArrayList<LatLng> CaminoOri=new ArrayList<LatLng>();
    //Si tanto el origen como el destino estan en un cuadrante estatico dibuja una polilinea y retorna false
    if((lngOrigin>-79.967939&&lngOrigin<-79.967206&&latOrigin>-2.147999&&latOrigin<-2.147494&&lngDestin>-79.967939&
        CaminoOri.add(origen);
        CaminoOri.add(destino);
        validar= false;
    }
    //se agrega el arraylist a un lineOptions
    lineOptions.addAll(CaminoOri);
    //se configura grosor y color de la linea a dibujar
    lineOptions.width(12);
    lineOptions.color(Color.rgb(0, 0, 255));
    //si el arraylist no esta vacio dibujara las polilineas al mapa
    if(lineOptions != null) {
        mMap.addPolyline(lineOptions);
    }
    //en la funcion principal si validar es false no dibujara mas lineas en el mapa ya que se dibujo
    //una polilinea entre el origen y destino y ya no es necesario dibujar lineas dinamicas
    return validar;
}
```

newLatln es una función que retorna una variable LatLng a la vez de que dibujara un camino estático dependiendo si la coordenadas recibidas están en algún cuadrante donde no existen caminos dinamicos. Esta función es llamada tanto para el origen como para el destino.

```
public LatLng newLatln(LatLng coord){
    double latOrigin=coord.latitude;
    double lngOrigin=coord.longitude;
    //arraylist de LatLng que sera la secuencia de las polilineas a dibujar
    ArrayList<LatLng> CaminoOri=new ArrayList<LatLng>();
    PolylineOptions lineOptions = new PolylineOptions();
    //se verifica si esta las coordenadas dentro de un cuadrante el cual no existe caminos dinamicos
    if(lngOrigin>-79.967939&&lngOrigin<-79.967206&&latOrigin>-2.147999&&latOrigin<-2.147494){
        CaminoOri.add(coord);
        CaminoOri.add(new LatLng(-2.147494,-79.967614));
        CaminoOri.add(new LatLng(-2.147250,-79.967597));
        //se cambia la coordenadas iniciales con una en donde si exite caminos dinamicos
        coord=new LatLng(-2.147250,-79.967597);
    }
    //se verifica si esta las coordenadas dentro de un cuadrante el cual no existe caminos dinamicos
    else if(lngOrigin>-79.967939&&lngOrigin<-79.967107&&latOrigin>-2.148723&&latOrigin<-2.147999){
        CaminoOri.add(coord);
        CaminoOri.add(new LatLng(-2.147999,-79.967495));
        CaminoOri.add(new LatLng(-2.147826,-79.967377));
        CaminoOri.add(new LatLng(-2.147494,-79.967614));
        CaminoOri.add(new LatLng(-2.147250,-79.967597));
        //se cambia la coordenadas iniciales con una en donde si exite caminos dinamicos
        coord=new LatLng(-2.147250,-79.967597);

    }
    //se verifica si esta las coordenadas dentro de un cuadrante el cual no existe caminos dinamicos
    else  if(lngOrigin>-79.968973&&lngOrigin<-79.967939&&latOrigin>-2.148278&&latOrigin<-2.147125){
        CaminoOri.add(coord);
        CaminoOri.add(new LatLng(-2.147337,-79.967939));
        CaminoOri.add(new LatLng(-2.147494,-79.967614));
        CaminoOri.add(new LatLng(-2.147250,-79.967597));
        //se cambia la coordenadas iniciales con una en donde si exite caminos dinamicos
        coord=new LatLng(-2.147250,-79.967597);
```

```
    }
    //se verifica si esta las coordenadas dentro de un cuadrante el cual no existe caminos dinamicos
    else  if(lngOrigin>-79.968571&&lngOrigin<-79.967903&&latOrigin>-2.147125&&latOrigin<-2.146125) {
        CaminoOri.add(coord);
        CaminoOri.add(new LatLng(-2.146614,-79.968077));
        CaminoOri.add(new LatLng(-2.146804,-79.967504));
        //se cambia la coordenadas iniciales con una en donde si exite caminos dinamicos
        coord=new LatLng(-2.146804,-79.967504);

    }
    //se agregan el arraylist a un lineOptions que dibujara las rutas estaticas que no existen
    lineOptions.addAll(CaminoOri);
    //se configura el grosor y color
    lineOptions.width(12);
    lineOptions.color(Color.rgb(0, 0, 255));
    if(lineOptions != null) {
        //se dibuja la polilinea en el mapa
        mMap.addPolyline(lineOptions);
    }
    return coord;
}
```

Para las dos funciones anteriores se crea rutas estaticas dependiendo de en que cuadrante se encuentra el origen o destino y cambiar sus coordenadas dependiendo si estan en esos cuadrantes como se puede apreciar en la **Figura 2.21**.

Figura 2.21: Cuadrantes estáticos

El metodo **voces(int dist)** es el metodo que reproducirá un sonido dependiendo de la distancia que tiene el usuario hacia su destino. Mientras se va acercando va reproduciendo el audio correspondiente.

```
public void voces(int dist){
    //funcion que reproducira un audio dependiendo de la distancia actual con el destino
    if(dist==100 && cont==0){
        //reproducira un audio cuando este el destino a 100 mts
        mp=MediaPlayer.create(this, R.raw.metros100);
        mp.start();
        cont++;
    }
    else if(dist==50&&cont==1){
        //reproducira un audio cuando se este a 50 mts y dependiendo del piso al cual se encuentre el destino
        if(getIntent().getExtras().getString("Piso").equals("PB")){
            mp=MediaPlayer.create(this, R.raw.destipb);
            mp.start();
            cont++;
        }
        else if(getIntent().getExtras().getString("Piso").equals("P1")){
            mp=MediaPlayer.create(this, R.raw.destip1);
            mp.start();
            cont++;
        }
        else if(getIntent().getExtras().getString("Piso").equals("P2")){
            mp=MediaPlayer.create(this, R.raw.destip2);
            mp.start();
            cont++;
        }
    }
    else if(dist==25&&cont==2){
        //reproducira un audio cuando se este a 25 mts
        mp=MediaPlayer.create(this, R.raw.destino);
        mp.start();

    }
```

```
    else if(dist<20){
        //cuando llegamos a una distancia menor a 20 se enviará a una activity Final con alguna informacion
        Intent intent = new Intent(MapsActivity.this,Final.class);
        int opcion = getIntent().getExtras().getInt("Tipo");
        intent.putExtra("Lugar","Destino: "+getIntent().getExtras().getString("Lugar"));
        intent.putExtra("Tipo",opcion);
        intent.putExtra("bloque",getIntent().getExtras().getInt("bloque"));
        //se envía información dependiendo del tipo de destino.
        switch (opcion){
            case 0:
                intent.putExtra("Latitud","Latitud: "+getIntent().getExtras().getDouble("Latitud"));
                intent.putExtra("Longitud","Longitud: "+getIntent().getExtras().getDouble("Longitud"));
                break;
            case 1:
                intent.putExtra("Codigo","Codigo: "+getIntent().getExtras().getString("Codigo"));
                intent.putExtra("Edificio","Bloque: "+getIntent().getExtras().getString("Edificio"));
                intent.putExtra("Piso","Piso: "+getIntent().getExtras().getString("Piso"));
                intent.putExtra("Facultad","Facultad: "+getIntent().getExtras().getString("Facultad"));
                break;
            case 2:
                intent.putExtra("Codigo","Oficina: "+getIntent().getExtras().getString("Codigo"));
                intent.putExtra("Edificio","Bloque: "+getIntent().getExtras().getString("Edificio"));
                intent.putExtra("Piso","Piso: "+getIntent().getExtras().getString("Piso"));
                intent.putExtra("Facultad","Facultad: "+getIntent().getExtras().getString("Facultad"));
                break;
        }
        startActivity(intent);
        finish();
    }
}
```

El metodo de la **Activity Maps** es el metodo **onBackPressed**, en este caso finalizara esta Activity y nos enviara a la **Activity Menu**:

```
public void onBackPressed() {
    Intent intent = new Intent(this, menu.class);
    startActivity(intent);
    finish();
}
```

2.4.5 Desarrollo de la página web.

El proyecto fue desarrollado con el fin de que nosotros, los desarrolladores, no tengamos que modificar la base de datos. Para eso se crea una página web que será manejada por personal administrativo de cada facultad. La información relevante que el personal administrativo debe modificar son los profesores y de las divisiones de los bloques.

Con referente a los profesores, se debe tomar en cuenta que solo se tomaron los profesores de la FIEC. Se debe agregar los profesores de las otras facultades, y en donde se encuentran sus oficinas.

Con respecto a la división de bloques, al igual que lo dicho anteriormente se optó por solo realizar las divisiones de los bloques de la FIEC, por lo que cada facultad debe agregar las divisiones de sus respectivos bloques.

INICIO

Es la página de bienvenida que mostrara al usuario un mensaje y parte de la visión y misión de la ESPOL. También una descripción de nuestra aplicación e información sobre los autores tal como indica la **Figura 2.22**.

Figura 2.22: Página de inicio

LA INSTITUCIÓN

Página que muestra información de la universidad, recomendaciones, y actividades complementarias que se pueden aplicar en la Universidad tal como indica la **Figura 2.23**.

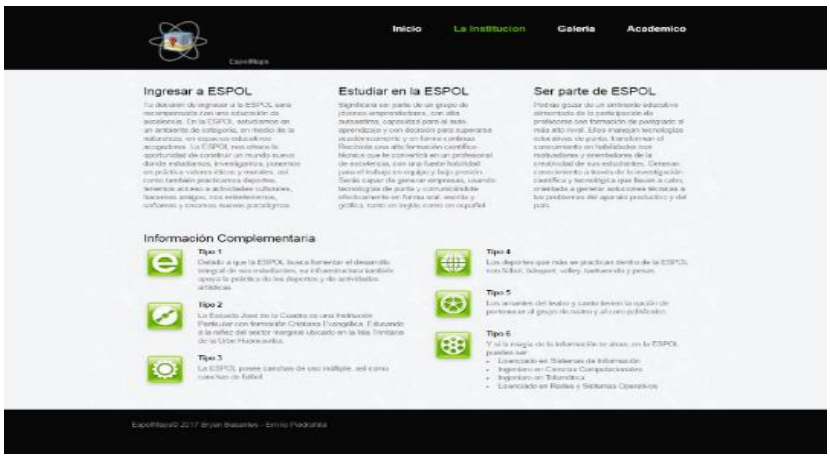

Figura 2.23: Página de información

GALERÍA

Muestra las diferentes imágenes subidas en nuestro servidor correspondiente al uso de nuestra aplicación tal como indica la **Figura 2.24**.

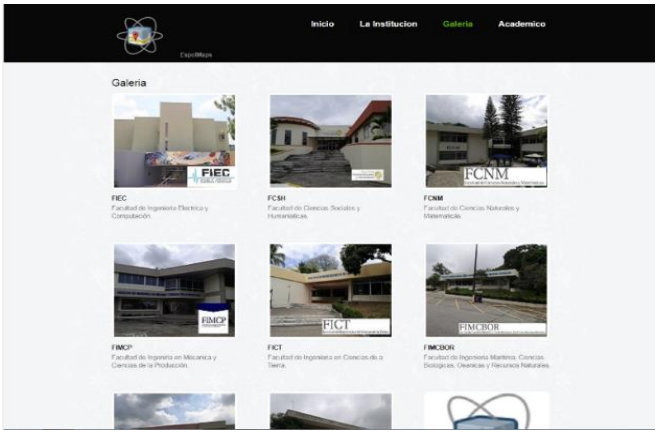

Figura 2.24: Página de Galería

LOGIN

Página de uso exclusivo para administradores de cada facultad. Es la sección encargada de modificar la base de datos de la aplicación tal como indica la **Figura 2.25**.

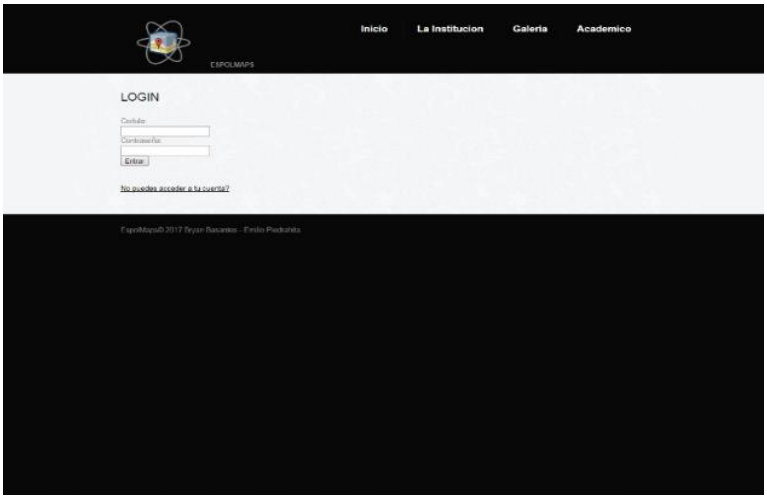

Figura 2.25: Página de administración

AGREGAR DATOS

Sección en donde se ingresarán nuevos datos a la base de datos. En nuestro caso se debe ingresar los profesores de las otras facultades, ya que solo se han ingresado los profesores de la FIEC tal como indica la **Figura 2.26**.

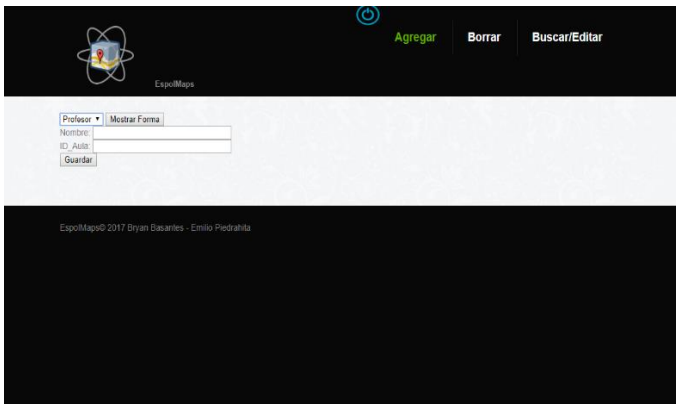

Figura 2.26: Página de agregar datos

BORRAR DATOS

Sección en donde se borrarán datos de la base de datos. Puede ser usada en el caso de que un profesor sea separado de la Universidad tal como indica la **Figura 2.27**.

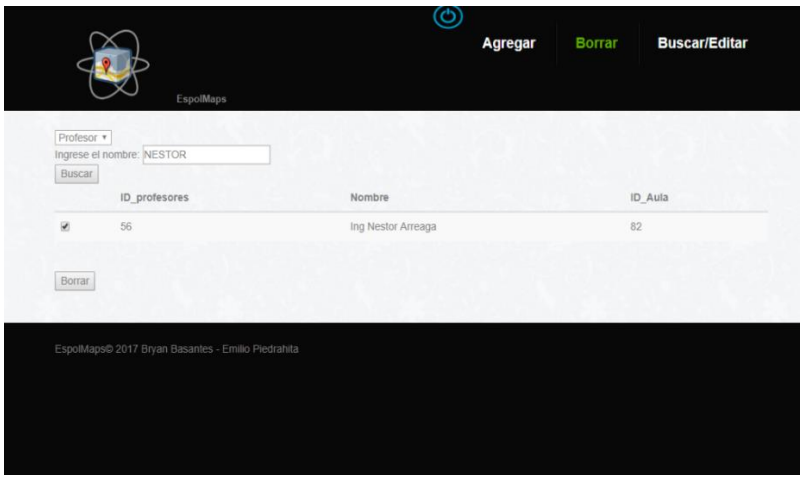

Figura 2.27: Página de borrar datos

BUSCAR/EDITAR

En esta sección se modificaran los datos de nuestra base de datos. Se modificará la información correspondiente a aulas, bloques y profesores.

En cada caso se hará la modificación según la facultad seleccionada. Después saldrán todos los datos y se deberá seleccionar el dato a modificar y por último se modifica la información obtenida tal como indica la **Figura 2.28**.

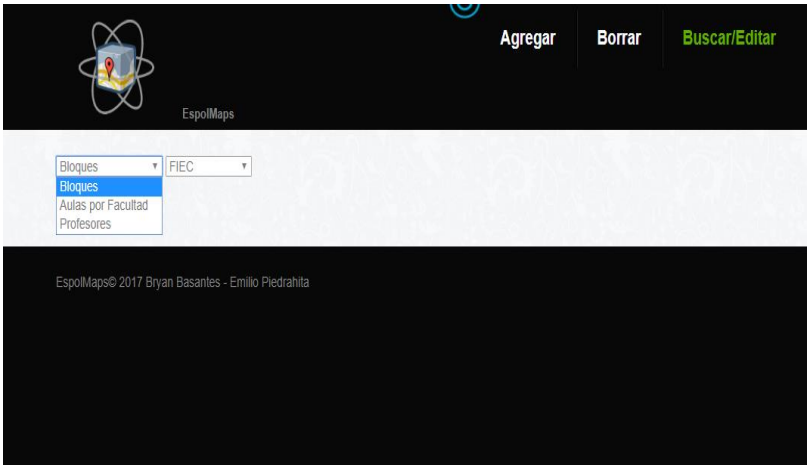

Figura 2.28: Página de Buscar/editar datos

Al presionar buscar con la opción **Bloques** saldrán todos los bloques pertenecientes a la respectiva facultad seleccionada. El administrador podrá seleccionar el bloque que desea modificar tal como indica la **Figura 2.29**.

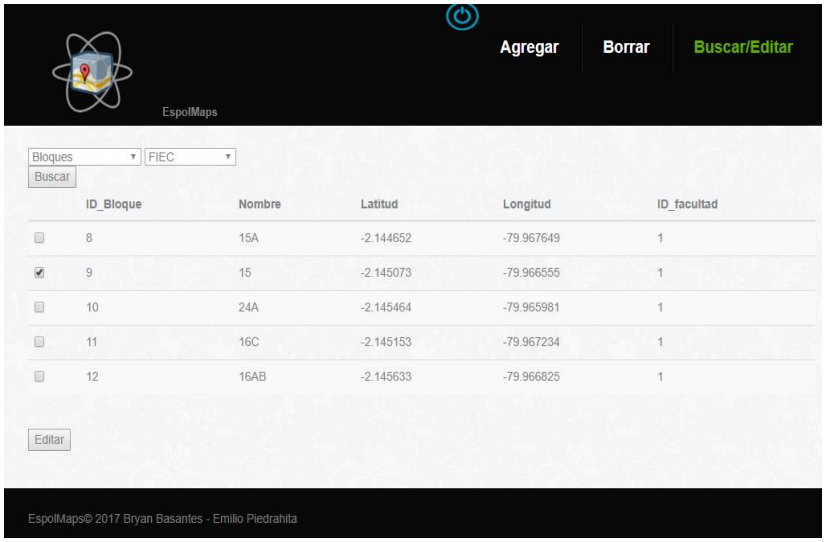

Figura 2.29: Buscando bloques dependiendo de Facultad

Al pulsar el botón editar nos enviara a una nueva ventana en la que se puede editar los campos mostrados tal como indica la **Figura 2.30**.

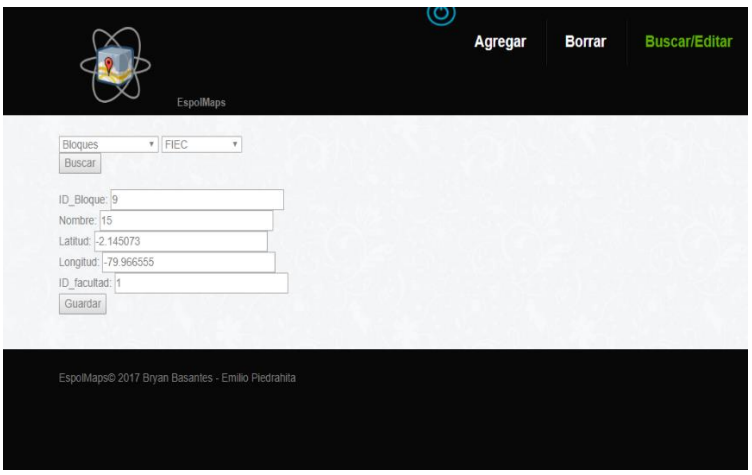

Figura 2.30: Editando Bloque seleccionado

Al presionar buscar con la opción **Aulas** saldrán todas las aulas u oficinas pertenecientes a la respectiva facultad seleccionada.

El administrador podrá seleccionar el aula que desea modificar tal como indica la **Figura 2.31**.

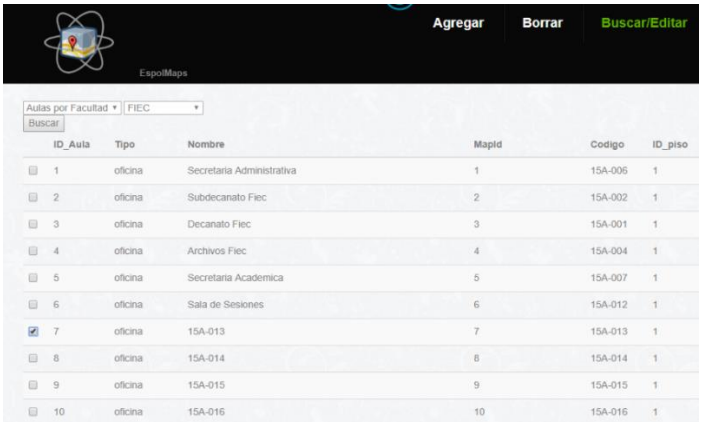

Figura 2.31: Buscando aulas por facultad

Al pulsar el botón editar nos enviara a una nueva ventana en la que se puede editar los campos mostrados tal como indica la **Figura 2.32**.

Figura 2.32: Editando Aula seleccionada

Al presionar buscar con la opción **Profesores** saldrán todos los profesores pertenecientes a la respectiva facultad seleccionada.

El administrador podrá seleccionar el aula que desea modificar tal como indica la **Figura 2.33**.

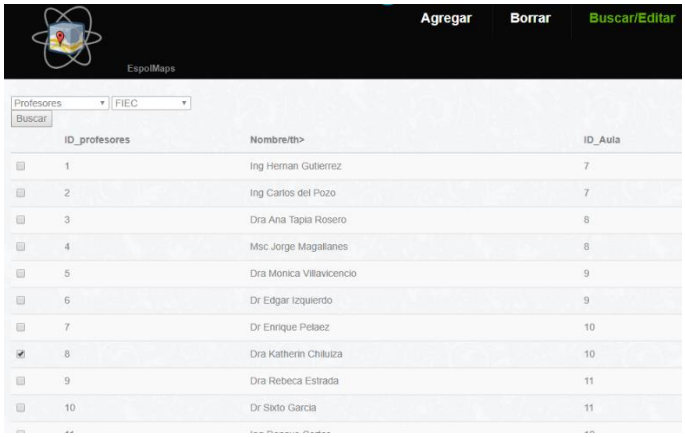

Figura 2.33: Buscando profesores por Facultad

Al pulsar el botón editar nos enviara a una nueva ventana en la que se puede editar los campos mostrados tal como indica la **Figura 2.34**.

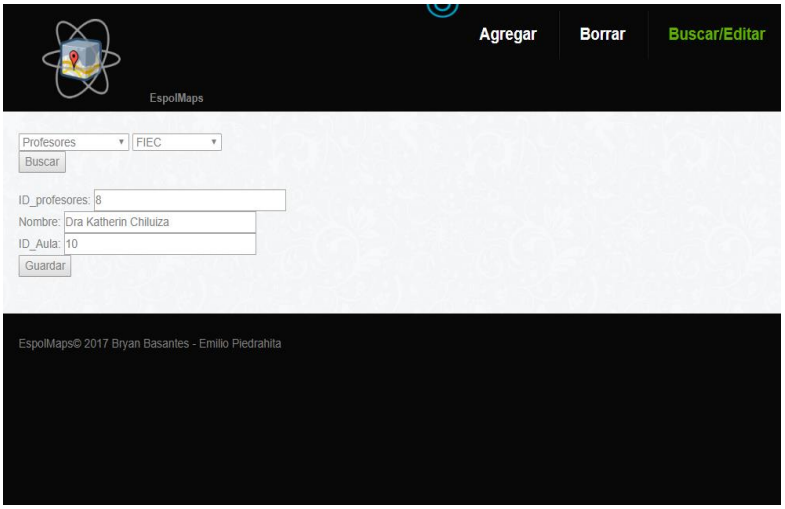

Figura 2.34: Editando Profesor Seleccionado

CAPÍTULO 3

3. RESULTADOS

Los siguientes resultados se laboraron con la ayuda de algunas personas que no habían estado en la Espol como una simulación a los estudiantes novatos, los cuales tienen el mismo problema, no conocer la ubicación de las distintas aulas, laboratorios u oficinas.

También se tomaron muestras de estudiantes que ya tienen cierta cantidad de semestres aprobados.

Para ambas poblaciones se realizó la toma de tiempo tanto sin la aplicación como con la aplicación.

Se tomó como muestra 20 personas sin usar la aplicación y 20 usando la aplicación. En ambas muestras hay 10 personas que estudian un tiempo en la universidad (Veteranos) y 10 que no han estado nunca en la universidad (Novatos). Cada persona realizó 3 rutas, al ser el tiempo el dato de estudio, en total tendremos una muestra de 60 sin usar la aplicación y 60 usando la aplicación. Las rutas son: Desde Sweet&Cofee hasta el Aula 24E-104 ubicado en el Bloque 24E de la facultad FIMCP indicado en la **Figura 3.1**:

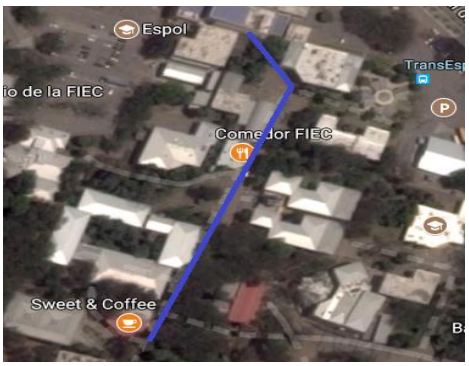

Figura 3.1 Ruta 1 hacia el bloque 24E

Desde el Bloque 24E hasta el aula A-24 ubicado en el bloque 20F de la facultad FICT indicado en la **Figura 3.2**:

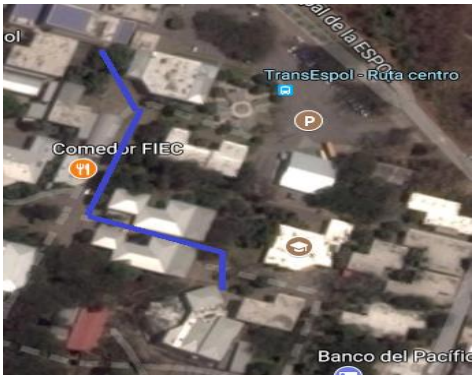

Figura 3.2 Ruta 2 hacia el bloque 20F

Desde el Bloque 20F hasta el Laboratorio de Electrónica B ubicado en el bloque 16AB de la facultad FIEC indicada en la **Figura 3.3**

Figura 3.3 Ruta 3 hacia el bloque 16AB

3.1 Tiempos de recorridos hacia las aulas sin la aplicación.

Para esta prueba se tomó muestra de 2 poblaciones, estudiantes de tres a cuatro semestres aprobados y de personas que nunca han conocido la universidad simulando a estudiantes novatos.

La muestra de estudiantes veteranos fue tomada de las facultades FCSH, FCNM y FIEC.

3.1.1 Muestra de personas con varios semestres aprobados

Estadística descriptiva:

Población: Estudiantes veteranos

Muestra: 10 estudiantes, 3 rutas (30)

Datos de estudio: Tiempo de llegada a un Aula.

Tiempo(minutos)				
5	3	6	8	10
6	9	9	4	14
4	10	11	9	11
8	14	6	6	4
6	12	5	3	8
4	14	5	7	7

Tabla 1: Muestra de veteranos en minutos S/A

Ampliación: Vmax-Vmin=14-3=11 (3.1)

Numero de Clases=$\sqrt{}$**N=**$\sqrt{}$**30=**5.5≈6 (3.2)

Tamaño de Clase=Ampliación/NumeroClases=11/6≈2 (3.3)

k	clase	Frecuencia absoluta	Frecuencia relativa	Frecuencia Absoluta Acumulada	Frecuencia Relativa acumulada
1	[3,5)	6	0.2	6	0.2
2	[5,7)	8	0.267	14	0.467
3	[7,9)	5	0.167	19	0.633
4	[9,11)	5	0.167	24	0.8
5	[11,13)	3	0.1	27	0.9
6	[13,15]	3	0.1	30	1

Tabla 2: Tabla de frecuencia de veteranos S/A

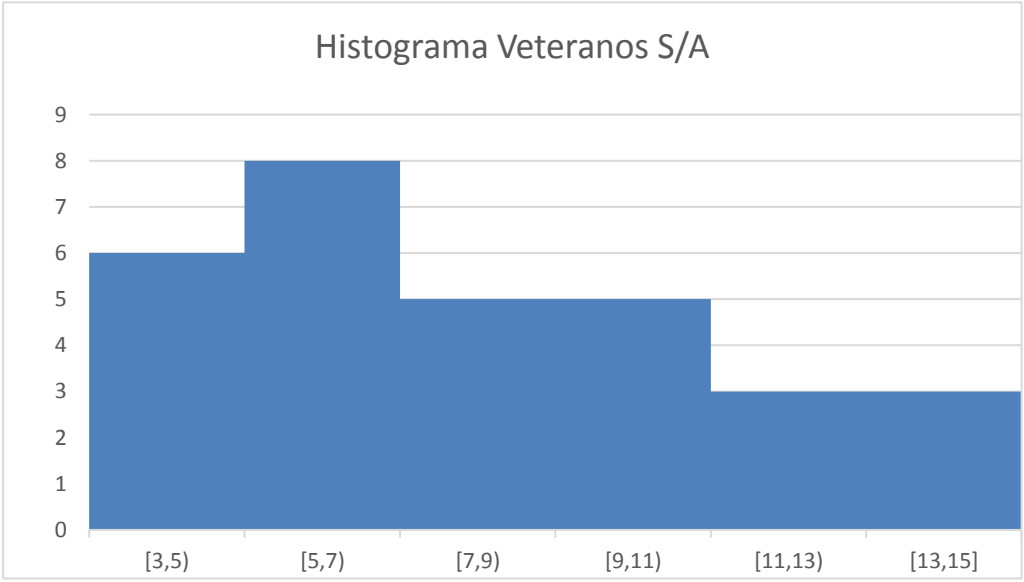

Figura 3.4: Histograma de veteranos S/A

$$\textbf{Media=}\frac{1}{n}\sum_{i=0}^{n} X_i=\frac{228}{30}\textbf{=7.6 min} \qquad (3.4)$$

$$\boldsymbol{\sigma^2 = \frac{1}{n-1} \sum_{i=0}^{n} (X_i - X) = 10.73} \quad (3.5)$$

3.1.2 Muestra de personas novatas

Estadística descriptiva:

Población: Personas que no conocen la Espol

Muestra: 10 estudiantes, 3 rutas (30)

Datos de estudio: Tiempo de llegada a un Aula.

Tiempo(minutos)				
15	23	10	12	14
24	6	27	9	11
26	8	16	9	32
7	13	15	19	16
10	31	11	17	7
14	21	23	27	15

Tabla 3: Muestra de novatos en minutos S/A

Ampliación: Vmax-Vmin=32-6=26 (3.6)

Numero de Clases=$\sqrt{}$**N=**$\sqrt{}$**30=**5.5≈6 (3.7)

Tamaño de Clase=Ampliación/NumeroClases=26/6≈5 (3.8)

k	Clase	Frecuencia absoluta	Frecuencia relativa	Frecuencia Absoluta Acumulada	Frecuencia Relativa acumulada
1	[6,11)	8	0.267	8	0.267
2	[11,16)	9	0.300	17	0.567
3	[16,21)	4	0.133	21	0.700
4	[21,26)	4	0.133	25	0.833
5	[26,31)	3	0.100	28	0.933
6	[31,36)	2	0.067	30	1.000

Tabla 4: Tabla de frecuencia de novatos S/A

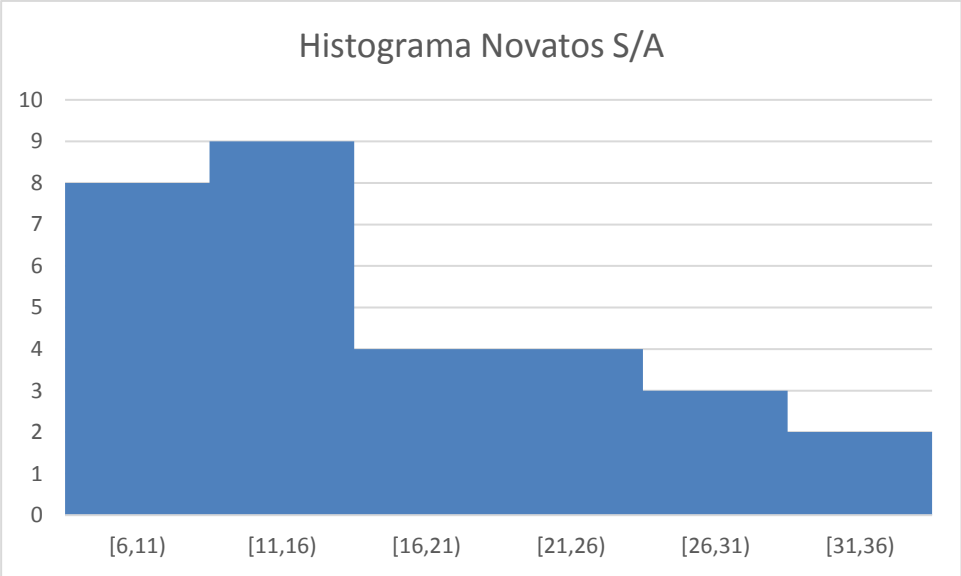

Figura 3.5: Histograma de novatos S/A

$$\text{Media}=\frac{1}{n}\sum_{i=0}^{n} X_i=\frac{488}{30}= \textbf{16.27 min} \qquad (3.9)$$

$$\boldsymbol{\sigma^2 = \frac{1}{n-1} \sum_{i=0}^{n} (X_i - X) = 55.17} \quad (3.10)$$

3.1.3 Muestra Total

Se unirán las dos muestras anteriores para determinar una población general de estudiantes de la Espol.

Estadística descriptiva:

Población: Estudiantes en General.

Muestra: 20 estudiantes, 3 rutas (60)

Datos de estudio: Tiempo de llegada a un Aula.

Tiempo(minutos)				
5	3	6	8	10
6	9	9	4	14
4	10	11	9	11
8	14	6	6	4
6	12	5	3	8
4	14	5	7	7
15	23	10	12	14
24	6	27	9	11
26	8	16	9	32
7	13	15	19	16
10	31	11	17	7
14	21	23	27	15

Tabla 4: Muestra General en minutos S/A

Ampliación: Vmax-Vmin=32-3=29 (3.11)

Numero de Clases=√N=√60=7.75≈8 (3.12)

Tamaño de Clase=Ampliación/NumeroClases=29/8≈4 (3.13)

k	Clase (minutos)	Frecuencia absoluta	Frecuencia relativa	Frecuencia Absoluta Acumulada	Frecuencia Relativa acumulada
1	[3,7)	15	0.250	15	0.250
2	[7,11)	17	0.283	32	0.533
3	[11,15)	12	0.200	44	0.733
4	[15,19)	6	0.100	50	0.833
5	[19,23)	2	0.033	52	0.867
6	[23,27]	4	0.067	56	0.933
7	[27,31)	2	0.033	58	0.967
8	[31,35)	2	0.033	60	1.000

Tabla 5: Tabla de frecuencia general S/A

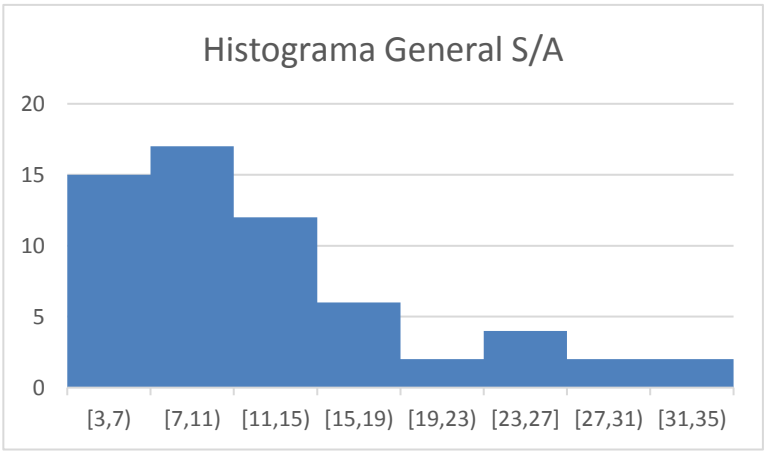

Figura 3.6: Histograma general S/A

$$\textbf{Media}=\frac{1}{n}\sum_{i=0}^{n} X_i=\frac{716}{60}=\textbf{11.93 min} \quad (3.14)$$

$$\sigma^2=\frac{1}{n-1}\sum_{i=0}^{n}(X_i-X)=104.75 \quad (3.15)$$

3.2 Tiempos de recorridos hacia las aulas con la aplicación.

Para esta prueba a la igual q la anterior se tomó muestra de 2 poblaciones, estudiantes de tres a cuatro semestres aprobados y de personas que nunca han conocido la universidad simulando a estudiantes novatos.

La muestra de estudiantes veteranos fue tomada de las facultades FCSH, FCNM y FIEC.

3.2.1 Muestra de personas con varios semestres aprobados

Estadística descriptiva:

Población: Estudiantes veteranos

Muestra: 10 estudiantes, 3 rutas (30)

Datos de estudio: Tiempo de llegada a un Aula.

Tiempo(minutos)				
3	2	3	3	5
4	3	2	2	5
6	3	5	3	2
3	5	7	3	4
5	4	2	4	3
2	4	4	2	3

Tabla 6: Muestra de veteranos en minutos C/A

Ampliación: Vmax-Vmin=7-2=5 (3.16)

Numero de Clases=√N=√30=5.5≈6 (3.17)

Tamaño de Clase=Ampliación/NumeroClases=5/6≈1 (3.18)

k	Clase	Frecuencia Absoluta	Frecuencia Relativa	Frecuencia Absoluta Acumulada	Frecuencia Relativa Acumulada
1	[2,3)	7	0.233	7	0.233
2	[3,4)	10	0.333	17	0.567
3	[4,5)	6	0.200	23	0.767
4	[5,6)	5	0.167	28	0.933
5	[6,7)	1	0.033	29	0.967
6	[7,8)	1	0.033	30	1.000

Tabla 7: Tabla de frecuencia de veteranos C/A

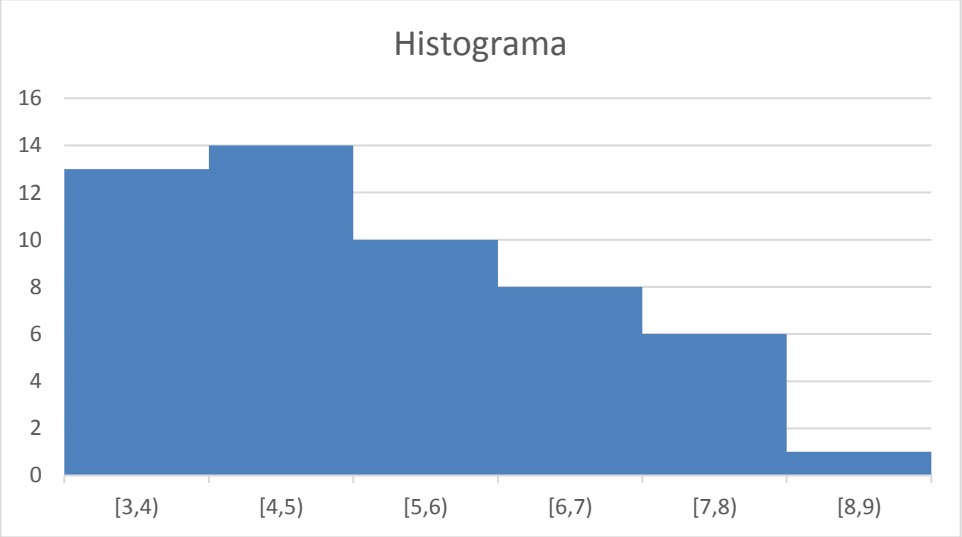

Figura 3.7: Histograma de veteranos C/A

Media=$\frac{1}{n}\sum_{i=0}^{n} X_i = \frac{106}{30}$**= 3.53 min** (3.19)

$$\boldsymbol{\sigma^2 = \frac{1}{n-1}\sum_{i=0}^{n}(X_i - X) = 1.71}$$ (3.20)

3.2.2 Muestra de personas novatas

Población: Personas que no conocen la Espol

Muestra: 10 estudiantes, 3 rutas (30)

Datos de estudio: Tiempo de llegada a un Aula.

Tiempo(minutos)				
4	3	4	6	3
5	5	7	6	4
4	3	6	5	6
6	6	8	4	7
4	6	9	7	5
7	5	7	4	4

Tabla 8: Muestra de novatos en minutos C/A

Ampliación: Vmax-Vmin=9-3=6 (3.21)

Numero de Clases=√**N=**√**30=**5.5≈6 (3.22)

Tamaño de Clase=Ampliación/NumeroClases=6/6≈1 (3.23)

K	Clase	Frecuencia Absoluta	Frecuencia Relativa	Frecuencia Absoluta Acumulada	Frecuencia Relativa Acumulada
1	[3,4)	3	0.100	3	0.100
2	[4,5)	8	0.267	11	0.367
3	[5,6)	5	0.167	16	0.533
4	[6,7)	7	0.233	23	0.767
5	[7,8)	5	0.167	28	0.933
6	[8,9]	2	0.067	30	1.000

Tabla 9: Tabla de frecuencia de novatos C/A

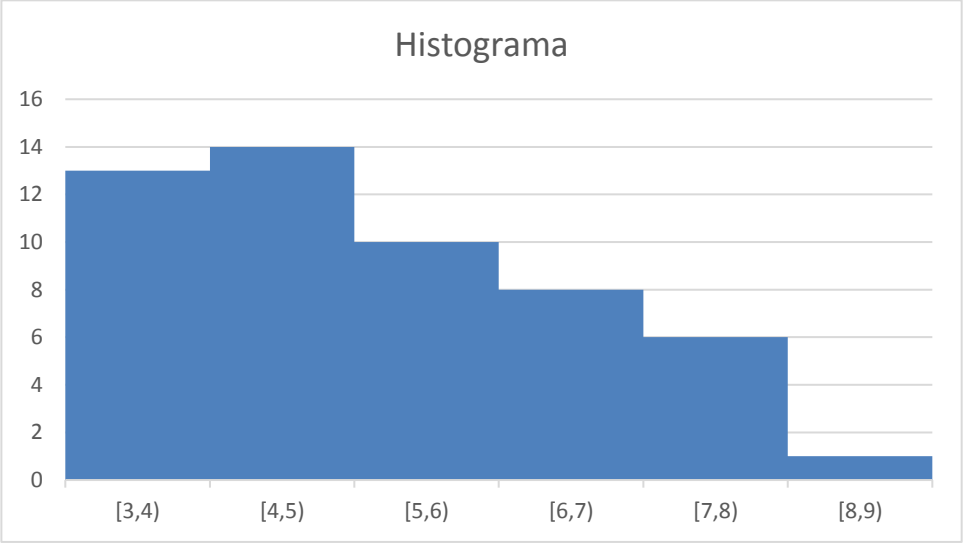

Figura 3.8: Histograma de Novatos C/A

$$\text{Media}=\frac{1}{n}\sum_{i=0}^{n} X_i=\frac{160}{30}= \textbf{5.33 min} \quad (3.24)$$

$$\sigma^2=\frac{1}{n-1}\sum_{i=0}^{n}(X_i-X)=2.37 \quad (3.25)$$

3.2.3 Muestra Total

Población: Estudiantes en General.

Muestra: 20 estudiantes, 3 rutas (60)

Datos de estudio: Tiempo de llegada a un Aula.

Tiempo(minutos)				
3	2	3	3	5
4	3	2	2	5
6	3	5	3	2
3	5	7	3	4
5	4	2	4	3
2	4	4	2	3
4	3	4	6	3
5	5	7	6	4
4	3	6	5	6
6	6	8	4	7
4	6	9	7	5
7	5	7	4	4

Tabla 10: Muestra General en minutos C/A

Ampliación: Vmax-Vmin=9-2=7 (3.26)

Numero de Clases=√N=√60=7.75≈8 (3.27)

Tamaño de Clase=Ampliación/NumeroClases=7/8≈1 (3.28)

k	Clase	Frecuencia Absoluta	Frecuencia Relativa	Frecuencia Absoluta Acumulada	Frecuencia Relativa Acumulada
1	[2,3)	7	0.117	7	0.117
2	[3,4)	13	0.217	20	0.333
3	[4,5)	14	0.233	34	0.567
4	[5,6)	10	0.167	44	0.733
5	[6,7)	8	0.133	52	0.867
6	[7,8)	6	0.100	58	0.967
7	[8,9)	1	0.017	59	0.983
8	[9,10)	1	0.017	60	1.000

Tabla 11: Tabla de frecuencia General C/A

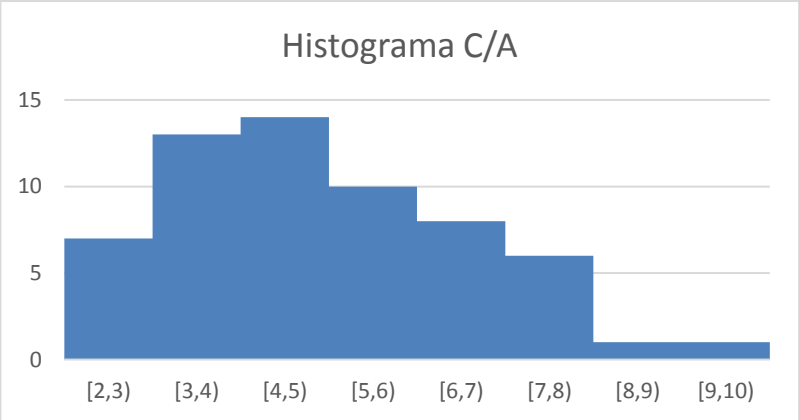

Figura 3.9: Histograma General C/A

$$\text{Media}=\frac{1}{n}\sum_{i=0}^{n} X_i=\frac{266}{60}= \textbf{4.43 min} \tag{3.29}$$

$$\sigma^2=\frac{1}{n-1}\sum_{i=0}^{n}(X_i-X)=5.75 \tag{3.30}$$

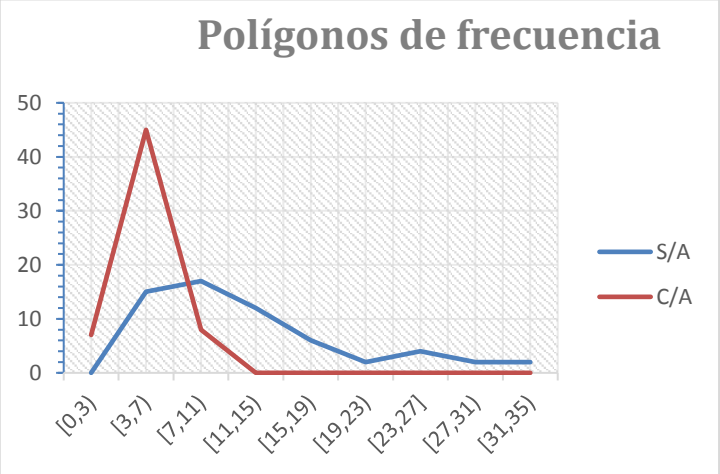

Figura 3.10: Polígono de frecuencia C/A y S/A

En la figura 3.10 podemos observar como los tiempos de las personas que usan la aplicación se concentran más en intervalos de tiempos más pequeños, mientras que en las personas que no la usan los tiempos son dispersos y llegan a tiempos muy grandes.

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS

Las comparaciones serán entre los tiempos obtenidos por:

- Estudiantes veteranos y los novatos sin utilizar la aplicación.
- Estudiantes veteranos y novatos utilizando la aplicación.
- Estudiantes veteranos que no usaron la aplicación con los que si la usaron
- Estudiantes novatos que no usaron la aplicación con los que si la usaron
- Todos los estudiantes que no usaron la aplicación y los que si la usaron.

Los cálculos estadísticos se encuentran en la parte de **ANEXOS.**

4.1 Comparación referente a los tiempos obtenidos.

4.1.1 Estudiantes veteranos y novatos sin utilizar la aplicación.

Para todas las comparaciones de cada muestra primero analizamos las varianzas obtenidas. Las varianzas determinan el promedio de la diferencia de los valores respecto a su media. También se comparó el valor de las medias.

Con respecto a ese punto observamos que la varianza de los novatos es demasiado elevada con respecto a la de los veteranos. Esto se debe porque los valores obtenidos contienen muchos valores aberrantes, es decir que tienen una gran diferencia con su media. Por ejemplo, tenemos que la media del tiempo de los novatos es de 16 minutos y hay datos que son o el doble o menos de la mitad a este (32 min y 6 min).

La varianza de las personas veteranas es mucho menor al de las personas novatas. Esto es porque las personas novatas tienen mayor noción de la ubicación de las facultades.

La media de las personas veteranas es menor a la media de las personas novatas. Siendo la misma razón expuesta en el párrafo anterior.

Se puede concluir con estos resultados que una persona veterana llegara a un destino dentro de la ESPOL en menor tiempo que una persona novata. Ambas sin usar la aplicación.

4.1.2 Estudiantes veteranos y novatos utilizando la aplicación.

Analizando las varianzas podemos ver que los valores son muy cercanos por lo que se puede deducir que las diferencias con respecto a sus medias son iguales. Esto es un punto que puede indicarnos que ambas muestras son iguales y que los tiempos que le toma a un novato y un veterano serán las mismas cuando usan la aplicación.

La varianza de los veteranos tiene una varianza de 1.71 mientras que la de los novatos es de 2.37. Son muy semejantes.

La media de las personas veteranas sigue siendo menor a la media de las personas novatas, pero a diferencia del punto anterior, la diferencia es mínima. La de los veteranos es de 3.53 min mientras que los de los novatos es de 5.33 min.

Aun así, se sigue determinando que la persona veterana llegara en menor tiempo que una persona novata, usando ambas la aplicación.

4.1.3 Estudiantes veteranos que no usaron la aplicación con los que si la usaron

En esta comparación determinaremos si existe un cambio significativo con los alumnos veteranos que no usan la aplicación con los que si la usaron.

Las varianzas entre las dos muestras son muy distantes. Mientras la que no usaron la aplicación es de 10.73, la de los que si la usaron es de 1.71 siendo 10 veces menor.

La media de las personas veteranas usando la aplicación es menor al de los que si la usaron. Mientras que los que no usaron la aplicación su media es de 7.6 min mientras que los que si la usaron su media es de 3.53 min.

Podemos concluir que el tiempo que le toma a una persona veterana que usa la aplicación es menor al tiempo de una persona veterana que no la usa.

4.1.4 Estudiantes novatos que no usaron la aplicación con los que si la usaron

En esta comparación determinaremos si existe un cambio significativo con los alumnos novatos que no usan la aplicación con los que si la usaron.

Las varianzas entre las dos muestras son muy distantes. Mientras la que no usaron la aplicación es de 55.17, la de los que si la usaron es de 2.37 siendo cerca de 23 veces menor.

La media de las personas usando la aplicación es menor al de los que si la usaron. Mientras que los que no usaron la aplicación su media es de 11.93 min mientras que los que si la usaron su media es de 5.33 min.

Podemos concluir que el tiempo que le toma a una persona novata que usa la aplicación es menor al tiempo de una

persona novata que no la usa.

4.1.5 Todos los estudiantes que no usaron la aplicación y los que si la usaron

En esta comparación determinaremos si existe un cambio significativo con los todos alumnos que no usan la aplicación con los que si la usaron.

Las varianzas entre las dos muestras son muy distantes. Mientras la que no usaron la aplicación es de 104.75, la de los que si la usaron es de 5.75 siendo cerca de 18 veces menor.

La media de las personas veteranas usando la aplicación es menor al de los que si la usaron. Mientras que los que no usaron la aplicación su media es de 16.27 min mientras que los que si la usaron su media es de 4.43 min.

Podemos concluir que el tiempo que le toma a una persona en general que usa la aplicación es menor al tiempo de una persona que no la usa.

4.2 Observación de mejoras en la aplicación.

4.2.1 Relevo de cargo en llenado de base de datos

Una de nuestras metas es dejar el proyecto para el uso exclusivo de la universidad, y al hacer eso lo realizamos de tal manera que el proyecto no necesite actualizar su código fuente, sino la información de la base de datos creada.

Para cumplir con este objetivo se realiza una página web de la cual podrán acceder personal administrativo de cada facultad y modificar la base de datos y coloque la oficina de cada profesor de su facultad ya que ellos saben mejor que nadie en qué oficina se encontrará el profesor.

Así mismo para los profesores de la FIEC, de los cuales su información se encuentra en la base de datos, se pueda

modificar si hay un caso de reubicación de oficina.

4.2.2 Relevo de cargo en divisiones de bloques

Para el caso de las divisiones de los bloques, hemos optado de que las imágenes correspondientes a los bloques divididos sean descargadas en segundo plano del servidor. Por lo que, para actualizar un bloque en sus divisiones, basta con modificar la imagen correspondiente dentro del servidor.

El personal administrativo de cada facultad tendrá permisos para poder acceder a realizar esos cambios. Los pisos son mostrados en la **Figura 4.1**

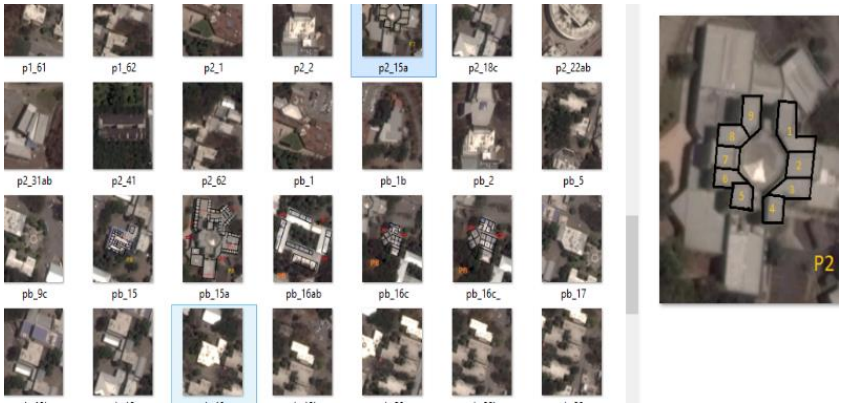

Figura 4.1: Pisos divididos

CONCLUSIONES Y RECOMENDACIONES

Al observar los resultados obtenidos en el capítulo 3 y haberlos analizados en el capítulo 4 podemos concluir:

El tiempo que tarda un estudiante novato siempre es mayor al tiempo que demora un estudiante veterano, sin importar si use la aplicación o no.

El tiempo que tarda un estudiante novato y uno veterano usando la aplicación tienden a tener una variación idéntica con respecto a su media. Esto quiere decir que, la mayoría de personas obtendrán un resultado muy cercano a la media.

El tiempo que tarda una persona usando la aplicación es significativamente menor al tiempo de una persona que no la usa.

Como recomendación, se observa que las personas no encuentran las diferentes aulas es por la falta de identificación en los edificios. Se recomienda identificar de mejor manera los edificios y aulas para la comodidad del estudiante.

BIBLIOGRAFÍA

[1] CSIC, C. S. (1 de Enero de 2017). *Ranking Web de Universidades*. Obtenido de http://www.webometrics.info/es/Latin_America_es/Ecuador

[2] CSI, C. d. (1 de Enero de 2017). *Campus de la Espol*. Obtenido de http://www.espol.edu.ec/espol/main.jsp?urlpage=campus.jsp&campus=1

[3] Ardións, A. (16 de 05 de 2016). *Android Studio: Requisitos mínimos*. Obtenido de https://androidstudiofaqs.com/conceptos/android-studio-requisitos-minimos

[4] Devolper, A. (1 de Marzo de 2017). *Funciones de Android Studio*. Obtenido de https://developer.android.com/studio/features.html?hl=es-419

[5] affiliates, O. C. (1 de Enero de 2017). *Why MySQL?* Obtenido de https://www.mysql.com/why-mysql/

[6] Tatham, S. (1 de Enero de 2017). *Download PuTTY - a free SSH and Telnet Client for Windows*. Obtenido de http://www.putty.org/

ANEXOS

Los cálculos señalados en esta sección son pertenecientes al capítulo 4.

- **Comparación referente a los tiempos obtenidos.**
  - **Estudiantes veteranos y novatos sin utilizar la aplicación.**

Para este caso nos planteamos la siguientes Hipótesis con respecto a sus varianzas:

$$\boldsymbol{Ho}: \boldsymbol{\sigma 1^2 = \sigma 2^2} \quad (4.1)$$

$$\boldsymbol{H_1}: \boldsymbol{\sigma 1^2 \neq \sigma^2} \quad (4.2)$$

Si Ho es cierta:

$$\boldsymbol{\sigma 1^2 \text{ será parecida a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \approx 1} \quad (4.3)$$

Si Ho es falsa:

$$\boldsymbol{\sigma 1^2 \text{ difieren mucho a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \neq 1} \quad (4.4)$$

$$\boldsymbol{\frac{\sigma 1^2}{\sigma 2^2} = \frac{55.17}{10.73} = 5.24 \neq 1}$$

Se rechaza Ho puesto que sus varianzas difieren mucho.

Ahora plantearemos las siguientes hipótesis con respecto a sus medias:

$$\boldsymbol{Ho}: \boldsymbol{\mu 1 = \mu 2} \quad (4.5)$$

$$\boldsymbol{H_1}: \boldsymbol{\mu 1 \neq \mu 2} \quad (4.6)$$

Si Ho es cierta:

$$\boldsymbol{\mu 1 \text{ será parecida a } \mu 2 \rightarrow \mu 1 - \mu 2 \approx 0} \quad (4.7)$$

Si Ho es falsa:

$$\boldsymbol{\mu 1 \text{ difieren mucho a } \mu 2 \rightarrow \mu 1 - \mu 2 \neq 0} \quad (4.8)$$

$$\boldsymbol{\mu 1 - \mu 2 = 7.6 - 16.27 = \ -8.67 \neq 0}$$

Se Rechaza la Ho puesto que las medias difieren mucho.

- **Estudiantes veteranos y novatos utilizando la aplicación.**

$$\boldsymbol{Ho}: \boldsymbol{\sigma 1^2 = \sigma 2^2} \quad (4.9)$$

$$\boldsymbol{H_1}: \boldsymbol{\sigma 1^2 \neq \ \sigma^2} \quad (4.10)$$

Si Ho es cierta:

$$\boldsymbol{\sigma 1^2 \text{ será parecida a } \sigma 2^2 \ \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \approx 1} \quad (4.11)$$

Si Ho es falsa:

$$\boldsymbol{\sigma 1^2 \text{ difieren mucho a } \sigma 2^2 \ \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \neq 1} \quad (4.12)$$

$$\boldsymbol{\frac{\sigma 1^2}{\sigma 2^2} = \frac{2.37}{1.71} = 1.39 \approx 1}$$

TEST F DE COMPARACIÓN DE VARIANZA

$$\boldsymbol{\frac{\sigma 1^2}{\sigma 2^2} \sim F_{N1-1,N2-1}} \quad (4.13)$$

Si $\boldsymbol{\frac{\sigma 1^2}{\sigma 2^2} < F_{N1-1,N2-1}}$ **Aceptamos Ho** (4.14)

Si $\boldsymbol{\frac{\sigma 1^2}{\sigma 2^2} > F_{N1-1,N2-1}}$ **Rechazamos Ho** (4.15)

$$\boldsymbol{F_{N1-1,N2-1}(\alpha = 1.6)}$$

$$\boldsymbol{1.39 < 1.6}$$

Aceptamos la hipótesis Ho.

Ahora plantearemos las siguientes hipótesis con respecto a sus medias:

$$\boldsymbol{Ho}: \boldsymbol{\mu 1 = \mu 2} \quad (4.16)$$

$$\boldsymbol{H_1}: \boldsymbol{\mu 1 \neq \mu 2} \quad (4.17)$$

Si Ho es cierta:

$$\boldsymbol{\mu 1 \text{ será parecida a } \mu 2 \rightarrow \mu 1 - \mu 2 \approx 0} \quad (4.18)$$

Si Ho es falsa:

$$\boldsymbol{\mu 1 \text{ difieren mucho a } \mu 2 \rightarrow \mu 1 - \mu 2 \neq 0} \quad (4.19)$$

$$\frac{\boldsymbol{\mu 1 - \mu 2}}{\boldsymbol{\sigma^2\sqrt{\frac{1}{N1}+\frac{1}{N2}}}} \sim \boldsymbol{t^{\alpha}_{(N1-1)+(N2-1)}} \quad (4.20)$$

$$\frac{\boldsymbol{\mu 1 - \mu 2}}{\boldsymbol{\sigma^2\sqrt{\frac{1}{N1}+\frac{1}{N2}}}} = \frac{\boldsymbol{5.33 - 3.53}}{\boldsymbol{2.04\sqrt{\frac{1}{15}}}} = \boldsymbol{3,41}$$

$$\boldsymbol{t_{58} = 1.67}$$

Como $3,41 > 1.67$ Rechazamos Ho.

- **Estudiantes veteranos que no usaron la aplicación con los que si la usaron**

  Para este caso nos planteamos la siguientes Hipótesis con respecto a sus varianzas:

  $$\boldsymbol{Ho}: \boldsymbol{\sigma 1^2 = \sigma 2^2} \quad (4.21)$$

  $$\boldsymbol{H_1}: \boldsymbol{\sigma 1^2 \neq \sigma^2} \quad (4.22)$$

  Si Ho es cierta:

  $$\boldsymbol{\sigma 1^2 \text{ será parecida a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \approx 1} \quad (4.23)$$

  Si Ho es falsa:

  $$\boldsymbol{\sigma 1^2 \text{ difieren mucho a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \neq 1} \quad (4.24)$$

$$\frac{\sigma 1^2}{\sigma 2^2} = \frac{10.73}{1.71} = 6.27 \neq 1$$

Se rechaza Ho puesto que sus varianzas difieren mucho.

Ahora plantearemos las siguientes hipótesis con respecto a sus medias:

$$Ho: \mu 1 = \mu 2 \quad (4.25)$$

$$H_1: \mu 1 \neq \mu 2 \quad (4.26)$$

Si Ho es cierta:

$$\mu 1 \text{ será parecida a } \mu 2 \rightarrow \mu 1 - \mu 2 \approx 0 \quad (4.27)$$

Si Ho es falsa:

$$\mu 1 \text{ difieren mucho a } \mu 2 \rightarrow \mu 1 - \mu 2 \neq 0 \quad (4.28)$$

$$\mu 1 - \mu 2 = 7.6 - 3.53 = 4.07 \neq 0$$

Se Rechaza la Ho puesto que las medias difieren mucho.

- **Estudiantes novatos que no usaron la aplicación con los que si la usaron**

Para este caso nos planteamos la siguientes Hipótesis con respecto a sus varianzas:

$$Ho: \sigma 1^2 = \sigma 2^2 \quad (4.29)$$

$$H_1: \sigma 1^2 \neq \sigma^2 \quad (4.30)$$

Si Ho es cierta:

$$\sigma 1^2 \text{ será parecida a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \approx 1 \quad (4.31)$$

Si Ho es falsa:

$$\sigma 1^2 \text{ difieren mucho a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \neq 1 \quad (4.32)$$

$$\frac{\sigma 1^2}{\sigma 2^2} = \frac{55.17}{2.37} = 23.28 \neq 1$$

Se rechaza Ho puesto que sus varianzas difieren mucho.

Ahora plantearemos las siguientes hipótesis con respecto a sus medias:

$$Ho: \mu 1 = \mu 2 \quad (4.33)$$

$$H_1: \mu 1 \neq \mu 2 \quad (4.34)$$

Si Ho es cierta:

$$\mu 1 \text{ será parecida a } \mu 2 \rightarrow \mu 1 - \mu 2 \approx 0 \quad (4.35)$$

Si Ho es falsa:

$$\mu 1 \text{ difieren mucho a } \mu 2 \rightarrow \mu 1 - \mu 2 \neq 0 \quad (4.36)$$

$$\mu 1 - \mu 2 = 16.27 - 5.33 = 10.94 \neq 0$$

Se Rechaza la Ho puesto que las medias difieren mucho.

- **Todos los estudiantes que no usaron la aplicación y los que si la usaron**

Para este caso nos planteamos la siguientes Hipótesis con respecto a sus varianzas:

$$Ho: \sigma 1^2 = \sigma 2^2 \quad (4.37)$$

$$H_1: \sigma 1^2 \neq \sigma^2 \quad (4.38)$$

Si Ho es cierta:

$$\sigma 1^2 \text{ será parecida a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \approx 1 \quad (4.39)$$

Si Ho es falsa:

$$\sigma 1^2 \text{ difieren mucho a } \sigma 2^2 \rightarrow \frac{\sigma 1^2}{\sigma 2^2} \neq 1 \quad (4.40)$$

$$\frac{\sigma 1^2}{\sigma 2^2} = \frac{104.75}{5.75} = 18.22 \neq 1$$

Se rechaza Ho puesto que sus varianzas difieren mucho.

Ahora plantearemos las siguientes hipótesis con respecto a sus medias:

$$Ho: \mu 1 = \mu 2 \quad (4.41)$$

$$H_1: \mu 1 \neq \mu 2 \quad (4.42)$$

Si Ho es cierta:

$$\mu 1 \text{ será parecida a } \mu 2 \rightarrow \mu 1 - \mu 2 \approx 0 \quad (4.43)$$

Si Ho es falsa:

$$\mu 1 \text{ difieren mucho a } \mu 2 \rightarrow \mu 1 - \mu 2 \neq 0 \quad (4.44)$$

$$\mu 1 - \mu 2 = 11.93 - 4.43 = 7.5 \neq 0$$

Se Rechaza la Ho puesto que las medias difieren mucho.