UNIDAD 5:
Base de datos remota (Método 1)

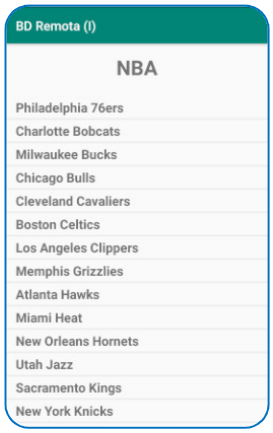
Por motivos de seguridad, se recomienda que todos los accesos a los datos de una BD sean gestionados por el propio servidor, y que las aplicaciones clientes se limiten a realizar peticiones al servidor y gestionar las respuestas obtenidas por este.

Hay muchas formas en las que el servidor puede ofrecer esos datos: desde un simple archivo de texto con los datos formateados para que puedan ser obtenidos, a sofisticadas API implementadas en forma de webservices.

La primera alternativa consistiría en crear scripts en el servidor para realizar cada una de las operaciones que se necesiten: insertar, obtener, etc. y realizar conexiones HTTP desde nuestra aplicación Android que realicen la petición y procesen los datos del archivo generado.

Como ejemplo vamos a crear una pequeña aplicación que solicite los datos de los equipos de la base de datos "nba" y muestre en una lista el nombre completo formado por ciudad y nombre del equipo, utilizando varios elementos que ya tenemos implementados introduciendo algunas modificaciones:

- Script PHP "obtenerEquipos": lo modificaremos para que devuelva toda la información de los equipos (nombre, ciudad, conferencia, división), pero que no permita obtener los datos si no se suministra un nombre de usuario y contraseña correcto en la petición.
- Clase "DescargaWeb": básicamente hará lo mismo pero estableciendo como parámetros de la petición un nombre de usuario y contraseña, y generando una lista de objetos "Equipo" cada uno de los cuales con los atributos: nombre, ciudad, conferencia y división.
- ListView: se adaptará para que muestre ciudad y nombre de cada uno de los equipos.



SCRIPT "obtenerEquipos.php"

Como se ha indicado vamos a modificar el script desarrollado en la práctica anterior para que devuelva toda la información de los equipos en lugar de sólo su nombre.

Para ello utilizaremos caracteres delimitadores, uno para cada campo del equipo (,) y otro para cada equipo (
), de manera que la información de cada equipo tendrá este formato:

```
Nombre,Ciudad,Conferencia,Division<br/>
```

Por otro lado, las peticiones de información deberán incluir cómo parámetros un nombre de usuario y contraseña válidos, de lo contrario no se realizará la conexión.

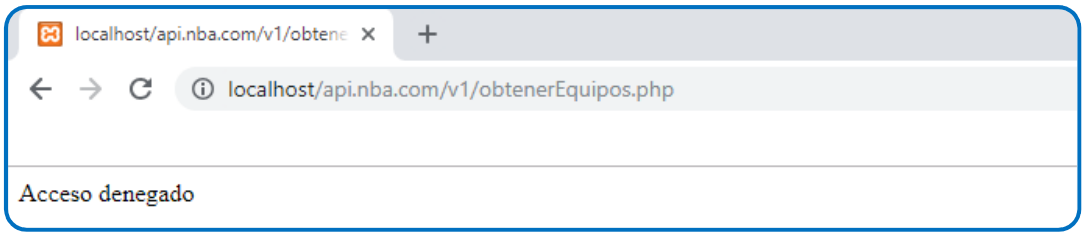
La petición deberá tener esta forma:

http://IP/api.nba.com/v2/obtenerEquipos.php?user=nbauser&pass=nbauser

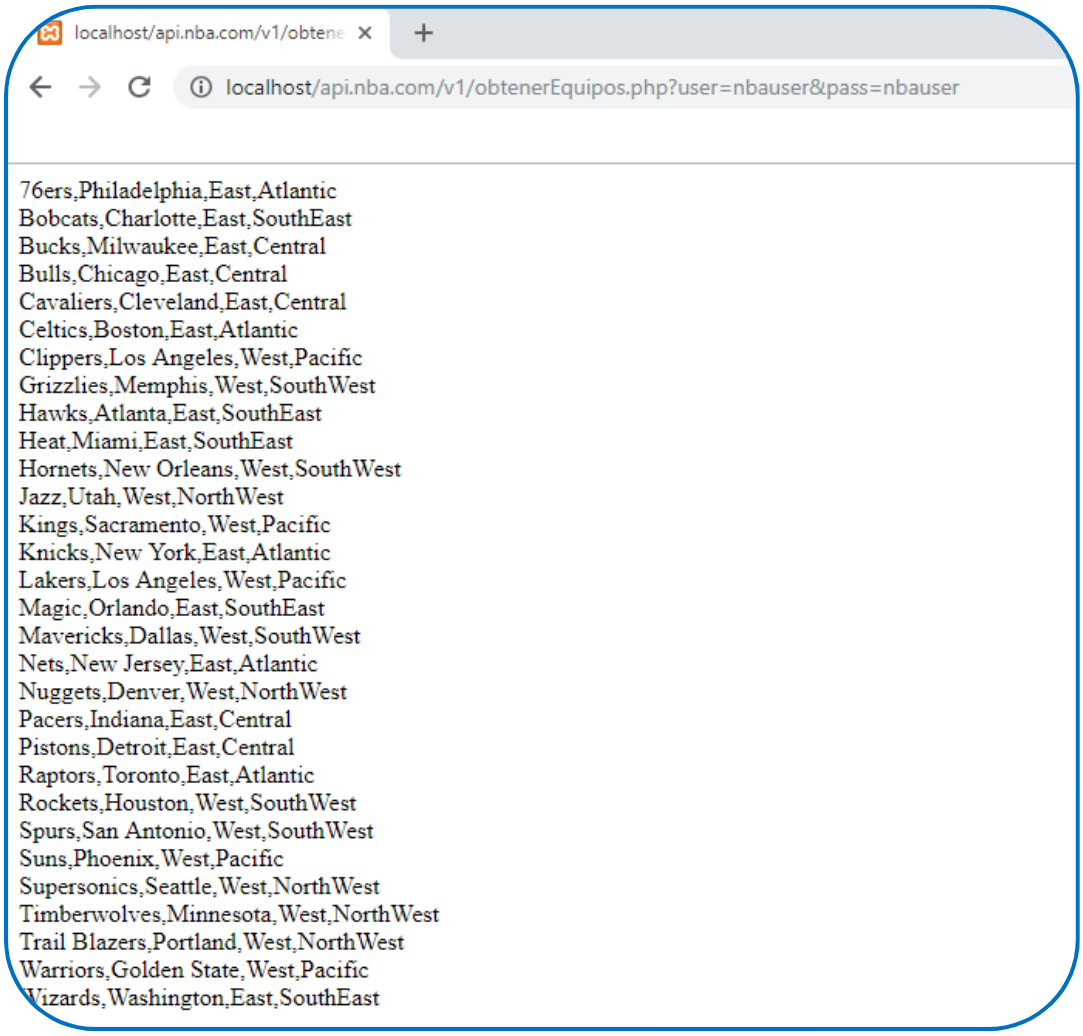
El script completo, que almacenaremos en la carpeta "htdocs\api.nba.com\v2" quedaría así:

```php
<?php
   $host = "localhost";
   $bd = "nba";
   $user = "";
   $pass = "";

   // Recogemos los parámetros suministrados
   if (isset($_REQUEST['user'])) {
      $user = $_REQUEST['user'];
   }
   if (isset($_REQUEST['pass'])) {
      $pass = $_REQUEST['pass'];
   }

   // Creamos la conexión, obtenemos los datos y les damos formato
   @$con = mysqli_connect($host, $user, $pass, $bd) or die("Imposible conectar");
   $query = "SELECT * FROM equipos";
   @$res = mysqli_query($con, $query) or die(mysqli_error($con));
   while ($equipo = mysqli_fetch_assoc($res)) {
   echo $equipo['Nombre'].",".$equipo['Ciudad'].",".$equipo['Conferencia']."," 
       .$equipo['Division'] ."<br/>";
   }
   mysqli_close($con);
?>
```

Si intentamos acceder sin facilitar los datos del usuario la respuesta será esta:



Una vez que agreguemos los parámetros a la petición, obtendremos los datos de los equipos en el formato establecido:



INTERFAZ

La interfaz de nuestra aplicación va a contar con una única actividad en la que mostraremos la lista de equipos, en la que para cada ítem (equipo) se mostrará un único *TextView* que contenga la ciudad y el nombre del equipo, por ejemplo: "Boston Celtics".

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txtTitulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:text="NBA"
        android:textAlignment="center"
        android:textSize="32sp"
        android:textStyle="bold"/>

    <ListView
        android:id="@+id/listEquipos"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"/>

</LinearLayout>
```

listitem_equipo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/itemNombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="20sp"
        android:layout_marginLeft="16dp"
        android:layout_marginVertical="4dp"/>

</LinearLayout>
```

LISTA DE EQUIPOS

Para gestionar la información de los equipos vamos a crear una clase "Equipo" con los atributos correspondientes a un equipo (nombre, ciudad, conferencia y división) y sus métodos *get* y *set* correspondientes.

Equipo.java

```
public class Equipo {

  private String nombre, ciudad, conferencia, division;

  public Equipo(String nom, String ciu, String conf, String div) {
    this.nombre = nom;
    this.ciudad = ciu;
    this.conferencia = conf;
    this.division = div;
  }

  public Equipo () {
    this.nombre = "";
    this.ciudad = "";
    this.conferencia = "";
    this.division = "";
  }

  public String getNombre() {
    return nombre;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }

  public String getCiudad() {
    return ciudad;
  }

  public void setCiudad(String ciudad) {
    this.ciudad = ciudad;
  }

  public String getConferencia() {
    return conferencia;
  }

  public void setConferencia(String conferencia) {
    this.conferencia = conferencia;
  }

  public String getDivision() {
    return division;
  }

  public void setDivision(String division) {
    this.division = division;
  }
}
```

Además, necesitaremos un adaptador para establecer la forma de visualización de los equipos, que en este caso va a ser un único *TextView* en el que se mostrarán la ciudad y el nombre del equipo concatenados.

AdaptadorEquipo.java

```
public class AdaptadorEquipo extends ArrayAdapter<Equipo> {

  private ArrayList<Equipo> datos;

  static class ViewHolder {
    TextView nombre;
  }

  public AdaptadorEquipo(Context c, ArrayList<Equipo> items) {
    super(c, R.layout.listitem_equipo, items);
    datos = items;
  }

  public View getView(int pos, View vista, ViewGroup parent){
    View item = vista;
    ViewHolder holder;

    if(item == null) {
      LayoutInflater inflater = LayoutInflater.from(getContext());
      item = inflater.inflate(R.layout.listitem_equipo, null);
      holder = new ViewHolder();
      holder.nombre = (TextView)item.findViewById(R.id.itemNombre);
      item.setTag(holder);

    } else {
      holder = (ViewHolder)item.getTag();
    }
      String txtNombre = datos.get(pos).getCiudad() + " " +
      datos.get(pos).getNombre();
      holder.nombre.setText(txtNombre);
      return(item);
    }
}
```

MAINACTIVITY

La actividad principal de nuestra aplicación va a mostrar una lista con los equipos de la nba, para ello obtendrá la información de la base de datos "nba" (obtenerEquipos) y a continuación suministrará el *ArrayList* obtenido al adaptador de la lista para mostrarlos en el *ListView*.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  ArrayList<Equipo> equipos = obtenerEquipos();
  AdaptadorEquipo adap = new AdaptadorEquipo(this, equipos);
  ListView listaEquipos = (ListView)findViewById(R.id.listEquipos);
  listaEquipos.setAdapter(adap);
}
```

El método "obtenerEquipos" comprobará que tengamos una conexión de red activa y a continuación intentará descargar la información generada por nuestro script PHP mediante una conexión HTTP (a la IP de nuestro servidor web), valiéndose de la clase "DescargaWeb" que utilizamos en la práctica de conexión a Internet con algunas modificaciones.

Fíjate que en la llamada al método "*execute*" se ha añadido una llamada a "*get*".

Dentro de "DescargaWeb" se asigna la descarga del archivo a una tarea asíncrona para evitar que la aplicación se quede "colgada" mientras obtiene los datos, pero como en este caso la aplicación no puede continuar hasta tener la lista, hemos de llamar a "*get*" para evitar que se cree la lista antes de haber obtenido los datos.

La llamada a "getEquipos" devolverá un *ArrayList* que si todo ha ido bien contendrá una lista con todos los equipos de la base de datos "nba" y en caso contrario estará vacía.

```
protected ArrayList<Equipo> obtenerEquipos() {
  ArrayList<Equipo> teams = new ArrayList<Equipo>();
  ConnectivityManager conMgr =
    (ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
  NetworkInfo netInfo = conMgr.getActiveNetworkInfo();
  if (netInfo != null && netInfo.isConnected()) {
    String myurl = "http://202.1.1.3/api.nba.com/v2/obtenerEquipos.php";
    DescargaWeb dw = new DescargaWeb();
    try { dw.execute(myurl).get();}
    catch (ExecutionException e) { e.printStackTrace(); }
    catch (InterruptedException e) { e.printStackTrace(); }
    teams = dw.getEquipos();
  } else {
    int duracion = Toast.LENGTH_SHORT;
    String mensa = "No se ha podido establecer la conexión";
    Toast alerta = Toast.makeText(MainActivity.this,mensa,duracion);
    alerta.show();
  }
  return teams;
  }
}
```

OBTENER LOS DATOS

Como ya se ha indicado, en este método los datos se obtienen creando una conexión HTTP al script PHP y recogiendo los datos que devuelve dicho script. Para ello vamos a partir de la clase "DescargaWeb" que utilizamos en la práctica de conexión a Internet para descargar un archivo vía HTTP y lo vamos a modificar para adaptarlo a las necesidades de esta aplicación:

1) **Añadir parámetros**: para evitar accesos indebidos a la información hemos incluido en el script PHP la necesidad de enviar un nombre de usuario y una contraseña válidos para acceder a la base de datos, de manera que nuestra solicitud deberá incluir ese nombre de usuario y contraseña.

   Para ello bastaría con concatenar los parámetros a la url en el formato adecuado:

```
String myurl = "http://202.1.1.3/api.nba.com/v2/obtenerEquipos.php";
myurl = myurl + "?user=nbauser&pass=nbauser";
```

   Pero vamos a hacerlo sobre la conexión (*HttpURLConnection*) para ver cómo se pueden agregar parámetros de forma dinámica (requiere establecer el método de envío *POST*):

```
URL url = new URL (myurl);
HttpURLConnection con;
con.setReadTimeout(10000);      // milisegundos
con.setConnectTimeout(15000);   // milisegundos
con.setRequestMethod("POST");
con.setDoInput(true);
String parametros = "user=nbauser&pass=nbauser";
con.setFixedLengthStreamingMode(parametros.getBytes().length);
PrintWriter out = new PrintWriter(con.getOutputStream());
out.print(parametros);
out.close();
```

   La primera parte es igual a lo que teníamos salvo el método de envío (*setRequestMethod*) que debe ser *POST*.

   Los datos de la conexión *HttpURLConnection* se guardan en un *buffer* (*getOutputStream*), de manera que para poder agregar los parámetros antes debemos aumentar el tamaño de dicho *buffer* el número de *bytes* necesario para albergar los parámetros mediante *setFixedLengthStreamingMode*.

   Tras aumentar el tamaño del *buffer* de salida, creamos un objeto *PrintWriter* para escribir sobre él, escribimos los parámetros (*print*) y cerramos el flujo (*close*).

2) **Generar salida**: en la clase DescargaWeb original generábamos un *String* con todos los caracteres del archivo, y en el método *onPostExecute* escribíamos el contenido de ese *String* directamente sobre un *TextView*. Ahora nuestro archivo de salida tiene un formato determinado, en el que cada línea es un equipo cuyos campos están separados con comas, y lo que vamos a hacer es almacenar dicha información en un *ArrayList* de objetos "Equipo" al que podremos acceder a través del método getEquipos.

```
protected ArrayList<Equipo> equipos;
...
protected void onPostExecute (String result) {
  if (result!=null) {
    String[] datos = result.split("<br/>");
    for (int i=0;i<datos.length;i++) {
      String [] datosEquipo = datos[i].split(",");
      if (datosEquipo.length == 4) {
        Equipo equipo = new Equipo();
        equipo.setNombre(datosEquipo[0]);
        equipo.setCiudad(datosEquipo[1]);
        equipo.setConferencia(datosEquipo[2]);
        equipo.setDivision(datosEquipo[3]);
        equipos.add(equipo);
      }
    }
  }
}
...
public ArrayList<Equipo> getEquipos() {
    return equipos;
}
```

La clase “DescargaWeb” completa quedaría así:

```
public class DescargaWeb extends AsyncTask<String, Void, String> {

  protected ArrayList<Equipo> equipos;

  public DescargaWeb () {
    equipos = new ArrayList<Equipo>();
  }


  // doInBackground: realizar la tarea asignada en segundo plano
  @Override
  protected String doInBackground(String... urls) {
    try {
      return descargaUrl (urls[0]);
    } catch (IOException e) {
      return null;
    }
  }


  // onPostExecute: genera ArrayList a partir del String obtenido
  @Override
  protected void onPostExecute (String result) {
    if (result!=null) {
      String[] datos = result.split("<br/>");
      for (int i=0;i<datos.length;i++) {
        if (datosEquipo.length == 4) {
          String [] datosEquipo = datos[i].split(",");
          Equipo equipo = new Equipo();
          equipo.setNombre(datosEquipo[0]);
          equipo.setCiudad(datosEquipo[1]);
          equipo.setConferencia(datosEquipo[2]);
          equipo.setDivision(datosEquipo[3]);
          equipos.add(equipo);
        }
      }
    }
  }


  // getEquipos: devuelve el ArrayList con los equipos de la nba
  public ArrayList<Equipo> getEquipos() {
    return equipos;
  }
```

```
  // descargaUrl: establece la conexión HTTP y devuelve el contenido del
  // InputStream convertido a String
  private String descargaUrl(String myurl) throws IOException {
    String result = null;
    InputStream is = null;
    try {
      // Preparamos la conexión
      URL url = new URL (myurl);
      HttpURLConnection con;
      if (myurl.startsWith("https")) {
        con = (HttpsURLConnection)url.openConnection();
      } else {
        con = (HttpURLConnection)url.openConnection();
      }
      con.setReadTimeout(10000);      // milisegundos
      con.setConnectTimeout(15000);  // milisegundos
      con.setRequestMethod("POST");
      con.setDoInput(true);

      // Añadimos parámetros (metodo POST)
      String parametros = "user=nbauser&pass=nbauser";
      con.setFixedLengthStreamingMode(parametros.getBytes().length);
      PrintWriter out = new PrintWriter(con.getOutputStream());
      out.print(parametros);
      out.close();

      // Descargamos los datos
      con.connect();
      is = con.getInputStream();

      // Convertimos los datos obtenidos (InputStream) a String
      ByteArrayOutputStream os = new ByteArrayOutputStream();
      int i;
      while ( (i=is.read()) != -1) {
        os.write(i);
      }

      // Convertimos al codigo de caracteres deseado antes de devolverlos
      // result = os.toString("UTF-8");
      result = os.toString("iso-8859-1");

    } catch (IOException e) {
      Log.d ("Error",e.getMessage());
    }

    finally {
      // Cerramos el InputStream
      if (is != null) {
        is.close();
      }
    }
    return result;
  }
}
```

MANIFIESTO

No debemos olvidar añadir los permisos necesarios en nuestro archivo de manifiesto:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

EJECUTANDO LA APP EN ANDROID 9

Desde la versión 9 de Android (Pie) se ha restringido el tráfico HTTP de texto sin encriptar, por lo que si deseamos poder recoger ese tipo de contenido tendremos que habilitarlo en el Manifiesto, mediante la propiedad "*usesCleartextTraffic*" dentro del elemento "*Application*".

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    android:usesCleartextTraffic="true">
```

Esta configuración permitirá el tráfico de texto plano para todas las comunicaciones.
Pero también podemos limitarlo sólo a los dominios en los que confiemos mediante un archivo de configuración de red que crearemos en la carpeta de recursos "res" y dentro de ella en "XML" al que podemos llamar "network_security_config.xml":

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">tuDominio/IP</domain>
    </domain-config>
</network-security-config>
```

Tras crear este archivo, indicando los dominios en los que se permite el tráfico de texto plano, deberemos añadir el archivo de configuración al Manifiesto, también dentro de "*application*".

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    android:networkSecurityConfig="@xml/network_security_config">
```