PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

TÉCNICO EN DESARROLLO DE APLICACIONESMULTIPLATAFORMA

Servicios en red
codificación JSON
XAMPP
PHP: lenguaje de programación backend

Veremos una manera de salvaguardar la base de datos de una app aunque ésta se desinstale del dispositivo móvil.

Introducción a JSON

Cuando queremos que una **aplicación**, independientemente de que sea de escritorio o de móvil, pueda **comunicarse con otra** que está en otro sistema, necesitaremos un **estándar** para que todas las aplicaciones puedan entenderlo.

En nuestro caso, vamos a comunicar una **aplicación móvil con un servidor** remoto y, para ello, podremos utilizar **dos tipos de estándares**, **XML y JSON**.

JSON es una **codificación de texto** que sirve para **intercambiar datos** de una forma **sencilla y estructurada**. Este tipo de formato presenta una serie de **ventajas sobre el XML**, que ya estudiamos en otras asignaturas, y es que **JSON es muy fácil de evaluar**, aunque esto no significa que XML no se utilice, todo lo contrario: en una **misma aplicación**, **se pueden utilizar tanto JSON como XML** para realizar las comunicaciones.

JSON soporta los siguientes datos:

- **Números**: Se pueden representar números positivos, negativos, enteros y reales, **separados por puntos**. Por ejemplo: 123.456
- **Cadenas**: Representan secuencias de **cero o más caracteres**. Se ponen entre **comillas dobles**. Por ejemplo: “Hola”.
- **Booleanos**: Representan valores booleanos y pueden tener dos valores: true y false.
- **Null**: Representan el valor nulo.
- **Array**: Representa una **lista ordenada** de **cero o más valores**, los cuales pueden ser de **cualquier tipo**. Los valores se separan por comas y el **vector** se introduce entre **corchetes**. Por ejemplo [“valor 1”,”valor 2”,”valor 3”].
- **Objetos**: Este tipo de dato viene representado por una **colección que no está ordenada**, compuesta por elementos del tipo <nombre>:<valor>, que podremos **separar por comas** y **delimitar entre llaves**. El **nombre**, obligatoriamente, debe ser una **cadena**, mientras que el **valor** puede ser **cualquier tipo** de dato conocido.

Ejemplo de JSON simple:

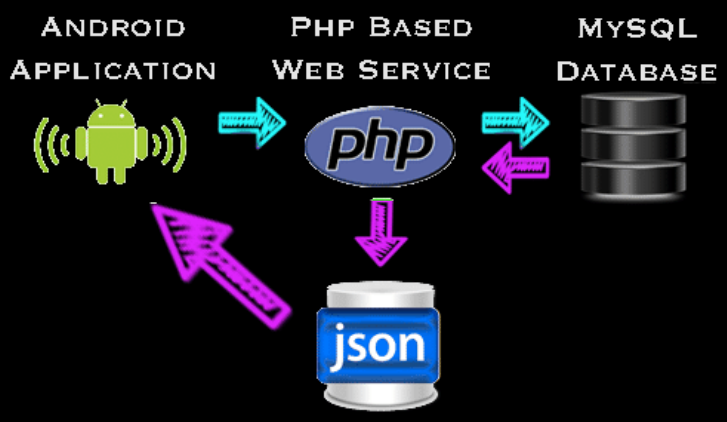
```
{
        "departamento": 8,
        "nombredepto": "Ventas",
        "director": "Juan Rodríguez",
        "empleados": [
                {
                        "nombre": "Pedro",
                        "apellido": "Fernández"
                },
                {
                        "nombre": "Jacinto",
                        "apellido": "Benavente"
                }
        ]
}
```

Usando JSON en Android

Vamos a conectar las aplicaciones Android con un servidor remoto mediante este estándar. Es decir, nuestra aplicación se conectará a un servidor externo que use **MySQL** para almacenar los datos, el **servidor obtendrá los datos** que se pidan y los **moldeará en formato JSON**, **devolviéndolos** a nuestra aplicación. Esta los **leerá**, pero necesitaremos un «**intermediario**» para realizar este proceso.

Concretamente, este intermediario será **PHP**.

Esquema Android – PHP – JSON – MySQL:



Para poder tratar con los datos en JSON, deberemos usar las clases:

JSONObject, que almacenará un objeto con formato JSON;

JSONArray, que guardará un array de objetos JSONObject. Estas clases vienen directamente integradas en la **JDK de Android**, así que no es necesario agregar ninguna biblioteca externa.

Un **JSONObject o un JSONArray** pueden ser creados directamente desde un **String que cumpla el formato JSON**, utilizando sus respectivos **constructores**.

Cada vez que realicemos operaciones con JSON deberemos utilizar la **excepción JSONException**.

Los JSONArray son un array **equivalente a un ArrayList**, salvo que ya está optimizado para datos del tipo JSON, por lo que, en los JSONArray, vamos a tener todos los **métodos** de los **ArrayList**.

En los objetos de tipo JSON, tendremos los **métodos get** para cada tipo de dato, getString, getInt, etc., que nos devolverán su valor, **pasándole el nombre del dato**.

Ejemplo de JSON paso a paso:

En el fichero MainActivity

```
public class MainActivity extends AppCompatActivity {

   private RecyclerView listaPescados;
   private AdaptadorPescado adaptador;

   private final String CONTENIDO = "[{\"fish_name\":\"Indian
Mackerel\",\"cat_name\":\"Marine  Fish\",\"size_name\":\"Medium\",\"price\":\"100\"},\n" +
         "{\"fish_name\":\"Manthal Repti\",\"cat_name\":\"Marine
Fish\",\"size_name\":\"Small\",\"price\":\"200\"},\n" +
         "{\"fish_name\":\"Baby Sole Fish\",\"cat_name\":\"Marine
Fish\",\"size_name\":\"Small\",\"price\":\"600\"},\n" +
         "{\"fish_name\":\"Silver Pomfret\",\"cat_name\":\"Marine
Fish\",\"size_name\":\"Large\",\"price\":\"300\"},\n" +
         "{\"fish_name\":\"Squid\",\"cat_name\":\"Shell
Fish\",\"size_name\":\"Small\",\"price\":\"800\"},\n" +
         "{\"fish_name\":\"Clam Meat\",\"cat_name\":\"Shell
Fish\",\"size_name\":\"Small\",\"price\":\"350\"},\n" +
         "{\"fish_name\":\"Indian Prawns\",\"cat_name\":\"Shell
Fish\",\"size_name\":\"Medium\",\"price\":\"270\"},\n" +
         "{\"fish_name\":\"Mud Crab\",\"cat_name\":\"Shell
Fish\",\"size_name\":\"Medium\",\"price\":\"490\"},\n" +
         "{\"fish_name\":\"Grey Mullet\",\"cat_name\":\"Backwater
Fish\",\"size_name\":\"Small\",\"price\":\"670\"},\n" +
```

```
        "{\"fish_name\":\"Baasa\",\"cat_name\":\"Backwater
Fish\",\"size_name\":\"Large\",\"price\":\"230\"},\n" +
        "{\"fish_name\":\"Pearl Spot\",\"cat_name\":\"Backwater
Fish\",\"size_name\":\"Small\",\"price\":\"340\"},\n" +
        "{\"fish_name\":\"Anchovy\",\"cat_name\":\"Marine
Fish\",\"size_name\":\"Small\",\"price\":\"130\"},\n" +
        "{\"fish_name\":\"Sole Fish\",\"cat_name\":\"Marine
Fish\",\"size_name\":\"Medium\",\"price\":\"250\"},\n" +
        "{\"fish_name\":\"Silver Croaker\",\"cat_name\":\"Marine
Fish\",\"size_name\":\"Small\",\"price\":\"220\"}]";

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // ***** Ligamos los elementos de la actividad *****
        listaPescados = findViewById(R.id.listaPescados);
        // ***********************************************
        try
        {
            ArrayList<Pescado> datospescados = new ArrayList<>();
            // Creo el JSONArray con los datos en formato JSON
            JSONArray jArray = new JSONArray(CONTENIDO);
            // Extraigo los datos que están en JSON
            for (int i = 0; i < jArray.length(); i++)
            {
                JSONObject json_data = jArray.getJSONObject(i);
                String nombre = json_data.getString("fish_name");
                String categoria = json_data.getString("cat_name");
                String tamano = json_data.getString("size_name");;
                int precio = json_data.getInt("price");
                Pescado pescado = new Pescado(nombre, categoria, tamano, precio);
                datospescados.add(pescado);
            }
            // Mostramos la lista
            // Con esto el tamaño del recyclerwiew no cambiará
            listaPescados.setHasFixedSize(true);
            // Creo un layoutmanager para el recyclerview
            LinearLayoutManager llm = new LinearLayoutManager(this);
            listaPescados.setLayoutManager(llm);
            adaptador = new AdaptadorPescado(this, datospescados);
            listaPescados.setAdapter(adaptador);
            adaptador.refrescar();
        }
```

```
        catch (JSONException e)
        {
            System.out.println("Error: " + e.toString());
        }
    }
}
```

En modelo (clase **Pescado**)

```
public class Pescado {

    private String nombre;
    private String categoria;
    private String tamano;
    private int precio;

    public Pescado(String nombre, String categoria, String tamano, int precio)
    {
        this.nombre = nombre;
        this.categoria = categoria;
        this.tamano = tamano;
        this.precio = precio;
    }

    public String getNombre()
    {
        return nombre;
    }

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

    public String getCategoria()
    {
        return categoria;
    }

    public void setCategoria(String categoria)
    {
        this.categoria = categoria;
    }
```

```
    public String getTamano()
    {
        return tamano;
    }

    public void setTamano(String tamano)
    {
        this.tamano = tamano;
    }

    public int getPrecio()
    {
        return precio;
    }

    public void setPrecio(int precio)
    {
        this.precio = precio;
    }
}
```

Y el AdaptadorPescado, para crear el ReciclerView

```
public class AdaptadorPescado extends
RecyclerView.Adapter<AdaptadorPescado.HolderPescado>{

    public static class HolderPescado extends RecyclerView.ViewHolder
    {
        ImageView ivFish;
        TextView textFishName, textPrice, textSize, textType;

        HolderPescado(View itemView)
        {
            super(itemView);
            ivFish = itemView.findViewById(R.id.ivFish);
            textFishName = itemView.findViewById(R.id.textFishName);
            textPrice = itemView.findViewById(R.id.textPrice);
            textSize = itemView.findViewById(R.id.textSize);
            textType = itemView.findViewById(R.id.textType);
        }
    };
```

```
    private ArrayList<Pescado> datos;
    private Context contexto;

    public AdaptadorPescado(Context contexto, ArrayList<Pescado> datos)
    {
        this.contexto = contexto;
        this.datos = datos;
    }

    /**
     * Agrega los datos que queramos mostrar
     * @param datos Datos a mostrar
     */
    public void add(ArrayList<HolderPescado> datos)
    {
        datos.clear();
        datos.addAll(datos);
    }

    /**
     * Actualiza los datos del ReciclerView
     */
    public void refrescar()
    {
        notifyDataSetChanged();
    }

    @Override
    public HolderPescado onCreateViewHolder(ViewGroup parent, int viewType)
    {
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.elemento_pescado,
parent, false);
        HolderPescado pvh = new HolderPescado(v);
        return pvh;
    }

    @Override
    public void onBindViewHolder(HolderPescado pescadoactual, int position)
    {
        pescadoactual.ivFish.setImageResource(R.drawable.ic_img_fish);
        pescadoactual.textFishName.setText(datos.get(position).getNombre());
        pescadoactual.textPrice.setText(String.valueOf(datos.get(position).getPrecio()));
        pescadoactual.textSize.setText(datos.get(position).getTamano());
        pescadoactual.textType.setText(datos.get(position).getCategoria());
        pescadoactual.textPrice.setTextColor(ContextCompat.getColor(contexto,
```

```
R.color.colorAccent));
    }

    @Override
    public int getItemCount()
    {
        return datos.size();
    }

    @Override
    public void onAttachedToRecyclerView(RecyclerView recyclerView)
    {
        super.onAttachedToRecyclerView(recyclerView);
    }
}
```

Los costes en los servidores remotos

Existen ciertos proveedores servidores remotos que ofrecen packs muy básicos de forma gratuita, pero, como es lógico, estos packs van a estar **muy limitados**.

Estas limitaciones se basan en el **uso de espacio para nuestro hosting**, es decir, el espacio en disco que podremos usar en el servidor remoto; también estarán limitadas el **número de bases de datos** que podremos gestionar, y, por último, habrá momentos en los que los **servidores** con servicio gratuito **no estarán disponibles**.

La suma de todo esto hará que nuestras aplicaciones no funcionen de un modo normal ni fiable.

Para el **uso didáctico**, este tipo de servidores **nos van a servir** a la perfección, ya que, normalmente, se usan para pruebas en las que estaremos continuamente montando y desmontando servicios para nuestro aprendizaje, sin ningún otro objetivo, es decir, sin dar un servicio a terceros (clientes por ejemplo). Por tanto, para esto sí son una opción que podemos tener muy en cuenta, y no tendremos que hacer un desembolso económico, por pequeño que sea.

Un ejemplo de este tipo de servicios gratuitos lo podemos encontrar en esta compañía: **https://es.000webhost.com/**

Introducción a PHP7

Para poder acceder y gestionar la base de datos del servidor externo que utilizaremos, necesitaremos un lenguaje de programación que se pueda ejecutar en la **parte de servidor**, conocida como **backend**.

Existen varios **lenguajes adecuados** para esto como, por ejemplo, **Python**, pero, en nuestro caso, vamos a utilizar **PHP** en su **versión 7**, ya que **soporta** la **programación dirigida a objetos**.

PHP es un **lenguaje de programación web** que se puede integrar fácilmente en ficheros HTML.

El intérprete de PHP solo **ejecutará el código** que se encuentra **entre sus delimitadores**, los cuales son:

<?php para abrir una sección PHP

?> para cerrarla. *

* Aunque es una práctica común en archivos que contienen solo código PHP omitir la etiqueta de cierre **?>** para evitar problemas de espacios en blanco.

El propósito de estos delimitadores es **separar el código PHP** del resto de código.

Las **variables** en este lenguaje de programación se van a **prefijar** con el **símbolo del dólar ($)**, y **no es necesario indicarle el tipo**, ya que lo adoptarán **automáticamente** cuando se igualen a algo.

Este lenguaje de programación soporta instrucciones **condicionales, bucles, funciones y clases**, como otros lenguajes de programación. Todo esto lo veremos poco a poco de una forma muy superficial en esta asignatura, únicamente con el objetivo de gestionar una base de datos.

Es importante no olvidar que, al igual que en Java, todas las instrucciones deberán **acabar en punto y coma (;)**.

En el bloque que estudiaremos sobre PHP, nos vamos a centrar en su funcionamiento como **mediador con la base de datos**. Así, no necesitaremos introducir datos por el usuario, pero sí **mostrar datos por pantalla** y, al ser un lenguaje de programación web, se **mostrarán en un navegador** (Firefox, Chrome, etc.), lo cual podremos hacer mediante la **instrucción echo**. Esta instrucción mostrará, en el navegador que utilicemos, el mensaje que queramos, pudiendo mostrar el valor de variables, como ya sabemos hacer.

Hola mundo en PHP. Guardado con extensión .php:

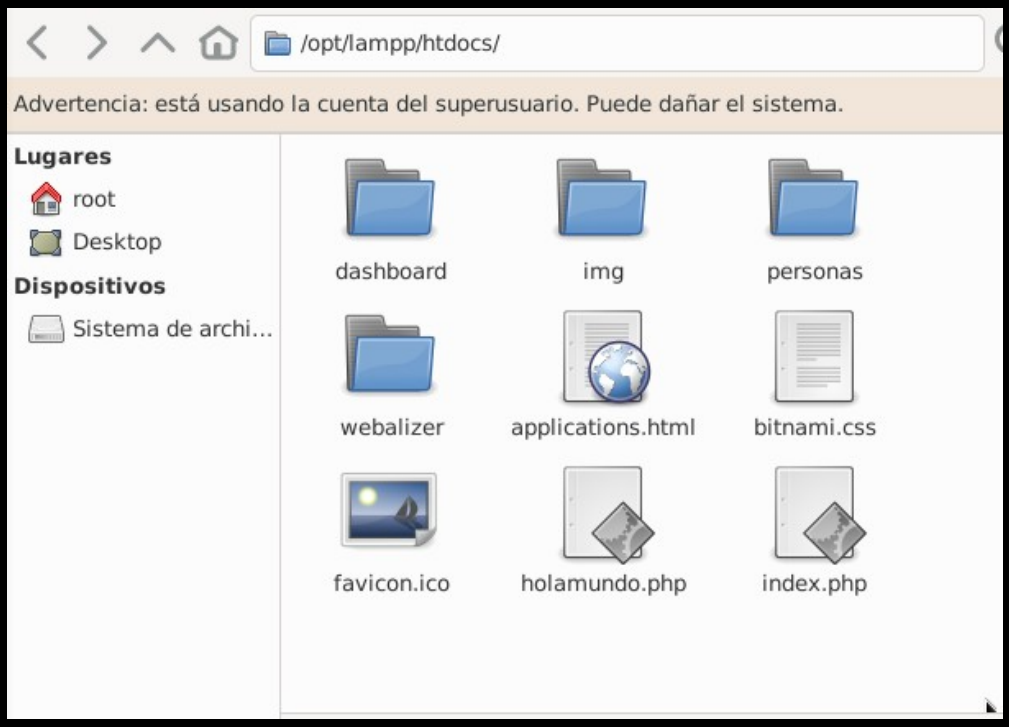
```
<!DOCTYPE HTML>
<html lang="es">
        <head>
                <meta charset="UTF-8" />
                <title>Ejemplo básico PHP</title>
        </head>
```

```
        <body>
            <p>
            <?php
                echo "Hola Mundo";
            ?>
            </p>
        </body>
    </html>
```

Si queremos ejecutar este archivo en nuestro navegador, se ejecutaría sin necesidad de incluirlo en un elemento html, de hecho esto se ejecutaría:

```
<?php
    echo "Hola Mundo";
?>
```

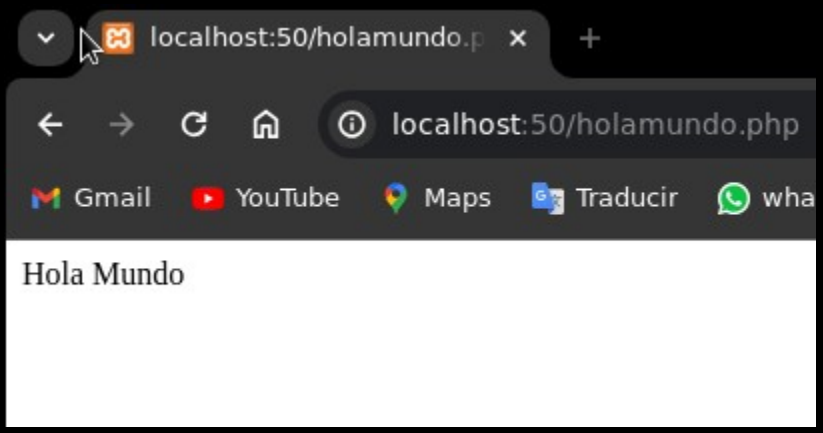
Pues bien, vamos al directorio donde instalamos nuestro XAMPP (LAMPP en Linux. La instalación de XAMPP la veremos más adelante en este pdf). Y buscamos el directorio htdocs, como se muestra en la siguiente captura, y ahí, con permisos de administrador (o de superusuario en Linux), incluímos nuestro archivo holamundo.php con el código de arriba, cualquiera de los dos.



Despues de lo cual, con nuestro servidor Apache “running” con XAMPP, vamos a nuestro navegador y escribimos como url a buscar:

http://localhost:50/holamundo.php

Si es el caso de que el puerto de nuestro servidor es el 50, si no lo sustituimos por el nuestro. Y veremos nuestro hola mundo en nuestro navegador:



Tipos de servidores

Como hemos comentado existen dos tipos de servidores remotos, los gratuitos, como un peor servicio, y los servidores de pago, que, aunque tengan un mejor servicio implican un desembolso económico.

Si no queremos dejarnos un dinero y tampoco queremos que nuestro servidor de problemas, una opción a tener en cuenta para tener un servidor que nos permita practicar con lo que estamos aprendiendo, es ***utilizar un servidor local****.*

Esto lo podemos conseguir con programas como ***XAMPP*** *por ejemplo, sobre el que profundizaremos a continuación.*

En la web ***apachefriends.org*** *puedes descargarlo y obtener más información.*

Instalación y configuración de PHP7 y MYSQL

Si no queremos utilizar un servidor web para utilizar PHP junto a MySQL, podremos instalarlos en nuestro propio equipo, convirtiéndolo en un servidor local, que funcionará a la perfección. De esta forma, cualquier contratiempo podrá resolverse sin depender de terceros.

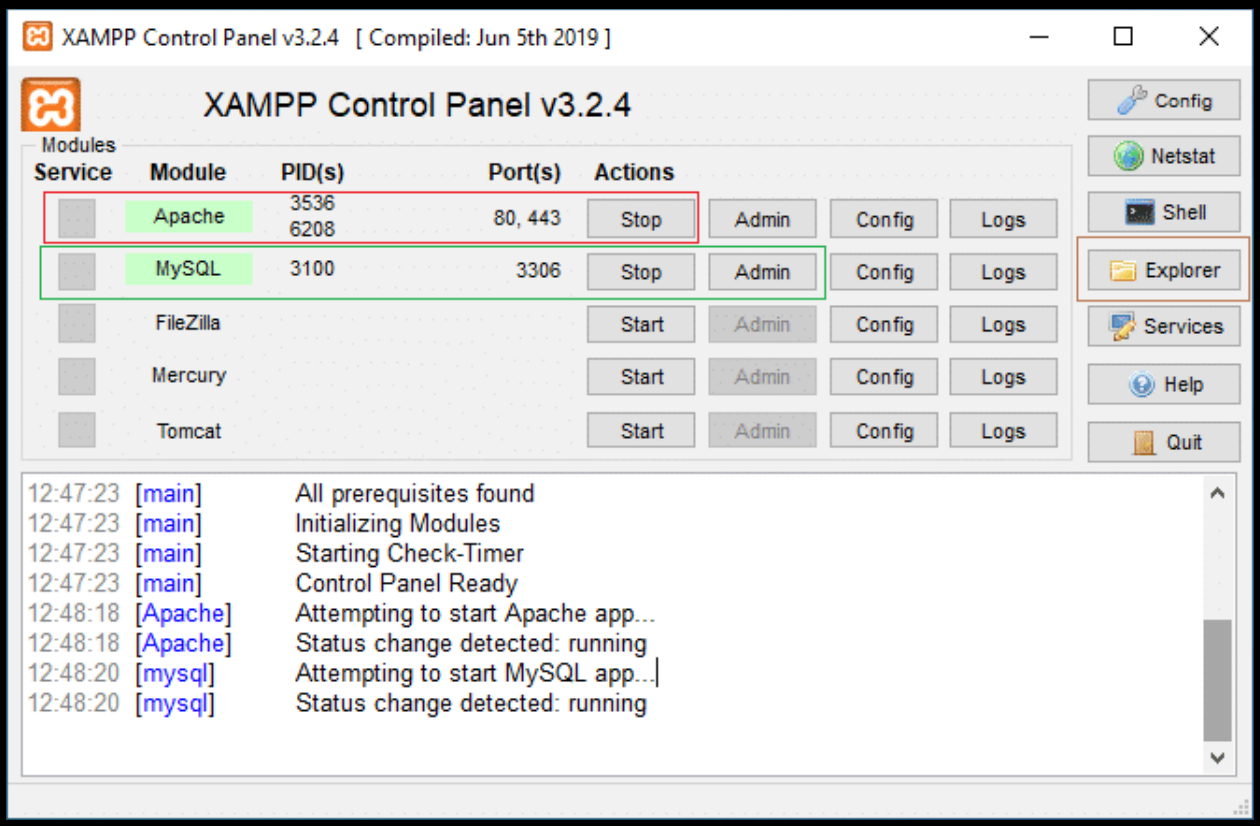
Para ello, necesitaremos instalar XAMPP, que podremos descargar de su página web: https://www.apachefriends. org/es/download.html

XAMPP nos ofrece, además de un **gestor de bases de datos MySQL**, la posibilidad de utilizar un **intérprete de PHP**.

Es decir, se **instalará un pequeño servidor** en nuestro ordenador, que podremos arrancar siempre que lo necesitemos, pudiendo utilizar, por tanto, PHP y MySQL para probar nuestras aplicaciones Android.

Una vez tengamos instalado XAMPP, tendremos las siguientes opciones disponibles.

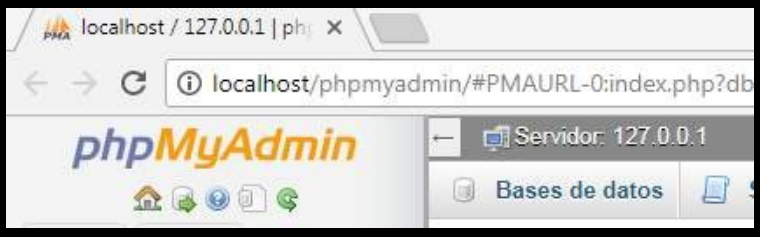
> Marcado en el **cuadrado rojo** tenemos la **opción de Apache**, que necesitamos para que nuestro **servidor** esté operativo. Pulsaremos el botón **Iniciar**.
>
> En el **recuadro verde** tenemos la **opción de MySQL**, que es necesario que activemos para que nuestra **base de datos** esté operativa. Pulsaremos **Iniciar**.



Si queremos iniciar el **gestor de bases de datos**, en este caso **phpMyAdmin**, deberemos pulsar en el **botón Admin** de la parte de **MySQL**, lo cual hará que se **abra** desde nuestro **navegador** predeterminado. El **usuario**, por defecto, es **root** y su **contraseña** está **vacía**.

En Linux (XAMPP es llamado LAMPP en la instalación), para acceder a **phpMyAdmin** no tendremos el botón Admin, de modo que una vez estén levantados el FTP, el servidor Apache y la BBDD MySQL, iremos a nuestro navegador y escribiremos como URL: http://localhost:**50**/phpmyadmin/, donde "50" es el puerto que le hemos asignado, que normalmente será el 80 si no lo tenéis ocupado con otro servicio en vuestro sistema operativo. Si tenéis dudas, podéis comprobar qué puerto tiene asignado XAMPP a Apache (servidor) si pulsamos en el botón "Configure" teniendo seleccionado Apache Web Server.

phpMyAdmin en localhost:



Para utilizar nuestro servidor local, deberemos hacer referencia a **localhost** en nuestro navegador o, en su defecto, a la dirección IP **127.0.0.1**.

Si no se conecta el servidor locar de Apache, lo más probable será que sea porque le puerto usado esté ocupado por otro servicio. De modo que lo solucionaremos cambiando el puerto, pulsando en el botón “config” podremos cambiar el puerto. Si el el 80 el puerto que da problemas, podríamos poner el 8080, o el 50, si no están ocupados por otro servicio. Este último me dió buen resultado en Debian (Linux).

Creación de base de datos Personas y de la tabla Persona:

Abrimos nuestro phpmyadmin, y selecionando la pestaña SQL vamos ejecutando cada sentencia SQL para la creacion de la tabla:

```
CREATE DATABASE personas;
USE personas;
```

para la creación de la tabla persona:

```
CREATE TABLE IF NOT EXISTS persona (
DNI VARCHAR(10) PRIMARY KEY,
nombre VARCHAR(100) NOT NULL,
apellidos VARCHAR(100) NOT NULL,
telefono int NOT NULL,
email VARCHAR(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
```

y para la inserción de registrops nuevos en la tabla personas:

```
INSERT INTO persona (DNI, nombre, apellidos, telefono, email) VALUES ('14786325P', 'Juan', 'López López', 666666666, 'juanlopez@gmail.com');
INSERT INTO persona (DNI, nombre, apellidos, telefono, email) VALUES ('24158749L', 'Pepe', 'Adamuz Núñez', 666956666, 'pepeadamuz@gmail.com');
INSERT INTO persona (DNI, nombre, apellidos, telefono, email) VALUES ('36214758I', 'Dolores', 'Pérez Aguilera', 684766666, 'doloresperez@gmail.com');
```

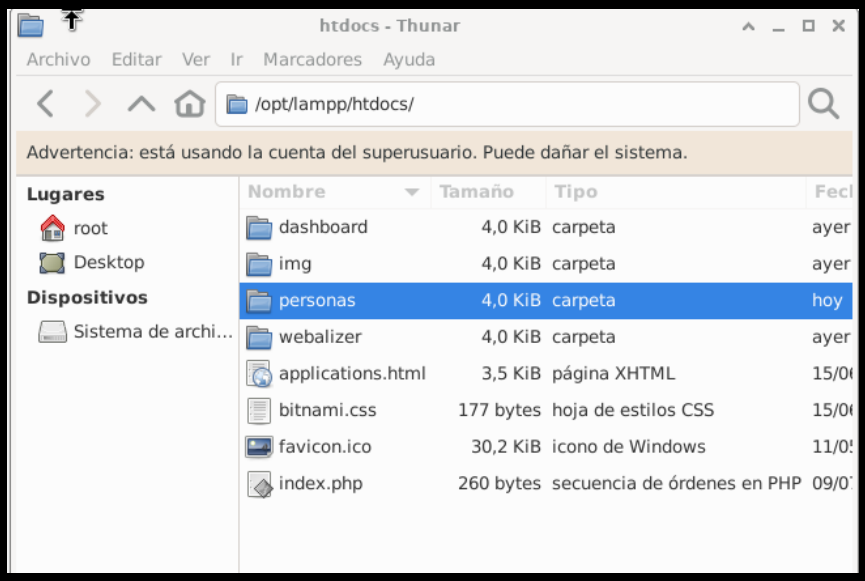
Pulsando en el botón “continuar” se irá ejecutando cada petición al gestor de BBDD MySQL...

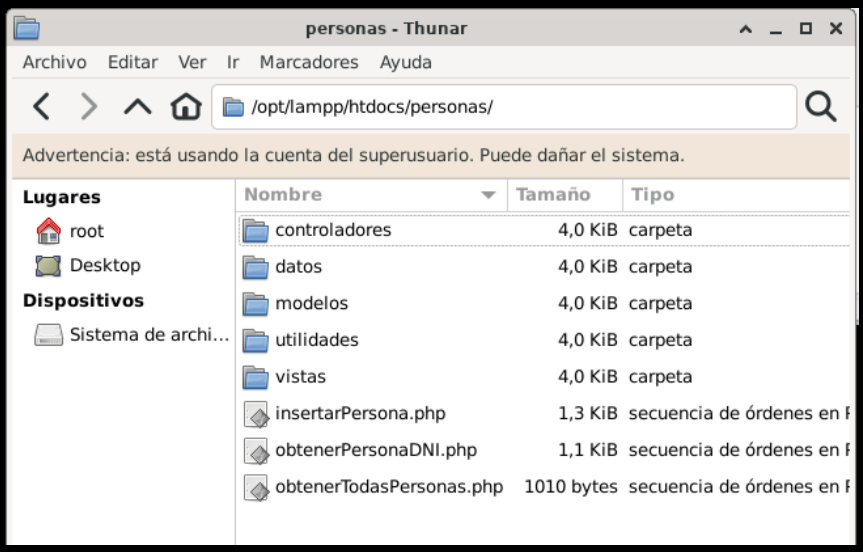
Los ficheros php deberemos de colocarlos dentro de nuestro servidor Apache. Para ello iremos al botón “exporer” si estamos en XAMPP de windows, o a “open application folder” si estamos enm XAMPP de Linux (LAMPP) y se nos abrirá el explorador en el directorio de XAMPP:

Vista de XAMPP en Linux (“Open Application Folder” para poder agregar nuestros ficheros .php)

Vista de XAMPP en Windows (“Explorer” para poder agregar nuestros ficheros .php)

Y nos situaremos en la carpeta /htdocs, donde agregaremos un directorio con el modelo vista controlador visto anteriormente:





Este directorio personas estará subido como directorio en este mismo repositorio de github.

Dentro del directorio modelos, tenemos a la clase Persona.php:

```
<?php

// Esta clase representa una persona

class Persona
{
        // Variables de clase
```

```
        private $DNI, $nombre, $apellidos, $telefono, $email;

        // Constructor
        public function __construct($nDNI, $nnombre, $napellidos, $ntelefono, $nemail)
        {
                $this->DNI = $nDNI;
                $this->nombre = $nnombre;
                $this->apellidos = $napellidos;
                $this->telefono = $ntelefono;
                $this->email = $nemail;
        }

        // Muestra los datos de la persona
        public function toString()
        {
                return
                        [
                                "DNI" => utf8_encode($this->DNI),
                                "nombre" => utf8_encode($this->nombre),
                                "apellidos" => utf8_encode($this->apellidos),
                                "telefono" => utf8_encode($this->telefono),
                                "email" => utf8_encode($this->email)
                        ];
        }
}
```

Dentro del directorio controladores tenemos al controladorPersona.php:

```
<?php
require_once('datos/ConexionBD.php');
require_once('utilidades/ExcepcionApi.php');
require_once('datos/mensajes.php');
// Esta clase representa un controlador para las personas
class ControladorPersonas
{
        // Nombres de la tabla y de los atributos, aunque no es obligatorio nos facilitará mucho todo
        const NOMBRE_TABLA = "persona";
        const DNI = "DNI";
        const NOMBRE = "nombre";
        const APELLIDOS = "apellidos";
        const TELEFONO = "telefono";
        const EMAIL = "email";
        /**
```

```php
* Descripción: Obtiene los datos de todas las personas que hay en el sistema
* mediante un SELECT *
* Este método se corresponederá con el fichero que está en la raiz del proyecto llamado
* obtenerTodasPersonas.
* @return Datos de todas las personas que hay en el sistema
*/
public function obtenerTodasPersonas()
{
	try
	{
		// Conectará a la base de datos con esta instrucción:
		$pdo = ConexionBD::obtenerInstancia()->obtenerBD();

		// Sentencia SELECT
		$comando = "SELECT * FROM " . self::NOMBRE_TABLA;

		$sentencia = $pdo->prepare($comando);

		// Dentro de $sentencia tendremos todos los resultados de la consulta
		$sentencia->execute();

		$array = array();

		// El método fetch nos irá devolviendo filas completas:
		while ($row = $sentencia->fetch(PDO::FETCH_ASSOC))
		{
		// Iremos agregando las filas a $array con el método array_push:
			array_push($array, $row);
		}

		return [
			[
				"estado" => ESTADO_CREACION_EXITOSA,
				"mensaje" => $array
			]
		];
	}
	catch (PDOException $e)
	{
		throw new ExcepcionApi(ESTADO_ERROR_BD, $e->getMessage());
	}
}
/**
* Descripción: Obtiene y devuelve una persona según su DNI
* @param DNI DNI de la persona
```

```php
* @return Datos de la persona indicada con su DNI
*/
public function obtenerPersonaDNI($DNI)
{
	try
	{
		$pdo = ConexionBD::obtenerInstancia()->obtenerBD();

		// Sentencia SELECT
		$comando = "SELECT * FROM " . self::NOMBRE_TABLA . " " .
		"WHERE " . self::DNI . " = ?";

		$sentencia = $pdo->prepare($comando);

		// Pongo los datos en la consulta INSERT
		$sentencia->bindParam(1, $DNI);

		$sentencia->execute();

		$array = array();

		while ($row = $sentencia->fetch(PDO::FETCH_ASSOC))
		{
			array_push($array, $row);
		}

		return [
			[
				"estado" => ESTADO_CREACION_EXITOSA,
				"mensaje" => $array
			]
		];
	}
	catch (PDOException $e)
	{
		throw new ExcepcionApi(ESTADO_ERROR_BD, $e->getMessage());
	}
}
/**
* Descripción: Inserta una persona en la base de datos
* @param persona Persona para insertar en la base de datos
* @return Indica si se ha insertado la persona correctamente (Código 1)
*/
public function insertarPersona($persona)
{
```

```
                try
                {
                        // Obtengo una instancia de la base de datos ya conectada
                        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
                        // Creo la sentencia INSERT
                        $comando = "INSERT INTO " . self::NOMBRE_TABLA . " ( " .
                                self::DNI . "," .
                                self::NOMBRE . "," .
                                self::APELLIDOS . "," .
                                self::TELEFONO . "," .
                                self::EMAIL . ")" .
                                " VALUES(?,?,?,?,?)";
                        $sentencia = $pdo->prepare($comando);
                        // Pongo los datos en la consulta INSERT
                        $sentencia->bindParam(1, $persona->DNI);
                        $sentencia->bindParam(2, $persona->nombre);
                        $sentencia->bindParam(3, $persona->apellidos);
                        $sentencia->bindParam(4, $persona->telefono);
                        $sentencia->bindParam(5, $persona->email);
                        // Ejecuto la consulta
                        $resultado = $sentencia->execute();
                }
                catch (PDOException $e)
                {
                        throw new ExcepcionApi(self::ESTADO_ERROR_BD, $e->getMessage());
                }

                switch ($resultado)
                {
                case self::ESTADO_CREACION_EXITOSA:
                        http_response_code(200);
                        return correcto;
                        break;
                case self::ESTADO_CREACION_FALLIDA:
                        throw new ExcepcionApi(self::ESTADO_CREACION_FALLIDA, "Ha ocurrido un
                        error.");
                        break;
                default:
                        throw new ExcepcionApi(self::ESTADO_FALLA_DESCONOCIDA, "Fallo
                        desconocido.", 400);
                }
        }
}
```

A continuación vemos el fichero **obtenerTodasPersonas**, del directorio raiz del proyecto, el cual será usado al ejecutar la función **obtenerTodasPersonas()** que acabamos de ver en la clase persona.php (justo arriba).

```php
<?php
// Hago que se muestren los errores si los hay
ini_set('display_errors', 1);
//  Importamos la vista y el controlador para poder utilizarlo:
require_once('vistas/VistaJson.php');
require_once('controladores/ControladorPersonas.php');
// Tipo de vista de la salida de datos.
$vista = new VistaJson();
// Con esta función nos aseguramos que cualquier excepción que ocurra se muestre adecuadamente
// en el mismo formato para evitar problemas.
set_exception_handler(function ($exception) use ($vista)
	{
		$cuerpo = array(
			array(
				"estado" => $exception->estado,
				"mensaje" => $exception->getMessage()
			)
		);
		if ($exception->getCode())
		{
			$vista->estado = $exception->getCode();
		}
		else
		{
			$vista->estado = 500;
		}

		$vista->imprimir($cuerpo);
	}
);
// Me creo un controlador de personas
$controladorp = new ControladorPersonas();
// Saco por pantalla en formato JSON el resultado, llamando al método obtenerTodasPersonas:
$vista->imprimir($controladorp->obtenerTodasPersonas());
```

Para llamar al fichero ObtenerTodasPersonas y que ejecute su código, tendremos que ir a nuestro navegador y escribir la ruta a ese fichero:

localhost:50/personas/obtenerTodasPersonas.php

localhost será la raiz de nuestro proyecto, dende se encuantran todos estos ficheros que hemos agregado, de modo que la ruta será:

Donde “50” recordemos que es el puerto que hemos asignado al servidor. Nos mostrará lo siguiente:


```
[
   {
      "estado": 1,
      "mensaje": [
         {
            "DNI": "14786325P",
            "nombre": "Juan",
            "apellidos": "López López",
            "telefono": 666666666,
            "email": "juanlopez@gmail.com"
         },
         {
            "DNI": "24158749L",
            "nombre": "Pepe",
            "apellidos": "Adamuz Núñez",
            "telefono": 666956666,
            "email": "pepeadamuz@gmail.com"
         },
         {
            "DNI": "36214758I",
            "nombre": "Dolores",
            "apellidos": "Pérez Aguilera",
            "telefono": 684766666,
```

```
            "email": "doloresperez@gmail.com"
        }
    ]
  }
]
```


Donde "estado": 1 será creación exitosa, tal como hemos definido en el fichero mensajes.php del directorio datos de nuestro proyecto y dentro del “mensaje” un array con todas las personas de nuestra base de datos.

Ahora veremos cómo obtener una persona con su DNI. En el fichero ControladorPersonas.php tenemos nuestra funcion obtenerPersonaDNI donde hay un atributo (?) en la sentencia WHERE que deberemos de indicar su valor, y para ello usamos el método bindParam:

```
$sentencia->bindParam(1, $DNI);
```

Indicando la posición de la interrogación y el atributo, ejecutamos la sentencia:

```
$sentencia→execute();
```

Obtenemos todos los resultyados y los devolvemos (como vemos en el fichero ControladorPersonas.php...)

Ahora creamos un método obtenerPersonaDNI que es el que tenemos en el fichero que lleva su nombre, en la raiz del proyecto. La función obtenerPersonaDNI al final de este fichero, tiene un parámetro $dni que se introducirá mediante la url que introdujimos antes, pero con una modificación:

localhost:50/personas/obtenerPersonaDNI.php**?DNI=14786325P**

Donde habrá que añadir una **interrogación (?)** despues del fichero que contiene el método, seguido del **nombre del parámetro (DNI)** y su **valor (14786325P)**, que será el que tenga en nuestra base de datos personas en la tabla Persona, creada anteriormente.

Obteniendo el siguiente resultado en nuestro navegador (servidor):


```
[
   {
      "estado": 1,
      "mensaje": [
         {
            "DNI": "14786325P",
            "nombre": "Juan",
            "apellidos": "López López",
            "telefono": 666666666,
            "email": "juanlopez@gmail.com"
         }
      ]
   }
]
```


Todo esto se consigue gracias al método:

```
        $dni = $_REQUEST['DNI'];
```

Y a la vista, que imprime el resultado obtenido con el controlador (ambos métodos usados en la clase obtenerPersonaDNI que acabamos de ejecutar en el servidor Apache:

```
        $vista→imprimir($controladorp→obtenerPersonaDNI($dni));
```

Si queremos insertar una persona, en nuestra clase ControladorPersona tenemos el método insertarPersona($persona), en el cual le pasaremos la persona que queramos agregar:

```
// Obtengo una instancia de la base de datos ya conectada
$pdo = ConexionBD::obtenerInstancia()->obtenerBD();
// Creo la sentencia INSERT
$comando = "INSERT INTO " . self::NOMBRE_TABLA . " ( " .
```

```
            self::DNI . "," .
            self::NOMBRE . "," .
            self::APELLIDOS . "," .
            self::TELEFONO . "," .
            self::EMAIL . ")" .
            " VALUES(?,?,?,?,?)";
$sentencia = $pdo->prepare($comando);
// Pongo los datos en la consulta INSERT
$sentencia->bindParam(1, $persona->DNI);
$sentencia->bindParam(2, $persona->nombre);
$sentencia->bindParam(3, $persona->apellidos);
$sentencia->bindParam(4, $persona->telefono);
$sentencia->bindParam(5, $persona->email);
// Ejecuto la consulta
$resultado = $sentencia->execute();
```

Cuando llamemos a este método deberemos de enviar más de un parámetro, en la url del navegador, los cuales se agregarán añadiendo un ámpersan (&) seguido del nombre del parámetro igual (=) a su valor. Con el carácter ámpersan podremos poner tantos atributos como necesitemos, en nuestro caso deberíamos de poner 5 (DNI, Nombre, Apellidos, Telefono y Email), tal y como aparecen en nuestra clase insertarPersona.php.

Conexión a MYSQL desde PHP7

Como ya sabemos, en este tema usaremos **PHP** como un **intermediario** entre nuestra aplicación **Android** y nuestra **base de datos MySQL** en un servidor «**remoto**» (aunque estará en local si usamos XAMPP). Vamos a ver cómo podemos acceder con PHP 7 a MySQL.

Como **PHP 7** es un lenguaje de programación orientado a objetos, vamos a crear una **clase ConexionDB** para **conectarlo a la base de datos**.

Por ahora (sabiendo que vamos a utilizar el **Modelo Vista-Controlador** más adelante), crearemos una **carpeta**, llamada «**datos**», que será la encargada de contener todo lo relacionado con el acceso a la base de datos y que contendrá los siguientes ficheros:

- **ConexionBD.php**: Este fichero tendrá una clase que se encargará de la **conexión y desconexión** de la base de datos.
- **login_mysql.php**: Este fichero tendrá todos los **datos** necesarios para realizar la **conexión** a la base de datos.
- **mensajes.php**: Contendrá los **mensajes y variables** que vamos a utilizar en el proyecto.

En el fichero **login_mysql.php**, tendremos que definir:

las variables que nos indicarán el **nombre del host** donde vamos a conectarnos (**localhost**, en nuestro caso),

el nombre de la **base de datos,**

los **datos de conexión del usuario**.

Esto lo podremos hacer con el **método define**, que definirá una **variable global** a nuestro proyecto.

Para realizar la **conexión** a la base de datos, utilizaremos la **clase de PHP 7** que nos provee dicha funcionalidad, a la cual tendremos que **pasarle todos los datos** anteriores. Como estamos utilizando una conexión a una base de datos, tendremos que **desconectarnos** cuando **terminemos** nuestras operaciones, lo cual podremos hacer en el **destructor** de la clase. Un destructor se llamará automáticamente cuando se destruya el objeto. Este _destructor estará definido en la clase ConexionBD.php.

Definición de variables en la clase **login_mysql.php**:

```
define("NOMBRE_HOST", "localhost");       // Nombre del host
define("BASE_DE_DATOS", "personas");// Nombre de la base de modelos
define("USUARIO", "root");                // Nombre del usuario
define("CONTRASENA", "");                 // Constraseña
```

Clase **ConexionBD.php**:

```php
<?php

/**
 * Esta clase sirve para poder realizar una conexión y desconexión a la base de datos
 * MySQL de nuestro servidor.
 */

require_once('login_mysql.php');

class ConexionBD
{
    const ESTADO_ERROR_BD = 3;

    /**
     * Atributo para la conexión de la base de datos
     */
    private static $db = null;

    /**
     * Instancia de PDO (extensión Objetos de Datos de PHP,
     * capa de abstracción de acceso a datos)
     */
    private static $pdo;

    /**
     * Constructor de la clase. Conecta con la base de datos.
     */
    final private function __construct()
    {
        try
        {
            // Crear nueva conexión PDO
            self::obtenerBD();
        }
        catch (PDOException $e)
        {
            // Manejo de excepciones
            throw new ExcepcionApi(ESTADO_ERROR_BD, $e->getMessage());
        }

    }
```

```php
/**
 * Retorna en la única instancia de la clase
 * @return ConexionBD|null
 */
public static function obtenerInstancia()
{
    if (self::$db === null)
    {
        self::$db = new self();
    }
    return self::$db;
}

/**
 * Crear una nueva conexión PDO basada
 * en las constantes de conexión
 * @return PDO Objeto PDO
 */
public function obtenerBD()
{
    if (self::$pdo == null)
    {
        self::$pdo = new PDO(
            'mysql:dbname=' . BASE_DE_DATOS .
            ';host=' . NOMBRE_HOST . ";",
            USUARIO,
            CONTRASENA,
            array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8")
        );

        // Habilitar excepciones
        self::$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    }

    return self::$pdo;
}

/**
 * Evita la clonación del objeto
 */
final protected function __clone() {}

/**
 * Desctructor de la clase. Cierra la conexión a la base de datos.
 */
```

```
    function _destructor()
    {
        self::$pdo = null;
    }
}

?>
```

Clase **mensajes.php**:

```
<?php

if (!defined('ESTADO_CREACION_EXITOSA'))
        define('ESTADO_CREACION_EXITOSA', 1);
if (!defined('ESTADO_CREACION_FALLIDA'))
        define('ESTADO_CREACION_FALLIDA', 2);
if (!defined('ESTADO_ERROR_BD'))
        define('ESTADO_ERROR_BD', 3);
if (!defined('ESTADO_AUSENCIA_CLAVE_API'))
        define('ESTADO_AUSENCIA_CLAVE_API', 4);
if (!defined('ESTADO_CLAVE_NO_AUTORIZADA'))
        define('ESTADO_CLAVE_NO_AUTORIZADA', 5);
if (!defined('ESTADO_URL_INCORRECTA'))
        define('ESTADO_URL_INCORRECTA', 6);
if (!defined('ESTADO_FALLO_DESCONOCIDO'))
        define('ESTADO_FALLO_DESCONOCIDO', 7);
if (!defined('ESTADO_PARAMETROS_INCORRECTOS'))
        define('ESTADO_PARAMETROS_INCORRECTOS', 8);

if (!defined('error'))
        define('error',
                [
                        [
                        "estado" => ESTADO_FALLO_DESCONOCIDO,
                                "mensaje" => utf8_encode("Error desconocido.")
                        ]
                ]
        );

if (!defined('correcto'))
        define('correcto',
                [
                        [
```

```
                    "estado" => ESTADO_CREACION_EXITOSA,
                            "mensaje" => utf8_encode("OK")
                    ]
            ]
        );

?>
```

Tipo de base de datos remota que conviene utilizar

En cuanto a tipos de bases de datos, no solo existen las bases de datos relacionales, sino que hay otros tipos, como las **bases de datos XML**, las cuales utilizan ficheros XML para almacenar los datos, **prescindiendo de tablas, de claves y de demás conceptos**.

Las **bases de datos relacionales** también pueden utilizarse en servidores remotos, pero tienen el inconveniente de que, **si son muy grandes**, pueden llegar a ser **lentas**, y eso no es muy bueno en una conexión remota.

Para usos de **bases de datos pequeñas o medianas**, podríamos usar una **base de datos relacional**, y, si tuviéramos que utilizar una **base de datos muy grande**, podríamos intentar utilizar otros tipos de bases de datos, ya que, en bases de datos grandes, las relacionales pueden volverse muy lentas. Las bases de datos no relacionales se utilizan en escenarios donde la escalabilidad, la velocidad y la flexibilidad son prioritarias.

Algunas grandes empresas que utilizan bases de datos **no relacionales** son: Facebook, Twitter, Youtube... debido a la ingente cantidad de información que manejan.

Operaciones en MYSQL desde PHP7

Las operaciones que podremos realizar **desde PHP** sobre nuestra **base de datos MySQL** son las siguientes:

- **Conexión** con la base de datos.
- **Desconexión** de la base de datos.
- **Inserción** de datos.
- **Borrado** de datos.
- **Actualización** de datos.
- **Recuperación** de datos.

Mediante el **método execute:**

→ podremos ejecutar, mediante sus **instrucciones SQL** pertinentes, las operaciones en datos:

- **Inserción**,
- **Borrado**,
- y **Actualización**.

Las operaciones de **selección de datos** podremos realizarlas mediante una consulta **SELECT** (tan compleja como la necesitemos), y podremos **ejecutarlas mediante el método execute**, que nos devolverá un **cursor** con todos los datos a devolver.

Para este tipo de tareas, utilizaremos el **modelo Vista-Controlador**, como ya comentamos anteriormente, y tendremos una estructura de carpetas como la que mostramos a continuación:

MVC de un proyecto PHP básico:

Explicación Modelo Vista-Controlador de un proyecto PHP básico:

Controladores (Todas las clases que son controladores de nuestros modelos):

ControladorPersonas.php (permitirá realizar operaciones con las personas)

Datos (todo lo que tenga que ver con la conexión a la BBDD):

ConexionBD.php

login_mysql.php

mensajes.php

Modelos (Todas la clases que representen un modelo en nuestro proyecto):

Persona.php

Utilidades (Todas las clases que nos proporcionarán una utilidad diferente):

ExcepcionApi.php (será una API para el control de excepciones en nuestro proyecto)

Vistas (Todos los ficheros que tienen que ver con las vistas):

VistaApi.php

VistaJson.php

(Nos permitirán formatear los resultados con formato JSON)

En la C**arpeta raíz del proyecto** (corresponden con las operaciones que vamos a hacer):

insertarPersona.php

obtenerPersonaDNI.php

obrtenerTodasPersonas.php