DISEÑO MONOLÍTICO DE PÁGINAS WEB

CRUD (Create (Crear), Read (Leer), Update (Actualizar), Delete (Borrar))

Base de datos

Ingresamos a MariaDB primero:
sudo mariadb -u root -p

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo mariadb -u root -p
[sudo] password for rodrigo:
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 5
Server version: 10.11.14-MariaDB-0+deb12u2 Debian 12
```

Note que se nos pide el password del usuario para ejecutar en WSL el programa MariaDB y después nos pide el password para ingresar como root a la base de datos, esto es común al iniciar la sesión de WSL como en este caso.
Creamos la base de datos para el proyecto con el siguiente comando:
CREATE DATABASE proyecto_lamp;

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 5
Server version: 10.11.14-MariaDB-0+deb12u2 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| sys                |
+--------------------+
5 rows in set (0.030 sec)

MariaDB [(none)]> CREATE DATABASE proyecto_lamp;
Query OK, 1 row affected (0.013 sec)

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| proyecto_lamp      |
| sys                |
+--------------------+
6 rows in set (0.001 sec)

MariaDB [(none)]>
```

Es buena práctica evitar usar al usuario *root* al administrador una base de datos específica para cada proyecto, así que crearemos un usuario administrador para esta base de datos específica y le otorgamos los permisos para gestionarla.
CREATE USER 'admin_proyecto'@'localhost' IDENTIFIED BY 'password_seguro_123';
No olvide cambiar el nombre de usuario ‘admin_proyecto’ por el de su preferencia así como en 'password_seguro_123' reescribir el password para este usuario. para este ejemplo usaremos usuario admin y contraseña 1234.
Y le otorgamos el control de la base de datos:
GRANT ALL PRIVILEGES ON proyecto_lamp.* TO 'admin'@'localhost';

```
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE USER 'admin'@'localhost' IDENTIFIED BY '1234';
Query OK, 0 rows affected (0.165 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON proyecto_lamp.* TO 'admin'@'localhost';
Query OK, 0 rows affected (0.033 sec)

MariaDB [(none)]>
```

Reiniciamos la tabla de privilegios de la base de datos:
FLUSH PRIVILEGES;
salimos con EXIT; y volvemos a ingresar pero ahora directamente a la base de datos recién creada.

```
MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.079 sec)

MariaDB [(none)]> EXIT;
Bye
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ mariadb -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 7
Server version: 10.11.14-MariaDB-0+deb12u2 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| proyecto_lamp      |
+--------------------+
2 rows in set (0.072 sec)

MariaDB [(none)]>
```

Listo ahora contamos con una base de datos y un usuario administrador.

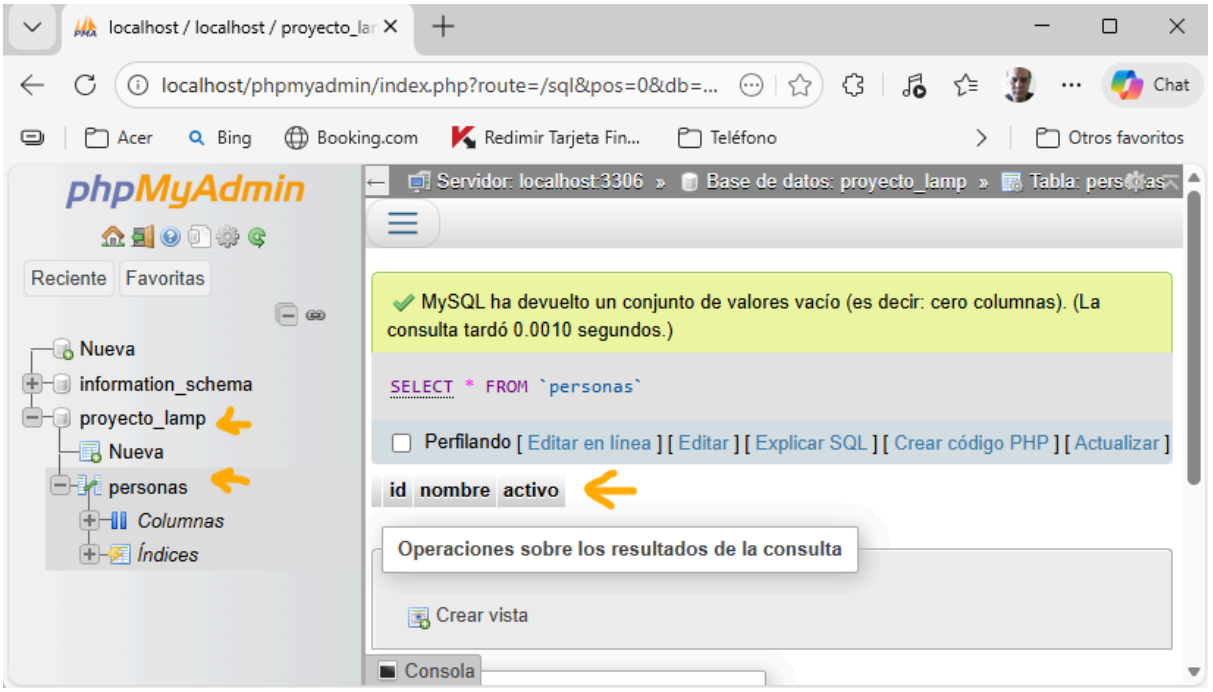
Seleccionamos la base de datos “proyecto_lamp” para trabajar en ella mediante el comando “USES proyecto_lamp;” y creamos la tabla con tres campos:
CREATE TABLE personas (id INT AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(100) NOT NULL, activo TINYINT(1) DEFAULT 1 -- 1 significa activo, 0 significa borrado lógico);

```
MariaDB [(none)]> USE proyecto_lamp;
Database changed
MariaDB [proyecto_lamp]> CREATE TABLE personas (
id INT    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     nombre VARCHAR(100) NOT NULL,
    ->     activo TINYINT(1) DEFAULT 1 -- 1 significa activo, 0 significa borrado lógico
    -> );
Query OK, 0 rows affected (0.173 sec)

MariaDB [proyecto_lamp]>
```

Verificamos la creación de la tabla:

```
MariaDB [proyecto_lamp]> SHOW TABLE personas;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'personas' at line 1
MariaDB [proyecto_lamp]> SHOW TABLES;
+------------------------+
| Tables_in_proyecto_lamp |
+------------------------+
| personas               |
+------------------------+
1 row in set (0.066 sec)

MariaDB [proyecto_lamp]> DESCRIBE personas;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| nombre | varchar(100) | NO   |     | NULL    |                |
| activo | tinyint(1)  | YES  |     | 1       |                |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.073 sec)

MariaDB [proyecto_lamp]>
```

Verificamos la configuración de la base de datos para el proyecto en la aplicación de phpmyadmin en la dirección http://localhost/phpmyadmin, nos ingresamos con el usuario y contraseña(admin 1234 para este caso).



Podemos ver que solo se muestra una base de datos y una tabla vacía.

Creamos una carpeta para almacenar nuestro archivos del proyecto, y para conectar a la base de datos usando PHP, usando el código siguiente dentro de un archivo llamado conexion.php:

src > proyecto_lamp > conexion.php

```php
<?php
// Configuración de las credenciales
$host = "localhost";
$user = "admin";      // El usuario que creamos
$pass = "1234"; // La contraseña que asignaste
$db   = "proyecto_lamp";       // El nombre de la base de datos

// Crear la conexión usando la extensión mysqli
$conn = mysqli_connect($host, $user, $pass, $db);

// Verificar si la conexión falló
if (!$conn) {
    // En producción es mejor no mostrar el error detallado, pero para desarrollo es vital
    die("Fallo la conexión: " . mysqli_connect_error());
}

// Configurar el juego de caracteres a UTF-8 para evitar problemas con acentos o eñes
mysqli_set_charset($conn, "utf8");

// Si llegamos aquí, la conexión fue exitosa
// echo "Conexión establecida correctamente"; // Descomenta para probar
?>
```

y otro archivo llamado test.php para probar en el navegador que la conexión funciona:

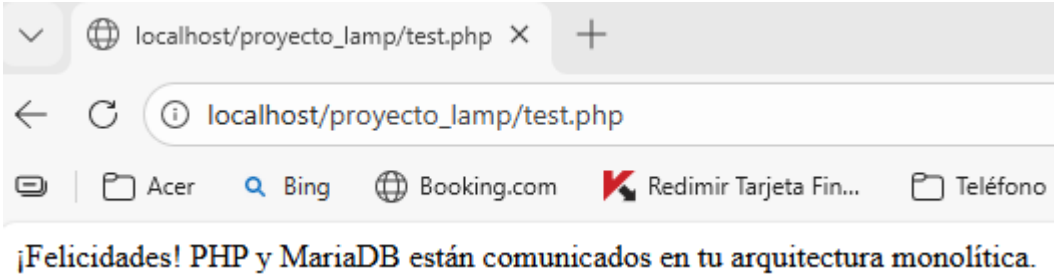
src > proyecto_lamp > test.php

```php
<?php
include 'conexion.php';
echo "¡Felicidades! PHP y MariaDB están comunicados en tu arquitectura monolítica.";
?>
```

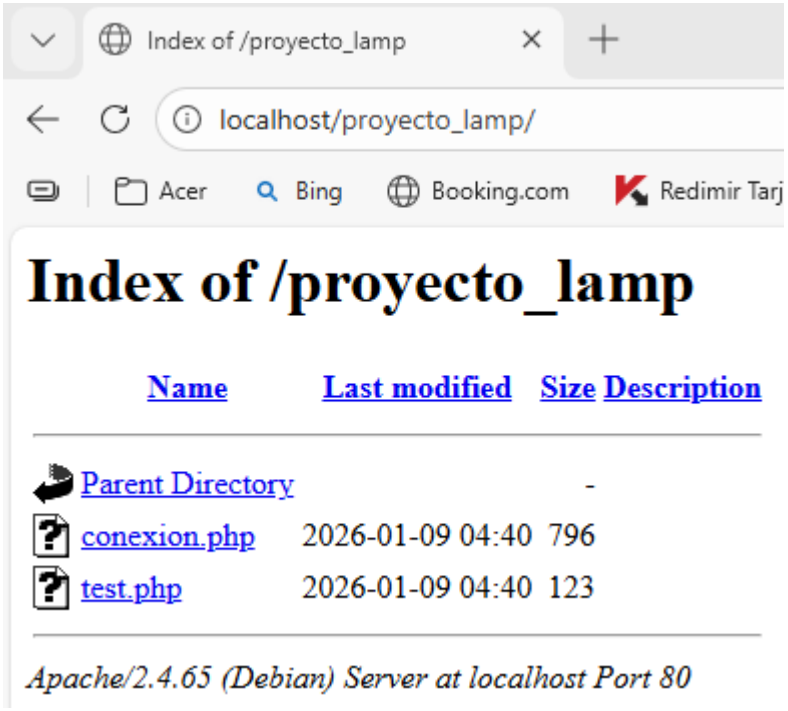
Copiamos estos archivos a la carpeta */var/www/html* donde se alojarán las proyectos de páginas web en el servidor web Apache:

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ ls
Documentacion  README.md  src
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ ls src/
index.php  proyecto_lamp
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo cp -r src/proyecto_lamp/ /var/www/html
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ ls /var/www/html
index.html  info.php  proyecto_lamp
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ ls /var/www/html/proyecto_lamp/
conexion.php  test.php
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$
```

Lo visualizamos en el navegador en la dirección http://localhost/proyecto_lamp/test.php,



Si pudimos ver el mensaje anterior, la conexión fue un éxito, note que si solo especificamos la dirección de la carpeta sin ningún archivo se nos mostrará el contenido del directorio si usamos por ejemplo Index of /proyecto_lamp:



La excepción es el archivo de nombre *index.php* el cual se muestra por defecto aunque no lo especifiquemos explícitamente, parte de la seguridad consiste en evitar mostrar carpetas con directorios al usuario, así llamaremos a todas los archivos que se muestran como páginas web FrontEnd mientras que aquellos que están ahí pero no se pueden ver como BackEnd.

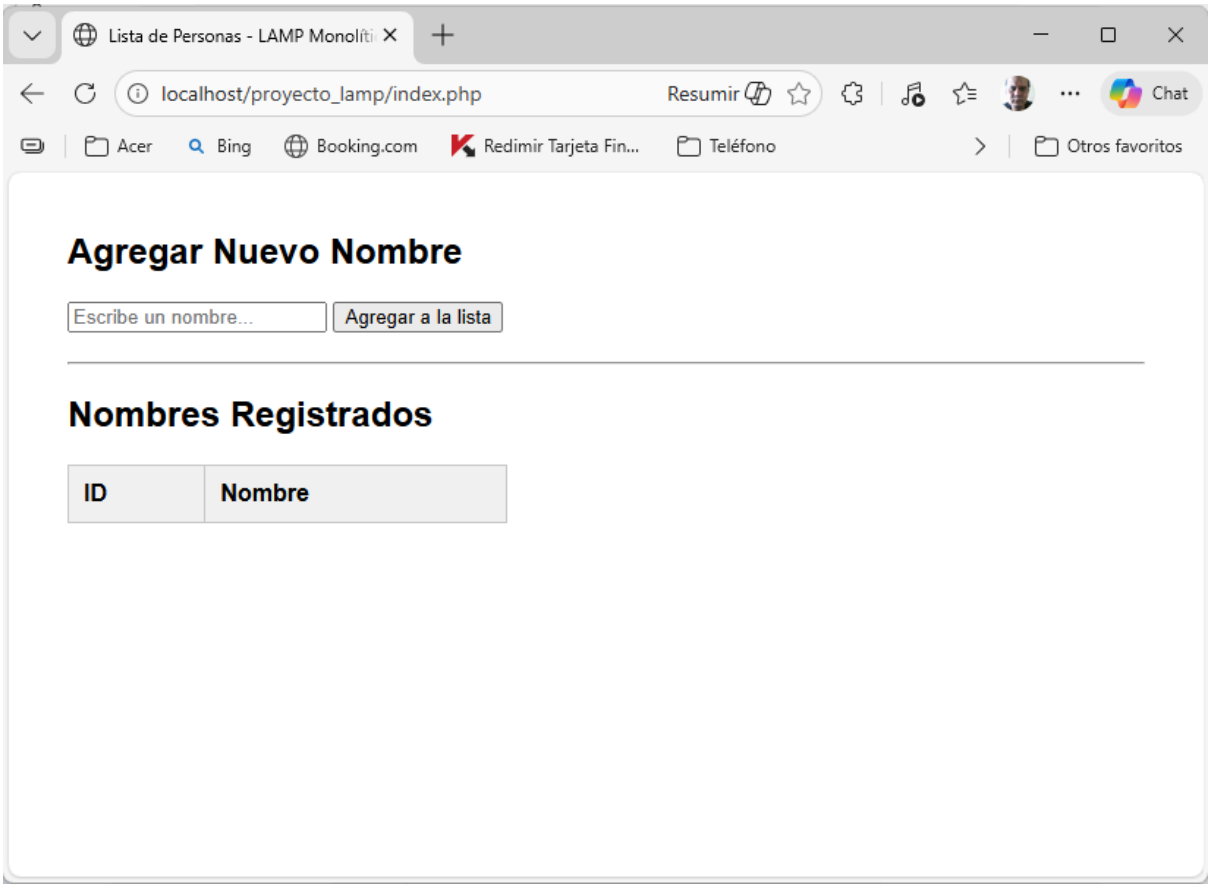
Es hora de crear el formato para ingresar datos a nuestra lista desde una página web, es decir la primera acción de nuestro CRUD, cree o mueva un archivo de nombre index.php con el siguiente código:

```php
<?php
// Incluimos la conexión que ya probamos
include 'conexion.php';

// Lógica para Insertar (esto se ejecuta cuando presionas el botón)
if (isset($_POST['agregar'])) {
    $nombre = mysqli_real_escape_string($conn, $_POST['nombre']);
    if (!empty($nombre)) {
        $sql_insert = "INSERT INTO personas (nombre, activo) VALUES
('$nombre', 1)";
        mysqli_query($conn, $sql_insert);
        // Recargar la página para limpiar el formulario y ver el nuevo
nombre
        header("Location: index.php");
    }
}
```

```
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Lista de Personas - LAMP Monolítico</title>
    <style>
        body { font-family: sans-serif; margin: 40px; }
        table { border-collapse: collapse; width: 300px; margin-top:
20px; }
        th, td { border: 1px solid #ccc; padding: 10px; text-align:
left; }
        th { background-color: #f4f4f4; }
        .formulario { margin-bottom: 20px; }
    </style>
</head>
<body>

    <h2>Agregar Nuevo Nombre</h2>

    <div class="formulario">
        <form method="POST" action="index.php">
            <input type="text" name="nombre" placeholder="Escribe un
nombre..." required>
            <button type="submit" name="agregar">Agregar a la
lista</button>
        </form>
    </div>

    <hr>

    <h2>Nombres Registrados</h2>
    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Nombre</th>
            </tr>
        </thead>
        <tbody>
            <?php
```

```
            // Solo seleccionamos los registros que NO tengan borrado
lógico
            $query = "SELECT id, nombre FROM personas WHERE activo =
1";
            $result = mysqli_query($conn, $query);

            while ($row = mysqli_fetch_assoc($result)) {
                echo "<tr>";
                echo "<td>" . $row['id'] . "</td>";
                echo "<td>" . $row['nombre'] . "</td>";
                echo "</tr>";
            }
            ?>
        </tbody>
    </table>

</body>
</html>
```

Verifique el resultado en la dirección web [Index of /proyecto_lamp]:



Cada vez que agreguemos una nueva entrada esta se listará en la parte inferior de la página mediante una tabla dinámica, la cual mostrará sólo los registros activos.

Las partes clave del código del archivo *index.php* son las contenidas en las marcas de lenguaje html como la que contiene las instrucciones para mostrar el cuadro de texto y el botón para agregar nuevos registros a nuestra base de datos,:

```
<div class="formulario"> <form method="POST" action="index.php"> <input type="text" name="nombre" placeholder="Escribe un nombre..." required> <button type="submit" name="agregar">Agregar a la lista</button> </form> </div>
```

Mediante la llamada a la función que hace la Query para insertar el dato en el registro de la tabla en la base de datos:

```
// Lógica para Insertar (esto se ejecuta cuando presionas el botón) if (isset($_POST['agregar'])) { $nombre = mysqli_real_escape_string($conn, $_POST['nombre']); if (!empty($nombre)) { $sql_insert = "INSERT INTO personas (nombre, activo) VALUES ('$nombre', 1)"; mysqli_query($conn, $sql_insert); // Recargar la página para limpiar el formulario y ver el nuevo nombre header("Location: index.php"); } } ?>
```

La sección que sirve para mostrar los registros está compuesta por una Query que recupera los datos y una ciclo *while* que crea dinámicamente la tabla:

```
<?php // Solo seleccionamos los registros que NO tengan borrado lógico $query = "SELECT id, nombre FROM personas WHERE activo = 1"; $result = mysqli_query($conn, $query); while ($row = mysqli_fetch_assoc($result)) { echo "<tr>"; echo "<td>" . $row['id'] . "</td>"; echo "<td>" . $row['nombre'] . "</td>"; echo "</tr>"; } ?>
```

Se incluye al principio el archivo la referencia al archivo *conexion.php* que crea el puente de acceso hacia la base de datos.

```
// Incluimos la conexión que ya probamos include 'conexion.php';
```

Y las definiciones de estilo para la apariencia de la página web;

```
<style> body { font-family: sans-serif; margin: 40px; } table { border-collapse: collapse; width: 300px; margin-top: 20px; } th, td { border: 1px solid #ccc; padding: 10px; text-align: left; } th { background-color: #f4f4f4; } .formulario { margin-bottom: 20px; } </style>
```

Con esto hemos procesado dos de los requerimientos de implementación de un CRUD Create (Crear), Read (Leer), Creamos el archivo editar.php el cual permitirá editar el campo nombre de los registros:

```php
// 2. Procesar la actualización
if (isset($_POST['actualizar'])) {
    $id_update = $_POST['id'];
    $nuevo_nombre = mysqli_real_escape_string($conn, $_POST['nombre']);

    $sql_update = "UPDATE personas SET nombre = '$nuevo_nombre' WHERE id = $id_update";

    if (mysqli_query($conn, $sql_update)) {
        header("Location: index.php"); // Regresar al inicio
    } else {
        echo "Error al actualizar: " . mysqli_error($conn);
    }
```

```
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Editar Registro</title>
</head>
<body>
    <h2>Editar Nombre</h2>
    <form method="POST" action="editar.php">
        <input type="hidden" name="id" value="<?php echo
$persona['id']; ?>">

        <input type="text" name="nombre" value="<?php echo
$persona['nombre']; ?>" required>
        <button type="submit" name="actualizar">Guardar
Cambios</button>
        <a href="index.php">Cancelar</a>
    </form>
</body>
</html>
```

Así como el archivo *borrar.php* que permite borrar lógicamente un registro (el registro sique en la base de datos, solo no se muestra):

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

if (isset($_GET['id'])) {
    $id = mysqli_real_escape_string($conn, $_GET['id']);

    // Implementamos el Borrado Lógico: cambiamos activo de 1 a 0
    $sql_borrar = "UPDATE personas SET activo = 0 WHERE id = $id";

    if (mysqli_query($conn, $sql_borrar)) {
        // Redirigir al index para ver los cambios
        header("Location: index.php");
    } else {
        echo "Error al eliminar el registro: " . mysqli_error($conn);
    }
} else {
    // Si alguien intenta entrar a borrar.php sin un ID, lo regresamos
al inicio
```

```
    header("Location: index.php");
}
?>
```

Además debemos modificar la sección que construye la tabla en el archivo index.php para que se muestre el botón para editar el campo en la lista así como el botón de borrar:

```
while ($row = mysqli_fetch_assoc($result)) { echo "<tr>"; echo "<td>" . $row['id'] . "</td>"; echo "<td>" . $row['nombre'] . "</td>"; echo "<td> <a href='editar.php?id=" . $row['id'] . "'>Editar</a> | <a href='borrar.php?id=" . $row['id'] . "'>Eliminar</a> </td>"; echo "</tr>"; }
```

Copiamos o movemos los archivos a la carpeta */var/www/html/proyecto_crud*

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo cp -r src/proyecto_lamp/ /var/www/html
[sudo] password for rodrigo:
Sorry, try again.
[sudo] password for rodrigo:
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ ls /var/www/html/proyecto_lamp/
conexion.php  editar.php  index.php  test.php
```

Y probamos en el navegador en la página [Lista de Personas - LAMP Monolítico](), veremos lo siguiente:

Al hacer click en el botón detalle veremos la siguiente página web:

Si elegimos el botón *Guardar Cambios* los cambios se efectuarán y nos regresará a la página anterior lo mismo para el enlace *Cancelar*, solo que este como su nombre implica no se efectuará acción alguna. Con esto se ha completado las acciones del CRUD completo, el proyecto está completado.

CONTENERIZACIÓN DEL DISEÑO MONOLÍTICO

Ahora que tenemos el proyecto ejecutándose correctamente podemos pasar a contenerizar sin preocuparnos en reparar errores en la interacción de los archivos del CRUD, así que los errores que puedan presentarse serán solo al realizar el proceso de contenerización, partiendo de lo que aprendimos anteriormente requerimos dos servicios para poder lanzar nuestra página web uno es el servidor web y la base de datos, en nuestro caso el LAMP server utiliza Apache y MariaDB, y aunque existe una versión de Apache para los contenedores cambiaremos de servidor web a Nginx porque es más adecuado para contenerización y que nos permitirá mover el proyecto de diseño estático a uno de microservicios en el futuro (proceso que haremos de manera gradual).

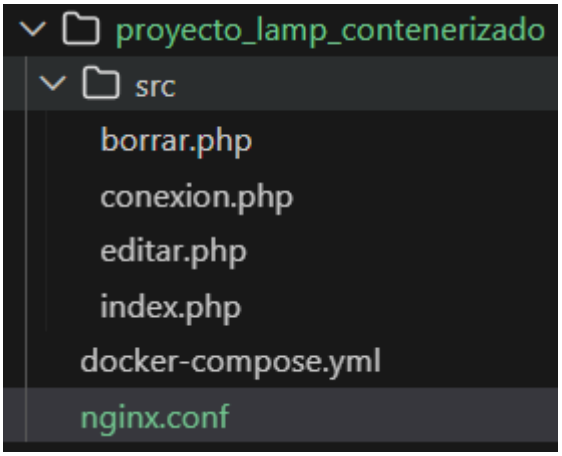
Debido a este cambio de software, los contenedores quedarán como sigue:

Nginx: El servidor web.

PHP-FPM: El procesador de PHP (Nginx no procesa PHP por sí solo, a diferencia de Apache).

MySQL/MariaDB: La base de datos.

Estos tres contenedores interactúan entre sí y funcionan como uno solo proceso, por lo que en nuestra carpeta del proyecto se verá ahora así:



Comenzamos describiendo el propósito del directorio *src* con las páginas web del proyecto, este será redireccionado al archivo */var/www/html* dentro del contenedor de nginx el cual las mostrará en el navegador (justo como con Apache). Como cada vez que creamos un contenedor este estará en cero, tendremos que configurar nuevamente el ambiente adecuado para que nuestra aplicación se ejecute correctamente (homogeneizar un ambiente de ejecución es el objetivo de usar Docker). Así el archivo nginx.conf le indicará al programa del servidor web (dentro del contenedor) como manejar los archivos *.php*, este archivo se enlaza con el archivo de configuración del servidor para asegurar que pueda mostrar las páginas web correctamente cada vez que se inicia el contendor, por ultimo el archivo docker-compose.yml contiene las definiciones de las imágenes de los tres contendores que debemos levantar, esto permite que en un solo paso poner en línea los tres contenedores y evita también que tengamos tres archivos de imagen para cada contenedor.

El contenido del archivo *docker-compose.yml* es el siguiente:

```
services:
  # Servidor Web Nginx
  web:
    image: nginx:alpine
    ports:
      - "8080:80"
    volumes:
      - ./src:/var/www/html
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - php

  # Procesador PHP
  php:
    image: php:8.2-fpm
    environment:
      # Pasamos las variables al sistema operativo del contenedor PHP
      MYSQL_DATABASE: nombre_base_datos
      MYSQL_USER: usuario_db
      MYSQL_PASSWORD: password_db
    volumes:
      - ./src:/var/www/html
    # Instalamos la extensión necesaria para MySQL
    command: >
      sh -c "docker-php-ext-install mysqli && php-fpm"

  # Base de Datos
  db:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: root_password
      MYSQL_DATABASE: nombre_base_datos
      MYSQL_USER: usuario_db
      MYSQL_PASSWORD: password_db
    volumes:
      - db_data:/var/lib/mysql

volumes:
  db_data:
```

Donde la sección para la base de datos debe modificarse acorde al contenido del archivo *conexion.php*:

```
// Configuración de las credenciales
```

```
$host = "db"; //antes "localhost";//IP del servidor de base de datos
$user = getenv('MYSQL_USER');       // El usuario que creamos
$pass = getenv('MYSQL_PASSWORD'); // La contraseña que asignaste
$db   = getenv('MYSQL_DATABASE'); // El nombre de la base de datos
```

Note también que hemos definido los nombres de los contendores a *web, php* y *db*.
Y la sección docker-compose.yml para la base de datos debe quedar así:

```
 db:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: Megamanzero
      MYSQL_DATABASE: proyecto_lamp
      MYSQL_USER: admin
      MYSQL_PASSWORD: 1234
    volumes:
      - db_data:/var/lib/mysql
```

Y la sección del procesador de PHP la modificamos así:

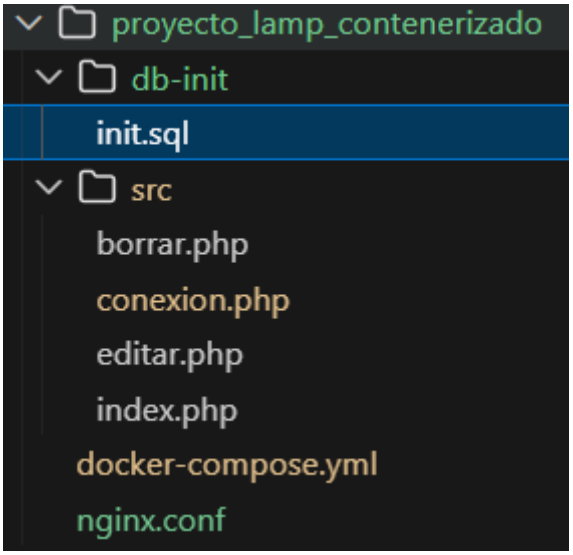
```
php:
    image: php:8.2-fpm
    environment:
      # Pasamos las variables al sistema operativo del contenedor PHP
      MYSQL_DATABASE: proyecto_lamp
      MYSQL_USER: admin
      MYSQL_PASSWORD: "1234"
    volumes:
      - ./src:/var/www/html
    # Instalamos la extensión necesaria para MySQL
    command: >
      sh -c "docker-php-ext-install mysqli && php-fpm"
```

El archivo *nginx.conf* tiene el siguiente contenido:

```
server {
    listen 80;
    index index.php index.html;
    server_name localhost;
    root /var/www/html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass php:9000; # Apunta al servicio 'php' definido en
docker-compose
```

```
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

Adicionalmente tendremos que crear el archivo de script, para crear la tabla que usará la base de datos ya que cada que se inicializa un contenedor debe configurar todo desde cero. Por lo que la **primera vez** que ejecutamos el proyecto la base de datos está totalmente vacía. Para automatizar el levantamiento de la base de datos, creamos un archivo llamado *init.sql* dentro del directorio db-init:



Dicho archivo *init.sql* contendrá lo siguiente:

```
CREATE TABLE IF NOT EXISTS personas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    activo TINYINT(1) DEFAULT 1
);

INSERT INTO personas (nombre, activo) VALUES ('Usuario Inicial', 1),
('Prueba Docker', 1);
```

Modificamos la sección *db* del archivo *docker-compose.yml* con la siguiente linea:

```
  db:
    image: mariadb:latest
    environment:
      #. #no se muestra la información
      #. #de las variables de ambiente
      #. #para acortar
    volumes:
      - db_data:/var/lib/mysql
      - ./db-init:/docker-entrypoint-initdb.d # <-- Agrega esta línea
```

Ahor estamos listos para “levantar” el entorno, asegure estar en el directorio raíz del proyecto (en este caso .”/*proyecto_lamp_contenerizado*”), y ejecute los siguientes comandos, el primero para ejecutar los contenedores en segundo plano y el segundo para verificar que los contenedores están en ejecución:

$ docker compose up -d

$ docker compose ps

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/proyecto_lamp_contenerizado$ docker compose up -d
[+] up 6/31
 Image mariadb:latest [.          ] Pulling
 Image php:8.2-fpm [              ] Pulling
 Image nginx:alpine [          ] Pulling
```

La primera vez se descargan los archivos requeridos para crear los contenedores

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/proyecto_lamp_contenerizado$ docker compose ps
NAME                                IMAGE            COMMAND                  SERVICE   CREATED         STATUS         PORTS
proyecto_lamp_contenerizado-db-1    mariadb:latest   "docker-entrypoint.s…"   db        4 minutes ago   Up 3 minutes   3306/tcp
proyecto_lamp_contenerizado-php-1   php:8.2-fpm      "docker-php-entrypoi…"   php       4 minutes ago   Up 3 minutes   9000/tcp
proyecto_lamp_contenerizado-web-1   nginx:alpine     "/docker-entrypoint.…"   web       3 minutes ago   Up 3 minutes   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
```

Deberá ver que los contenedores tiene el *STATUS* “run” o “up”, y probar en el navegador en la dirección: http://localhost:8080 , note que hemos agregado el número de puerto (justo como lo definimos en el archivo *docker-compose.yml*), para evitar colisionar con el puerto usado por Apache (puerto 80 para http). Pruebe que todas las acciones del CRUD se ejecutan normalmente.Si por alguna razón debe borrar todo para comenzar desde cero nuevamente utilice el comando siguiente:

docker compose down -v

Este comando detiene los contenedores con la opción *down* y elimina los archivos en los volúmenes persistentes, si solo desea detener los contenedores use solamente *down*.

Detenemos la ejecución de los contenedores:

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/proyecto_lamp_contenerizado$ docker compose down
[+] down 4/4
 ✓ Container proyecto_lamp_contenerizado-web-1 Removed
 ✓ Container proyecto_lamp_contenerizado-db-1  Removed
 ✓ Container proyecto_lamp_contenerizado-php-1 Removed
 ✓ Network proyecto_lamp_contenerizado_default Removed
```

Los volvemos a iniciar nuevamente, para verificar que en esta ocasión no se inició la base de datos y aparte los datos anteriores se mantuvieron (persistencia).

Si desea ver los registros de los contenedores en tiempo real utilice:

$ docker compose logs -f

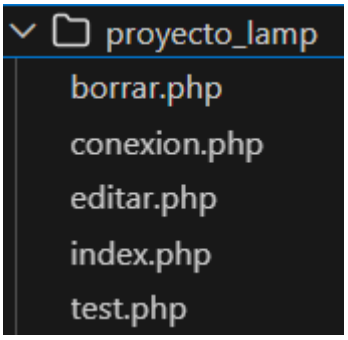
Si desea reiniciar un contenedor en específico (por ejemplo el contenedor web), use:

$ docker compose restart web

Con esto hemos contenerizado nuestro proyecto, solo resta indicar un detalle adicional, el proceso de crear un contenedor docker involucra dos pasos uno es crear una imagen que será la plantilla del contenedor y el otro es propiamente lanzar el contenedor, el proceso es similar a la clase y el objeto en la programación orientada a objetos, donde la clase es la plantilla que define al objeto y este es en última instancia creado solo en tiempo de ejecución, lo mismo aplica a contenedores, solo que para este ejemplo no creamos la imagen ni lanzamos al contenedor, estos pasos los realiza Docker Compose por nosotros así nos ahorramos el trabajo de crear tres imágenes y lanzarlas después manualmente, en su lugar creamos un archivo de definición *.yaml* y llamamos a Docker Compose el programa se encarga de crear las imágenes y lanzar los contendores a la vez.

CONVIRTIENDO EL DISEÑO MONOLÍTICO A UNO DE MICROSERVICIOS.

Nuestro proyecto de LAMP, se encuentra actualmente formado por los siguientes archivos:



Los cuales son auspiciados en de la carpeta de diseños hacia la carpeta de la página web ubicada en */var/www/html* para que el software del servidor Apache pueda "servirlos" archivos hacia el explorador web.

Para poder convertir nuestro diseño hacia una arquitectura de microservicios debemos separar funciones, en este caso el acceso y manejo de las operaciones de lectura o escritura hacia la base de datos deben ser desacopladas del código de las páginas web y relegarse a consultas hacia una api que se encargará de tratar las conexiones con los datos administrados por la base de datos, por ejemplo:

El archivo *index.php* de la versión monolítica para agregar un nuevo registro requiere llamar al archivo de inclusión backend *conexión.php* el cual prepara la conexión al motor de la base de datos, para después mediante un método *POST* enviar directamente una query que se ejecute por el motor de la base de datos y agregue un nuevo registro. como muestra el siguiente fragmento de código:

```php
<?php
// Incluimos la conexión que ya probamos
include 'conexion.php';

// Lógica para Insertar (esto se ejecuta cuando presionas el botón)
if (isset($_POST['agregar'])) {
    $nombre = mysqli_real_escape_string($conn, $_POST['nombre']);
    if (!empty($nombre)) {
        $sql_insert = "INSERT INTO personas (nombre, activo) VALUES ('$nombre', 1)";
        mysqli_query($conn, $sql_insert);
        // Recargar la página para limpiar el formulario y ver el nuevo nombre
        header("Location: index.php");
    }
}
?>
```

index.php de la versión monolítica (la primera que fue creada en este documento)

Ahora nuestro código de *index.php* no debe preocuparse de realizar la conexión al motor de la base de datos, en su defecto, lo que hará será enviar una petición de generar un nuevo registro con los datos a registrar en formato JSON, como se muestra en la siguiente imagen:

```php
<?php
// Ya NO incluimos conexion.php aquí.
// La comunicación es exclusivamente por HTTP.

$api_url = "http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php";


// Función para centralizar las peticiones cURL
function consumir_api($url, $metodo, $datos = null) {
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $metodo);
    curl_setopt($ch, CURLOPT_TIMEOUT, 10);
    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 10);
    curl_setopt($ch, CURLOPT_USERAGENT, 'Mozilla/5.0 (Compatible; PHP-API-Client)');

    if ($datos) {
        curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($datos));
        curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: application/json'));
    }

    $respuesta = curl_exec($ch);
    curl_close($ch);

    return json_decode($respuesta, true);
}

// Lógica para Insertar (POST)
if (isset($_POST['agregar'])) {
    $nombre = $_POST['nombre'];
    if (!empty($nombre)) {
        consumir_api($api_url, "POST", ["nombre" => $nombre]);
        header("Location: index.php");
    }
}
```

En lugar de usar un archivo de inclusión se utilizará una variable de ambiente:

```
$api_url =
"http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php";
```

Para indicar la dirección URL donde escucha nuestra api las peticiones de escritura o lectura en la base de datos (la cual ahora está escrita en IPv6).

Para el caso de ingresar un nuevo registro utilizando el método POST, bastará con llamar a la función *consumir_api()*, e incluir los parámetros correspondientes (URL, método, datos), dicha función configura los valores de las variables:

```
CURLOPT_RETURNTRANSFER, CURLOPT_CUSTOMREQUEST, CURLOPT_TIMEOUT,
CURLOPT_CONNECTTIMEOUT, CURLOPT_USERAGENT
```

Requeridos para el método CURL, en este caso como hay datos incluidos estos se convertirán a formato JSON mediante:

```
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($datos));
```

y a su vez enviados a la api con:

```
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
```

Por defecto se efectuará una petición de retorno de los datos actuales en nuestra base de datos mediante:

```
$respuesta = curl_exec($ch);
```

Para posteriormente terminar la petición CURL:

```
curl_close($ch);
```

Los datos retornados por la api se decodificaron como pares clave : valor típicos de un formato JSON que serán retornados al término de la función:

```
return json_decode($respuesta, true);
```

Y estos serán reclamados por la variable *$personas* cuando se realice una llamada con el método GET:

```
// Lógica para Leer (GET)
$personas = consumir_api($api_url, "GET");
```

Para que sean formateados y presentados en el formato HTML correspondiente. en la sección:

```
<tbody>
    <?php if (!empty($personas)): ?>
        <?php foreach ($personas as $p): ?>
        <tr>
            <td><?php echo $p['id']; ?></td>
            <td><?php echo $p['nombre']; ?></td>
            <td>
                <a href="editar.php?id=<?php echo $p['id']; ?>">Editar</a> |
                <a href="borrar.php?id=<?php echo $p['id']; ?>" class="btn-del">Elimin
            </td>
        </tr>
        <?php endforeach; ?>
    <?php else: ?>
        <tr><td colspan="3">No hay registros.</td></tr>
    <?php endif; ?>
</tbody>
```

Que anteriormente estaba manejada por código PHP que realizaba la query escrita en el código del archivo *index.php* de la arquitectura monolítica:

```
<tbody>
    <?php
    // Solo seleccionamos los registros que NO tengan borrado lógico
    $query = "SELECT id, nombre FROM personas WHERE activo = 1";
    $result = mysqli_query($conn, $query);

    while ($row = mysqli_fetch_assoc($result)) {
        echo "<tr>";
        echo "<td>" . $row['id'] . "</td>";
        echo "<td>" . $row['nombre'] . "</td>";
        echo "<td>
                <a href='editar.php?id=" . $row['id'] . "'>Editar</a> |
                <a href='borrar.php?id=" . $row['id'] . "'>Eliminar</a>
            </td>";
        echo "</tr>";
    }
    ?>
</tbody>
</table>
```

index.php de la versión monolítica (la primera que fue creada en este documento)

Note la diferencia entre los dos estilos de diseño, se elimina el uso de la instrucción *while* y no se requiere explícitamente escribir la query y por lo tanto no es requerido el archivo de inclusión *conexion.php* en la cabecera de *index.php* para la versión de microservicios.

Al final el nuevo archivo *index.php* en arquitectura de microservicios es:

```php
<?php
// Ya NO incluimos conexion.php aquí.
// La comunicación es exclusivamente por HTTP.

$api_url =
"http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php";


// Función para centralizar las peticiones cURL
function consumir_api($url, $metodo, $datos = null) {
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $metodo);
    curl_setopt($ch, CURLOPT_TIMEOUT, 10);
    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 10);
    curl_setopt($ch, CURLOPT_USERAGENT, 'Mozilla/5.0 (Compatible;
PHP-API-Client)');

    if ($datos) {
        curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($datos));
        curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
    }

    $respuesta = curl_exec($ch);
    curl_close($ch);

    return json_decode($respuesta, true);
}

// Lógica para Insertar (POST)
if (isset($_POST['agregar'])) {
    $nombre = $_POST['nombre'];
    if (!empty($nombre)) {
        consumir_api($api_url, "POST", ["nombre" => $nombre]);
        header("Location: index.php");
    }
}
```

```php
// Lógica para Leer (GET)
$personas = consumir_api($api_url, "GET");
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Lista de Personas - Frontend Desacoplado</title>
    <style>
        body { font-family: sans-serif; margin: 40px; }
        table { border-collapse: collapse; width: 400px; margin-top:
20px; }
        th, td { border: 1px solid #ccc; padding: 10px; }
        .btn-del { color: red; }
    </style>
</head>
<body>

    <h2>Agregar Nuevo Nombre (Vía API)</h2>
    <form method="POST">
        <input type="text" name="nombre" placeholder="Escribe un
nombre..." required>
        <button type="submit" name="agregar">Agregar</button>
    </form>

    <hr>

    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Acciones</th>
            </tr>
        </thead>
        <tbody>
            <?php if (!empty($personas)): ?>
                <?php foreach ($personas as $p): ?>
                <tr>
                    <td><?php echo $p['id']; ?></td>
                    <td><?php echo $p['nombre']; ?></td>
```

```
                    <td>
                        <a href="editar.php?id=<?php echo $p['id'];
?>">Editar</a> |
                        <a href="borrar.php?id=<?php echo $p['id']; ?>"
class="btn-del">Eliminar</a>
                    </td>
                </tr>
                <?php endforeach; ?>
            <?php else: ?>
                <tr><td colspan="3">No hay registros.</td></tr>
            <?php endif; ?>
        </tbody>
    </table>
</body>
</html>
```

así como el los archivos *editar.php:*<?php

```
// web/editar.php
$api_url =
"http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php";

// 1. Obtener los datos actuales para mostrar en el formulario
if (isset($_GET['id'])) {
    $id = $_GET['id'];

    $ch = curl_init($api_url . "?id=" . $id);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $respuesta = curl_exec($ch);
    $datos = json_decode($respuesta, true);
    curl_close($ch);

    // Si la API no devolvió nada, regresamos al index
    if (empty($datos)) {
        header("Location: index.php");
        exit;
    }
    $persona = $datos[0]; // La API devuelve un array, tomamos el
primer elemento
}

// 2. Procesar la actualización (cuando el usuario da clic en Guardar)
if (isset($_POST['actualizar'])) {
    $datos_update = [
        "id" => $_POST['id'],
```

```
        "nombre" => $_POST['nombre']
    ];

    $ch = curl_init($api_url);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT"); // Método PUT para
actualizar
    curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($datos_update));
    curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type:
application/json']);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    curl_exec($ch);
    curl_close($ch);

    header("Location: index.php");
    exit;
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Editar Registro</title>
</head>
<body>
    <h2>Editar Nombre</h2>
    <form method="POST" action="editar.php">
        <input type="hidden" name="id" value="<?php echo
$persona['id']; ?>">

        <input type="text" name="nombre" value="<?php echo
$persona['nombre']; ?>" required>
        <button type="submit" name="actualizar">Guardar
Cambios</button>
        <a href="index.php">Cancelar</a>
    </form>
</body>
</html>
```

Seguido de *borrar.php*:

```
<?php
// web/borrar.php
```

```php
if (isset($_GET['id'])) {
    $id = $_GET['id'];
    $api_url =
"http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php?id="
. $id;

    $ch = curl_init($api_url);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE"); // Especificamos
que es un borrado
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($ch);
    curl_close($ch);

    // Una vez procesado, volvemos a la lista
    header("Location: index.php");
} else {
    header("Location: index.php");
}
?>
```

Estos tres forman parte del frontEnd del proyecto, para la api tenemos *conexion.php*:

```php
<?php
// Configuración de las credenciales
$host = "localhost";
$user = "admin";      // El usuario que creamos
$pass = "1234"; // La contraseña que asignaste
$db   = "proyecto_lamp";       // El nombre de la base de datos

// Crear la conexión usando la extensión mysqli
$conn = mysqli_connect($host, $user, $pass, $db);

// Verificar si la conexión falló
if (!$conn) {
    // En producción es mejor no mostrar el error detallado, pero para
desarrollo es vital
    die("Fallo la conexión: " . mysqli_connect_error());
}

// Configurar el juego de caracteres a UTF-8 para evitar problemas con
acentos o eñes
mysqli_set_charset($conn, "utf8");

// Si llegamos aquí, la conexión fue exitosa
```

```
// echo "Conexión establecida correctamente"; // Descomenta para probar
?>
```

Que no es diferente del anterior diseño monolítico, y el nuevo archivo *personas.php*:

```
<?php
include 'conexion.php'; // Aquí sí es necesario
header("Content-Type: application/json");

$metodo = $_SERVER['REQUEST_METHOD'];

switch($metodo) {
    case 'GET':
        if (isset($_GET['id'])) {
            // Escenario A: Obtener una sola persona por ID
            $id = mysqli_real_escape_string($conn, $_GET['id']);
            $query = "SELECT id, nombre FROM personas WHERE id = $id
AND activo = 1 LIMIT 1";
        } else {
            // Escenario B: Obtener todos los activos (lo que ya
tenías)
            $query = "SELECT id, nombre FROM personas WHERE activo =
1";
        }

        $result = mysqli_query($conn, $query);
        $registros = mysqli_fetch_all($result, MYSQLI_ASSOC);
        echo json_encode($registros);
        break;

    case 'POST':
        // Leer el cuerpo de la petición (JSON)
        $json = file_get_contents('php://input');
        $datos = json_decode($json, true);

        if (isset($datos['nombre'])) {
            $nombre = mysqli_real_escape_string($conn,
$datos['nombre']);
            $sql = "INSERT INTO personas (nombre, activo) VALUES
('$nombre', 1)";
            if (mysqli_query($conn, $sql)) {
                echo json_encode(["status" => "ok"]);
            }
        }
        break;
```

```
    // Aquí podrías agregar CASE 'DELETE'para borrar logicamente y CASE
'PUT' para editar

    // Dentro del switch($metodo) en api/personas.php
    case 'DELETE':
        // Capturamos el ID que viene en la URL (?id=5)
        if (isset($_GET['id'])) {
            $id = mysqli_real_escape_string($conn, $_GET['id']);
            $sql = "UPDATE personas SET activo = 0 WHERE id = $id"; //
Borrado lógico

            if (mysqli_query($conn, $sql)) {
                echo json_encode(["status" => "deleted"]);
            } else {
                http_response_code(500);
                echo json_encode(["error" => "No se pudo eliminar"]);
            }
        }
        break;


        case 'PUT':
        $json = file_get_contents('php://input');
        $datos = json_decode($json, true);

        if (isset($datos['id']) && isset($datos['nombre'])) {
            $id = mysqli_real_escape_string($conn, $datos['id']);
            $nombre = mysqli_real_escape_string($conn,
$datos['nombre']);

            $sql = "UPDATE personas SET nombre = '$nombre' WHERE id =
$id";
            if (mysqli_query($conn, $sql)) {
                echo json_encode(["status" => "updated"]);
            } else {
                http_response_code(500);
                echo json_encode(["error" => "Error al actualizar"]);
            }
        }
        break;

}
```

```
?>
```

El cúal si necesita incluir al archivo *conexion.php* para poder conectar al motor de la base de datos, y mediante una sentencia de selección, contiene el código para manejar las peticiones POST GET DELETE y PUT, que forman nuestro CRUD básico.
Para ejecutar basta con colocar en la carpeta del servidor Apache */var/www/html* el directorio de nuestra aplicación tal como muestra la URL:

```
http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php
```

El árbol dentro de la carpeta */var/www/html* se ve así:

```
rodrigo@RodrigoTR:/var/www/html/proyecto_lamp_desacoplado$ tree
.
├── README.md
└── src
    ├── faseA
    │   ├── api
    │   │   ├── conexion.php
    │   │   └── personas.php
    │   └── web
    │       ├── borrar.php
    │       ├── editar.php
    │       └── index.php
    └── faseB
```

Nota: si no cuenta con el comando *tree* instale con la siguiente línea:

```
sudo apt update && sudo apt install tree
```

La conclusión que podemos sacar de aquí es que ahora es posible modificar la api *personas,php* para conectar a otra base de datos que podría ser NoSQL o inclusive un simple archivo de texto o una hoja de excel y no tendríamos que hacer ningún cambio a los archivos del frontend, cosa que antes no era posible.

CONTENERIZAR EL DISEÑO DE MICROSERVICIOS.

Los archivos frontEnd del proyecto requieren un cambio sutil en la URL:

```
$api_url = "http://api_web/personas.php";
```

Ahora no accedemos a nuestra aplicación por una dirección IP estática al servidor local, el acceso será mediante el servidor DNS interno que maneja Docker para los contenedores.
Por lo que los archivos quedan así:

index.php

```
<?php
// Ya NO incluimos conexion.php aquí.
// La comunicación es exclusivamente por HTTP.

//$api_url =
"http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php";
$api_url = "http://api_web/personas.php";

// Función para centralizar las peticiones cURL
function consumir_api($url, $metodo, $datos = null) {
```

```
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $metodo);
    curl_setopt($ch, CURLOPT_TIMEOUT, 10);
    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 10);
    curl_setopt($ch, CURLOPT_USERAGENT, 'Mozilla/5.0 (Compatible;
PHP-API-Client)');

    if ($datos) {
        curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($datos));
        curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
    }

    $respuesta = curl_exec($ch);
    curl_close($ch);

    return json_decode($respuesta, true);
}

// Lógica para Insertar (POST)
if (isset($_POST['agregar'])) {
    $nombre = $_POST['nombre'];
    if (!empty($nombre)) {
        consumir_api($api_url, "POST", ["nombre" => $nombre]);
        header("Location: index.php");
    }
}

// Lógica para Leer (GET)
$personas = consumir_api($api_url, "GET");
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Lista de Personas - Frontend Desacoplado</title>
    <style>
        body { font-family: sans-serif; margin: 40px; }
        table { border-collapse: collapse; width: 400px; margin-top:
20px; }
        th, td { border: 1px solid #ccc; padding: 10px; }
```

```
        .btn-del { color: red; }
    </style>
</head>
<body>

    <h2>Agregar Nuevo Nombre (Vía API)</h2>
    <form method="POST">
        <input type="text" name="nombre" placeholder="Escribe un
nombre..." required>
        <button type="submit" name="agregar">Agregar</button>
    </form>

    <hr>

    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Acciones</th>
            </tr>
        </thead>
        <tbody>
            <?php if (!empty($personas)): ?>
                <?php foreach ($personas as $p): ?>
                <tr>
                    <td><?php echo $p['id']; ?></td>
                    <td><?php echo $p['nombre']; ?></td>
                    <td>
                        <a href="editar.php?id=<?php echo $p['id'];
?>">Editar</a> |
                        <a href="borrar.php?id=<?php echo $p['id']; ?>"
class="btn-del">Eliminar</a>
                    </td>
                </tr>
                <?php endforeach; ?>
            <?php else: ?>
                <tr><td colspan="3">No hay registros.</td></tr>
            <?php endif; ?>
        </tbody>
    </table>
</body>
</html>
```

editar.php

```php
<?php
// web/editar.php
//$api_url =
"http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php";
$api_url = "http://api_web/personas.php";

// 1. Obtener los datos actuales para mostrar en el formulario
if (isset($_GET['id'])) {
    $id = $_GET['id'];

    $ch = curl_init($api_url . "?id=" . $id);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $respuesta = curl_exec($ch);
    $datos = json_decode($respuesta, true);
    curl_close($ch);

    // Si la API no devolvió nada, regresamos al index
    if (empty($datos)) {
        header("Location: index.php");
        exit;
    }
    $persona = $datos[0]; // La API devuelve un array, tomamos el
primer elemento
}

// 2. Procesar la actualización (cuando el usuario da clic en Guardar)
if (isset($_POST['actualizar'])) {
    $datos_update = [
        "id" => $_POST['id'],
        "nombre" => $_POST['nombre']
    ];

    $ch = curl_init($api_url);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT"); // Método PUT para
actualizar
    curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($datos_update));
    curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type:
application/json']);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    curl_exec($ch);
    curl_close($ch);
```

```
    header("Location: index.php");
    exit;
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Editar Registro</title>
</head>
<body>
    <h2>Editar Nombre</h2>
    <form method="POST" action="editar.php">
        <input type="hidden" name="id" value="<?php echo
$persona['id']; ?>">

        <input type="text" name="nombre" value="<?php echo
$persona['nombre']; ?>" required>
        <button type="submit" name="actualizar">Guardar
Cambios</button>
        <a href="index.php">Cancelar</a>
    </form>
</body>
</html>
```

borrar.php

```
<?php
// web/borrar.php

if (isset($_GET['id'])) {
    $id = $_GET['id'];
    //$api_url =
"http://[::1]/proyecto_lamp_desacoplado/src/faseA/api/personas.php?id="
. $id;
    $api_url = "http://api_web/personas.php?id=" . $id;

    $ch = curl_init($api_url);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE"); // Especificamos
que es un borrado
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($ch);
```

```
    curl_close($ch);

    // Una vez procesado, volvemos a la lista
    header("Location: index.php");
} else {
    header("Location: index.php");
}
?>
```

En el caso de los archivos de backEnd:

personas.php

```
<?php
include 'conexion.php'; // Aquí sí es necesario
header("Content-Type: application/json");

$metodo = $_SERVER['REQUEST_METHOD'];

switch($metodo) {
    case 'GET':
        if (isset($_GET['id'])) {
            // Escenario A: Obtener una sola persona por ID
            $id = mysqli_real_escape_string($conn, $_GET['id']);
            $query = "SELECT id, nombre FROM personas WHERE id = $id
AND activo = 1 LIMIT 1";
        } else {
            // Escenario B: Obtener todos los activos (lo que ya
tenías)
            $query = "SELECT id, nombre FROM personas WHERE activo =
1";
        }

        $result = mysqli_query($conn, $query);
        $registros = mysqli_fetch_all($result, MYSQLI_ASSOC);
        echo json_encode($registros);
        break;

    case 'POST':
        // Leer el cuerpo de la petición (JSON)
        $json = file_get_contents('php://input');
        $datos = json_decode($json, true);

        if (isset($datos['nombre'])) {
            $nombre = mysqli_real_escape_string($conn,
$datos['nombre']);
```

```
            $sql = "INSERT INTO personas (nombre, activo) VALUES
('$nombre', 1)";
            if (mysqli_query($conn, $sql)) {
                echo json_encode(["status" => "ok"]);
            }
        }
        break;

    // Aquí podrías agregar CASE 'DELETE'para borrar logicamente y CASE
'PUT' para editar

    // Dentro del switch($metodo) en api/personas.php
    case 'DELETE':
        // Capturamos el ID que viene en la URL (?id=5)
        if (isset($_GET['id'])) {
            $id = mysqli_real_escape_string($conn, $_GET['id']);
            $sql = "UPDATE personas SET activo = 0 WHERE id = $id"; //
Borrado lógico

            if (mysqli_query($conn, $sql)) {
                echo json_encode(["status" => "deleted"]);
            } else {
                http_response_code(500);
                echo json_encode(["error" => "No se pudo eliminar"]);
            }
        }
        break;


    // En personas.php - Caso PUT
    case 'PUT':
        $json = file_get_contents('php://input');
        $datos = json_decode($json, true);

        if (isset($datos['id']) && isset($datos['nombre'])) {
            $id = mysqli_real_escape_string($conn, $datos['id']);
            $nombre = mysqli_real_escape_string($conn,
$datos['nombre']);

            $sql = "UPDATE personas SET nombre = '$nombre' WHERE id =
$id";
            if (mysqli_query($conn, $sql)) {
                echo json_encode(["status" => "updated"]);
```

```
            } else {
                http_response_code(500); // Esto ayuda al cliente cURL
a detectar fallos
                echo json_encode(["error" => "Error al actualizar"]);
            }
        }
        break;

}
?>
```

Solo sufre algunos ligeros cambios para asegurar que los datos se manejan más seguros, para el caso del archivo *conexion.php* ahora las variables de ambiente que permiten el acceso a los datos en la base de datos, se obtienen mediante variables de ambiente proporcionadas por Docker (en vez de estar “hardcodeadas”.

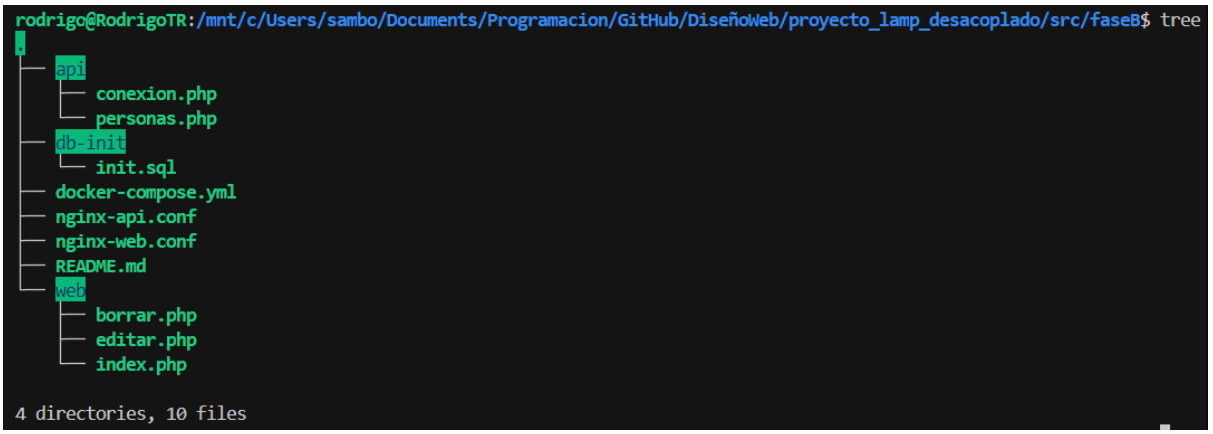
```
<?php
// Configuración de las credenciales
$host = "db"; //antes "localhost";//IP del servidor de base de datos
$user = getenv('MYSQL_USER');      // El usuario que creamos
$pass = getenv('MYSQL_PASSWORD'); // La contraseña que asignaste
$db   = getenv('MYSQL_DATABASE'); // El nombre de la base de datos

// Crear la conexión usando la extensión mysqli
$conn = mysqli_connect($host, $user, $pass, $db);

// Verificar si la conexión falló
if (!$conn) {
    // En producción es mejor no mostrar el error detallado, pero para
desarrollo es vital
    die("Fallo la conexión: " . mysqli_connect_error());
}

// Configurar el juego de caracteres a UTF-8 para evitar problemas con
acentos o eñes
mysqli_set_charset($conn, "utf8");

// Si llegamos aquí, la conexión fue exitosa
// echo "Conexión establecida correctamente"; // Descomenta para probar
?>
```

Como los contenedores son efímeros, requieren prestar atención a detalles especiales, como iniciar una base de datos vacía la primera vez que se ejecuta el volumen del contenedor, para esto creamos la carpeta db-init y dentro creamos el archivo *init.sql* el cual contendrá instrucciones adicionales para crear la base de datos la primera vez, el contenido del archivo se muestra a continuación:

```
CREATE TABLE IF NOT EXISTS personas (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    activo TINYINT(1) DEFAULT 1
);

INSERT INTO personas (nombre, activo) VALUES ('Usuario Inicial', 1),
('Prueba Docker', 1);
```

El árbol de directorios del proyecto ahora es así:



Dónde especial atención requiere el archivo *docker-compose.yml* el cual indica cómo deben configurarse los contenedores:

```
services:
  # --- BASE DE DATOS ---
  db:
    image: mariadb:latest
    container_name: db_faseB
    environment:
      MYSQL_ROOT_PASSWORD: Megamanzero
      MYSQL_DATABASE: proyecto_lamp
      MYSQL_USER: admin
      MYSQL_PASSWORD: "1234"
    volumes:
      - db_data_faseB:/var/lib/mysql
      - ./db-init:/docker-entrypoint-initdb.d

  # --- MICROSERVICIO API (Lógica y Datos) ---
  api_php:
    image: php:8.2-fpm
    container_name: api_php
    environment:
      MYSQL_USER: admin
      MYSQL_PASSWORD: "1234"
      MYSQL_DATABASE: proyecto_lamp
```

```
    volumes:
      - ./api:/var/www/html
    command: >
      sh -c "docker-php-ext-install mysqli && php-fpm"
    depends_on:
      - db

  api_web:
    image: nginx:alpine
    container_name: api_server
    ports:
      - "8081:80" # Puerto para la API
    volumes:
      - ./api:/var/www/html
      - ./nginx-api.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - api_php

  # --- MICROSERVICIO WEB (Frontend) ---
  web_php:
    image: php:8.2-fpm
    container_name: web_php
    environment:
      MYSQL_USER: admin
      MYSQL_PASSWORD: "1234"
      MYSQL_DATABASE: proyecto_lamp
    volumes:
      - ./web:/var/www/html
    command: >
      sh -c "docker-php-ext-install mysqli && php-fpm"

  web_srv:
    image: nginx:alpine
    container_name: web_server
    ports:
      - "8080:80" # Puerto para el usuario final
    volumes:
      - ./web:/var/www/html
      - ./nginx-web.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - web_php
      - api_web
```

```
volumes:
  db_data_faseB:
```

Observe que se han colocado los datos para las variables del entorno de contenedores Docker:

```
environment:
  MYSQL_ROOT_PASSWORD: Megamanzero
  MYSQL_DATABASE: proyecto_lamp
  MYSQL_USER: admin
  MYSQL_PASSWORD: "1234"
```

Y en esta ocasión se han definido los siguientes contenedores:
db servidor de la base de datos MariadB
api_php Servidor de la api que conecta a la base de datos
api_web Servidor intermedio para que nginx pueda reconocer código PHP
web_php Servidor intermedio para que nginx pueda reconocer código PHP
web_srv Servidor de las páginas del frontEnd
Como necesitamos dos servidores intermedios para que Nginx pueda entender PHP, requerimos crear dos archivos de configuración casi idénticos para cada contenedor ejecutando código PHP
nginx-api.conf

```
server {
    listen 80;
    index index.php index.html;
    server_name localhost;
    root /var/www/html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass api_php:9000; # Apunta al servicio 'php' definido
en docker-compose
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

Y el archivo *nginx-web.conf:*

```
server {
    listen 80;
```

```
    index index.php index.html;
    server_name localhost;
    root /var/www/html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass web_php:9000; # Apunta al servicio 'php' definido
en docker-compose
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

La única diferencia está en la línea 13 de cada archivo:

```
fastcgi_pass api_php:9000; # Apunta al servicio 'php' definido en
docker-compose
fastcgi_pass web_php:9000; # Apunta al servicio 'php' definido en
docker-compose
```

Donde se indica a cada servidor cual contenedor es del que depende como interprete de código PHP.

Para probar requerimos lanzar los contenedores desde el directorio donde se encuentra el archivo docker-compose.yml utilizando el comando:

docker compose up -d

Verificamos que los contenedores se encuentren en ejecución:

docker compose ps

Y probamos la conexión a la Api con la IP:

```
http://localhost:8081/personas.php`
```

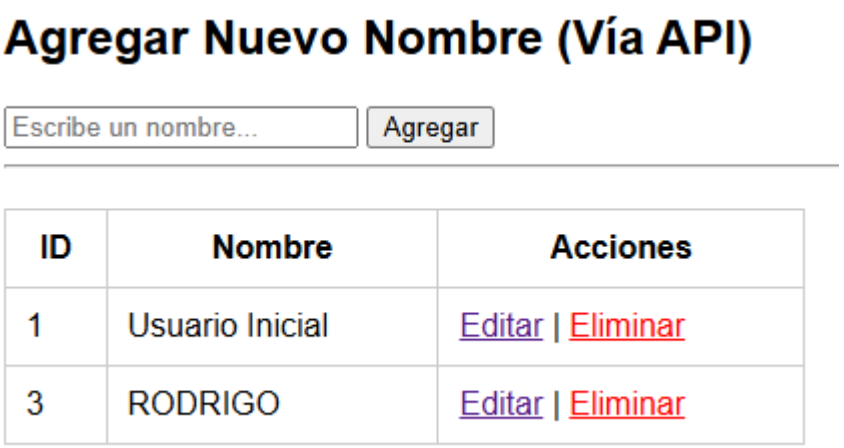
Veremos en el navegador:

[{"id":"1","nombre":"Usuario Inicial"},{"id":"3","nombre":"RODRIGO"}]

Y la aplicación web en la dirección:

```
http://localhost:8080
```

Se mostrará en el navegador:



Contenerizar un aplicación de microservicios respecto a una monolítica nos permite distribuir la carga, respecto a la aplicación en ejecución en un servidor Apache, ahora es posible crear contenedores adicionales de front end para distribuir las peticiones de los usuarios, y cuando el tráfico haya bajado eliminar los contenedores adicionales, esto es especialmente cuando nuestra pagina web se encuentra en la nube.

APÉNDICE

Instalar Apache Web Server en WSL Debian

Asegure de tener actualizado el sistema antes de instalar:

sudo apt update && sudo apt upgrade -y

Instale Apache usando el comando:

sudo apt install apache2 -y

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo apt install apache2 -y
Reading package lists... Done
Building dependency tree... Done
```

Apache no se inicia en WSL de manera automática al iniciar la sesión habrá que hacerlo cada vez que iniciemos sesión.

sudo service apache2 start

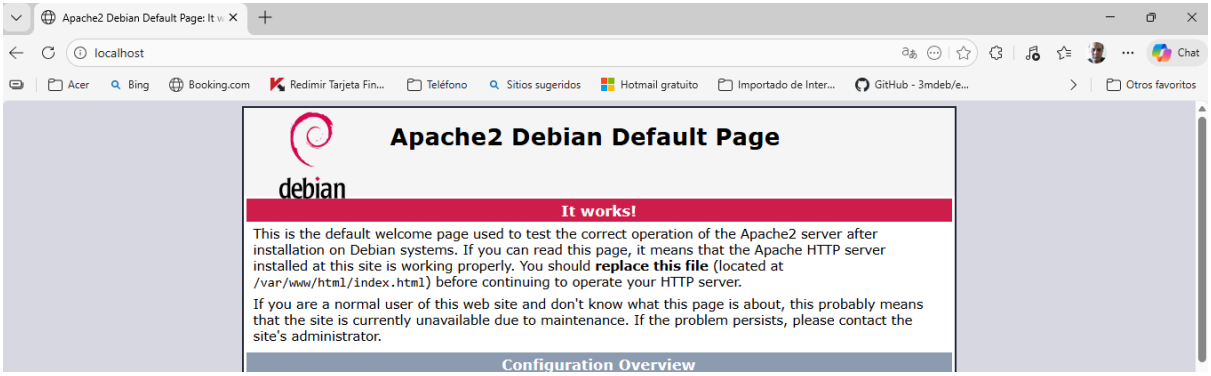
```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo service apache2 start
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$
```

Ahora puede verificar que la instalación se haya realizado correctamente:

sudo service apache2 status

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo service apache2 status
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset: enabled)
     Active: active (running) since Thu 2026-01-08 23:21:37 CST; 4min 32s ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 7591 (apache2)
      Tasks: 55 (limit: 4597)
     Memory: 21.1M
     CGroup: /system.slice/apache2.service
             ├─7591 /usr/sbin/apache2 -k start
             ├─7592 /usr/sbin/apache2 -k start
             └─7593 /usr/sbin/apache2 -k start

Jan 08 23:21:37 RodrigoTR systemd[1]: Starting apache2.service - The Apache HTTP Server...
Jan 08 23:21:37 RodrigoTR systemd[1]: Started apache2.service - The Apache HTTP Server.
```

En el Navegador en la dirección http://localhost, verifique que aparezca la página de bienvenida:



En la carpeta */var/www/html/*, es donde se guardaran los archivos de nuestra página web, ahí el punto de entrada será un archivo llamado *index.html* o *index.php* (nota si al intentar escribir archivos en la carpeta recibe un mensaje de error tendrá que modificar los permisos usando *chmod*).

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ ls /var/www/html
index.html
```

Algunos comandos útiles para administrar el servicio de Apache se listan a continuación:

Iniciar `sudo service apache2 start`

Detener `sudo service apache2 stop`

Reiniciar `sudo service apache2 restart`

Instalar MariaDB en WSL Debian

Asegure de tener actualizado su sistema WSL Debian y proceda a instalar el servicio de MariaDB:

sudo apt update

sudo apt install mariadb-server -y

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo apt update
sudo apt install mariadb-server -y
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://security.debian.org/debian-security bookworm-security InRelease
Hit:3 http://ftp.debian.org/debian bookworm-backports InRelease
Hit:4 http://deb.debian.org/debian bookworm-updates InRelease
Hit:5 https://download.docker.com/linux/debian bookworm InRelease
```

Deberá iniciar el servicio cada que entre en la sesión de WSL, y verificar que está ejecutando correctamente:

sudo service mariadb start

sudo service mariadb status

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo service mariadb start
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo service mariadb status
● mariadb.service - MariaDB 10.11.14 database server
     Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; preset: enabled)
     Active: active (running) since Thu 2026-01-08 23:38:49 CST; 2min 29s ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
   Main PID: 9390 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 9 (limit: 30342)
     Memory: 114.2M
     CGroup: /system.slice/mariadb.service
             └─9390 /usr/sbin/mariadbd

Jan 08 23:38:48 RodrigoTR mariadbd[9390]: 2026-01-08 23:38:48 0 [Note] Plugin 'FEEDBACK' is disabled.
Jan 08 23:38:48 RodrigoTR mariadbd[9390]: 2026-01-08 23:38:48 0 [Note] InnoDB: Buffer pool(s) load completed at 260108 23:38:48
Jan 08 23:38:49 RodrigoTR mariadbd[9390]: 2026-01-08 23:38:49 0 [Warning] You need to use --log-bin to make --expire-logs-days or --binlog-expire-logs-seconds work.
Jan 08 23:38:49 RodrigoTR mariadbd[9390]: 2026-01-08 23:38:49 0 [Note] Server socket created on IP: '127.0.0.1', port: '3306'.
Jan 08 23:38:49 RodrigoTR mariadbd[9390]: 2026-01-08 23:38:49 0 [Note] /usr/sbin/mariadbd: ready for connections.
Jan 08 23:38:49 RodrigoTR mariadbd[9390]: Version: '10.11.14-MariaDB-0+deb12u2'  socket: '/run/mysqld/mysqld.sock'  port: 3306  Debian 12
Jan 08 23:38:49 RodrigoTR systemd[1]: Started mariadb.service - MariaDB 10.11.14 database server.
Jan 08 23:38:49 RodrigoTR /etc/mysql/debian-start[9406]: Upgrading MySQL tables if necessary.
Jan 08 23:38:49 RodrigoTR /etc/mysql/debian-start[9417]: Checking for insecure root accounts.
Jan 08 23:38:49 RodrigoTR /etc/mysql/debian-start[9421]: Triggering myisam-recover for all MyISAM tables and aria-recover for all Aria tables
```

Proceda a configurar la seguridad (muy importante):

sudo mysql_secure_installation

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!
```

Se sugiere responder lo siguiente a las preguntas que se realicen durante la configuración:
Enter current password for root: Presiona *Enter* (está vacío por defecto). para este ejemplo usaremos 1234

```
In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
```

Switch to unix_socket authentication? Responde n (para usar contraseñas tradicionales en este ejercicio).

```
Switch to unix_socket authentication [Y/n] n
 ... skipping.
```

Change the root password? Responde y y define una contraseña que recuerdes bien.

```
Change the root password? [Y/n] y
New password:
Re-enter new password:
Warning: World-writable config file '/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/.my.cnf.9783' is ignored
Password updated successfully!
Reloading privilege tables..
Warning: World-writable config file '/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/.my.cnf.9783' is ignored
 ... Success!
```

Remove anonymous users? Responde y.

```
Remove anonymous users? [Y/n] y
Warning: World-writable config file '/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/.my.cnf.9783' is ignored
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.
```

Disallow root login remotely? Responde y.

```
Disallow root login remotely? [Y/n] y
Warning: World-writable config file '/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/.my.cnf.9783' is ignored
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.
```

Remove test database? Responde y.

```
Remove test database and access to it? [Y/n] y
 - Dropping test database...
Warning: World-writable config file '/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/.my.cnf.9783' is ignored
 ... Success!
 - Removing privileges on test database...
Warning: World-writable config file '/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/.my.cnf.9783' is ignored
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.
```

Reload privilege tables now? Responde y.

```
Reload privilege tables now? [Y/n] y
Warning: World-writable config file '/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb/.my.cnf.9783' is ignored
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

Verifique el acceso a la base de datos:
sudo mariadb -u root -p

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo mariadb -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 39
Server version: 10.11.14-MariaDB-0+deb12u2 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Para salir teclee *quit*.

Notas para tu documentación

En una **arquitectura monolítica**, MariaDB guarda físicamente los datos en el sistema de archivos de Debian. Es vital conocer estas rutas:

- Directorio de datos: `/var/lib/mysql` (Aquí es donde "viven" tus bases de datos).
- Archivo de configuración: `/etc/mysql/mariadb.conf.d/50-server.cnf`.
- Log de errores: `/var/log/mysql/error.log`.

En este entorno WSL, si borras Debian, **pierdes tus bases de datos.**

Instalación de PHP en WSL Debian

Actualice su sistema, para instalar utilice el siguiente comando:

sudo apt install php libapache2-mod-php php-mysql -y

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo apt install php libapache2-mod-php php-mysql -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libapache2-mod-php8.2 php-common php8.2 php8.2-cli php8.2-common php8.2-mysql php8.2-opcache php8.2-readline
```

Reinicie PHP para terminar la instalación:

sudo service apache2 restart

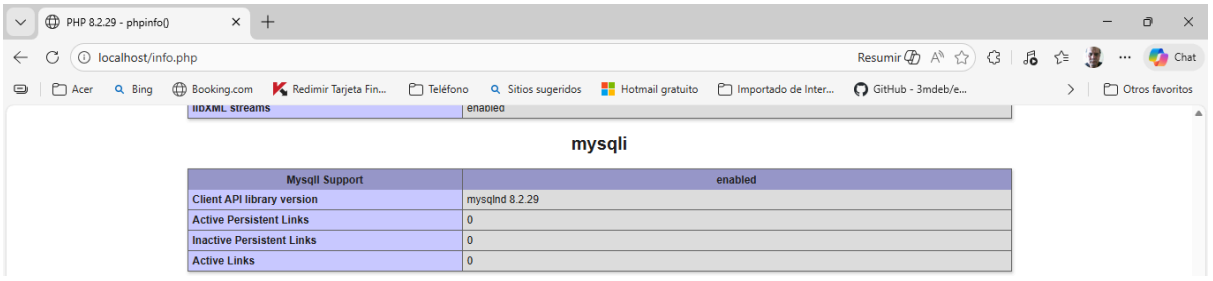
```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo service apache2 restart
```

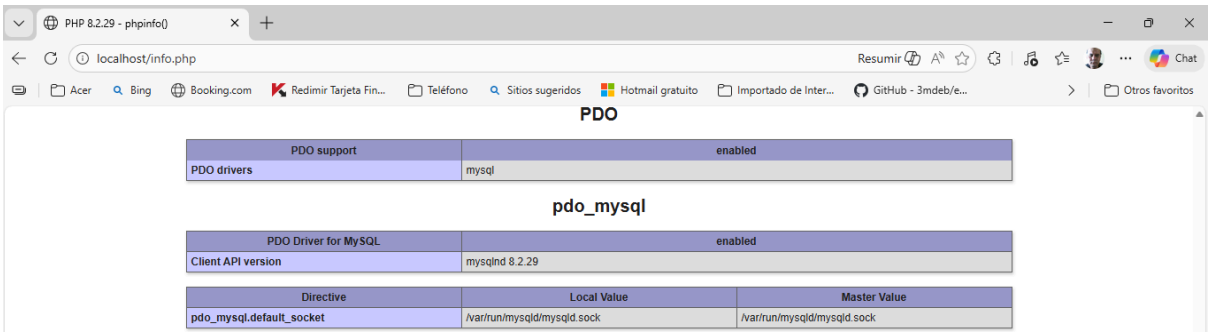
Para verificar la instalación se ha realizado correctamente, cree el siguiente archivo de prueba (el cual se enviará a la carpeta del servicios Web de Apache):

echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php
<?php phpinfo(); ?>
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ ls /var/www/html
index.html  info.php
```

Si tiene problemas para escribir en la carpeta utilice lo siguiente:
sudo chown -R $USER:$USER /var/www/html
Adicionalmente en el navegador verifique la instalación en la página web http://localhost/info.php deberá ver la información siguiente:





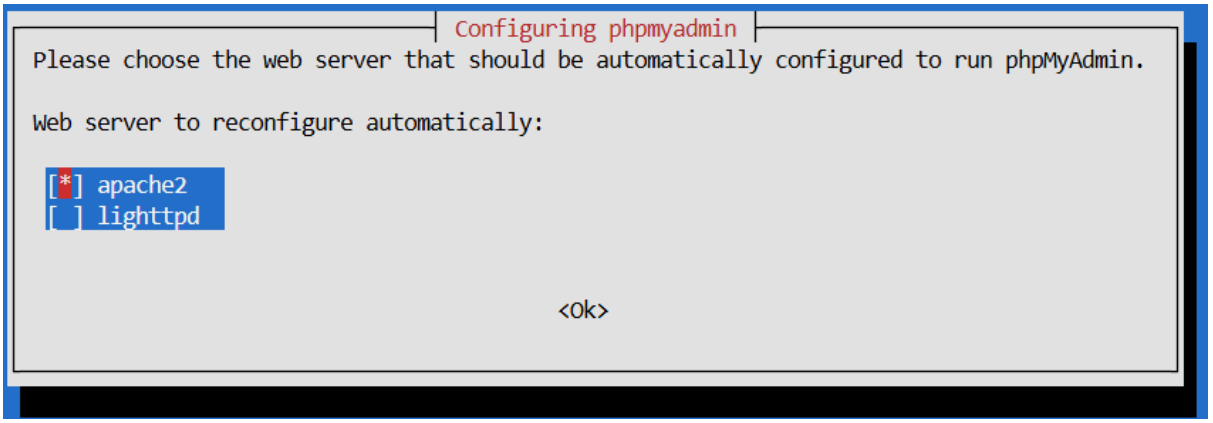
Indicando que PHP está instalado correctamente y además puede conectar con MariaDB.

Instalación de PHPMyAdmin en WSL Debian
Tener un gestor de base de datos con interfaz gráfica es de mucha utilidad antes de instalar asegure tener el sistema actualizado:
sudo apt update
sudo apt install phpmyadmin -y

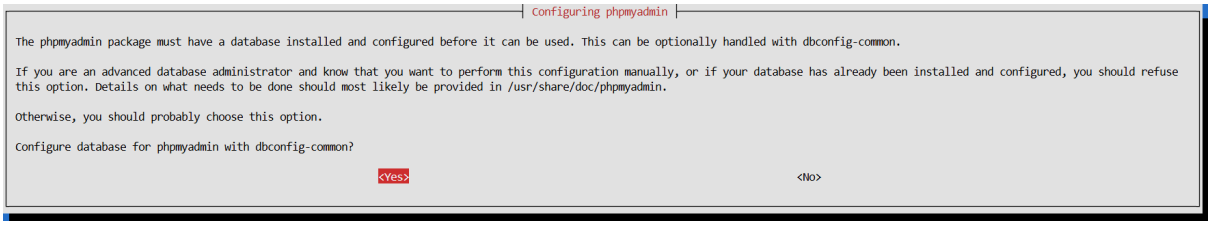
```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo apt install phpmyadmin -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
```

En las siguientes opciones de configuración se recomienda lo siguiente:

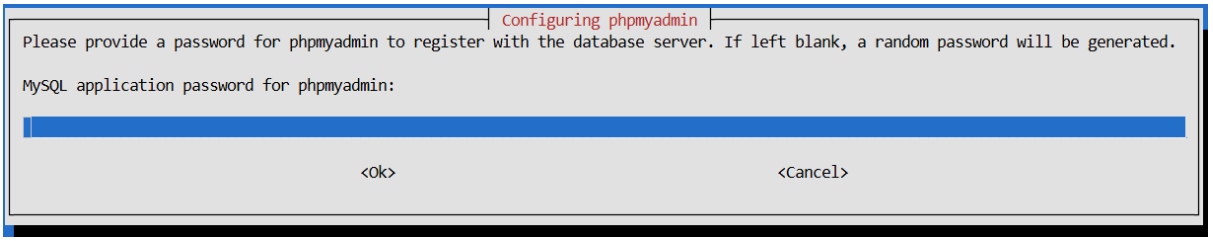
1. Web server to reconfigure automatically**:** Selecciona `apache2` (usa la tecla **Espacio** para marcar el asterisco `[*]` y luego **Enter**).



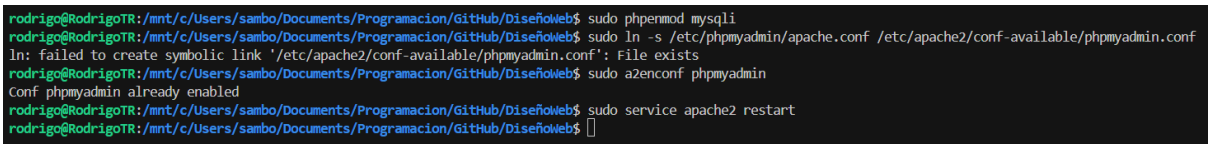
2. Configure database for phpmyadmin with dbconfig-common? Selecciona **Yes**.



3. MySQL application password for phpmyadmin: Define una contraseña para el uso interno de la herramienta (o déjala en blanco para que genere una aleatoria, ya que no la usarás frecuentemente).



Habilite el acceso de phpmyadmin con los siguientes comandos:
sudo phpenmod mysqli
sudo ln -s /etc/phpmyadmin/apache.conf /etc/apache2/conf-available/phpmyadmin.conf
sudo a2enconf phpmyadmin
sudo service apache2 restart



Configurar el acceso a MariaDB
sudo mariadb -u root -p
Y ejecute los siguientes comandos:
ALTER USER 'root'@'localhost' IDENTIFIED VIA mysql_native_password USING PASSWORD('tu_password');

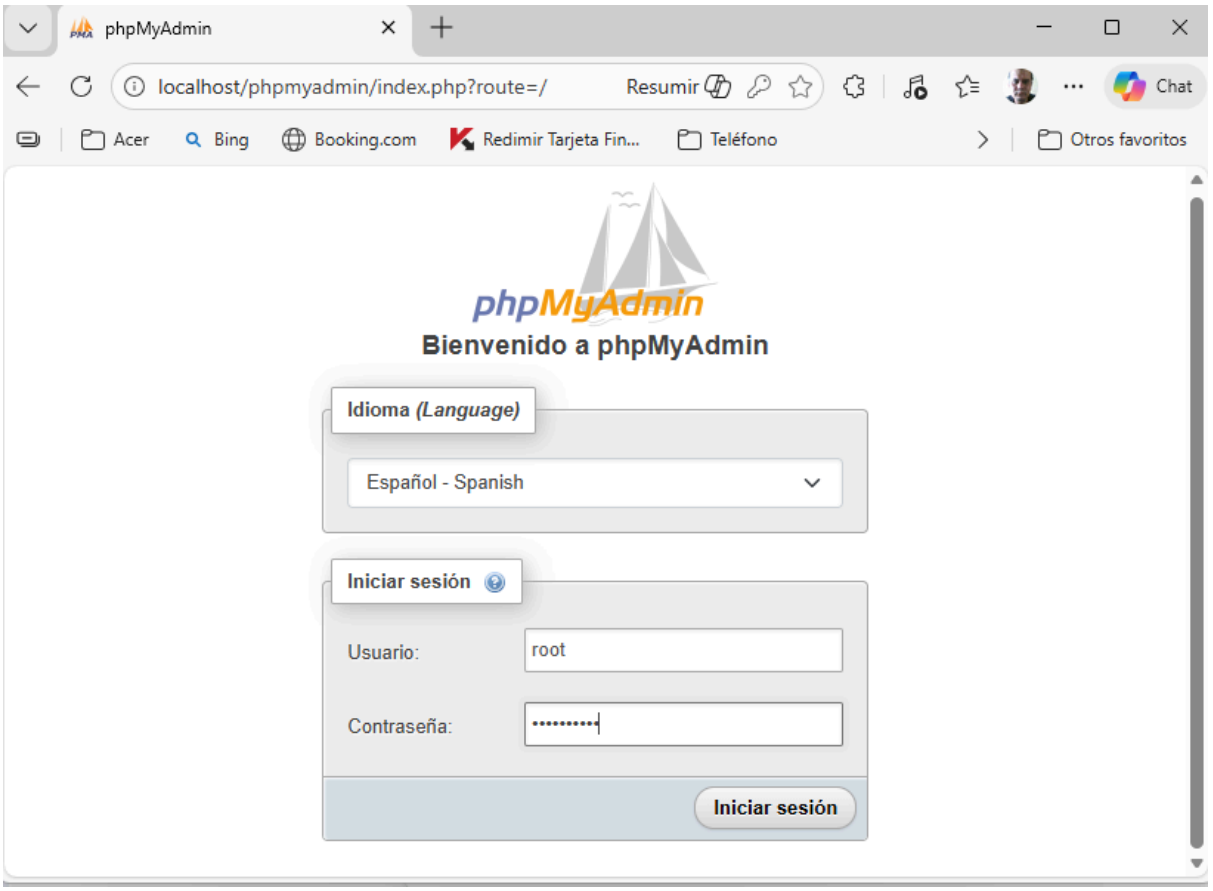
No olvide agregar su password que usarás para ingresar.
FLUSH PRIVILEGES;

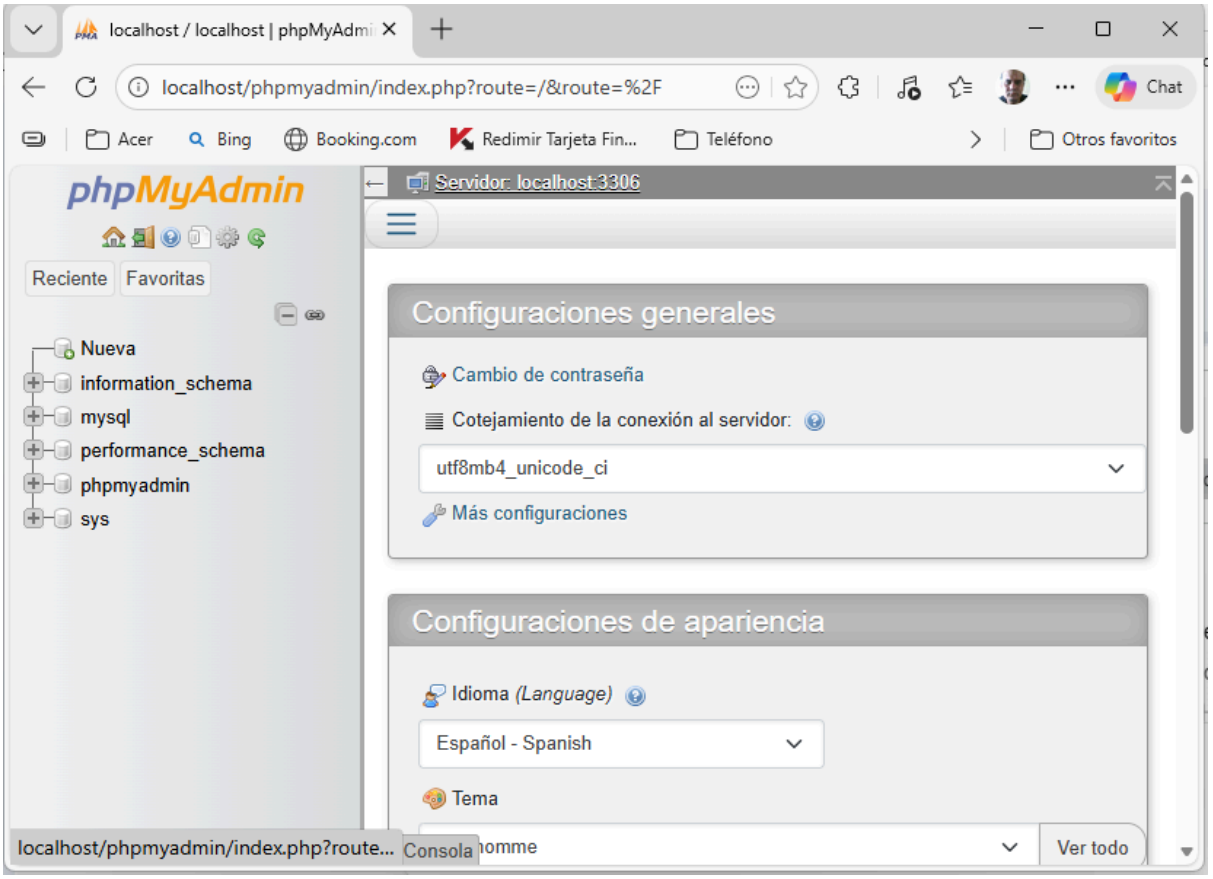
```
MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.049 sec)
```

EXIT;

```
MariaDB [(none)]> EXIT;
Bye
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$
```

Esto para habilitar el ingreso de phpmyadmin como root a MariaDB. Por último verifique en el navegador en la página http://localhost/phpmyadmin que se pueda ingresar con el usuario *root* y el password.





Listo, ya se ha configurado el gestor de base de datos para MariaDB.

Cuando se inicie la sesión de WSL no es requerido iniciar el servicio de PHPMyAdmin, ya que se inicia automáticamente con Apache, lo que si es requerido es asegurar que la base de datos está en ejecución, utilice:

```
sudo service apache2 start
sudo service mariadb start
```

Cada inicio de sesión de WSL para asegurar que todo esté en ejecución.

Desinstalar MariaDb de WSL Debian

Detener el servicio de MariaDB

sudo service mariadb stop

Desinstalar paquetes y dependencias con “purge” en vez de “remove”

sudo apt purge mariadb-server mariadb-client mariadb-common -y

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo service mariadb stop
[sudo] password for rodrigo:
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo apt purge mariadb-server mariadb-client mariadb-common -y
```

Aparecerá un mensaje que nos preguntará si deseamos mantener las bases de datos, en este caso elegimos que no ya que deseamos una instalación limpia.

Eliminamos las dependencias que ya no son necesarias:

sudo apt autoremove -y

sudo apt autoclean

```
rodrigo@RodrigoTR:/mnt/c/Users/sambo/Documents/Programacion/GitHub/DiseñoWeb$ sudo apt autoremove -y
sudo apt autoclean
Reading package lists... Done
Building dependency tree... Done
```

Limpieza manual de archivos residuales (importante limpiar):
sudo rm -rf /var/lib/mysql
sudo rm -rf /etc/mysql
Con esto habremos eliminado la instalación de MariaDb y estaremos listos para una instalación fresca y limpia.

Proyecto GitHub
RodrigoSamborms/Dise-oWeb