DOCUMENTACIÓN DEL PROYECTO CRUD - API REST EN PHP

PRESENTADO POR:

ALEJANDRO DE MENDOZA

EMPRESA AIRTECH

BOGOTA

24/02/2025

Tabla De Contenido

Introducción

En el contexto del desarrollo web, las API (Interfaz de Programación de Aplicaciones) son esenciales para permitir la comunicación entre diferentes aplicaciones y servicios. Este proyecto se centra en la creación de una API REST utilizando PHP y MySQL, con el objetivo de gestionar una base de datos de clientes mediante un conjunto de operaciones CRUD (Crear, Leer, Actualizar y Eliminar). El proyecto está diseñado para manejar peticiones HTTP mediante los métodos estándar de REST: POST, GET, PUT y DELETE. Estos métodos permiten interactuar con los registros de los clientes en la base de datos de manera eficiente y segura. Además, se implementó un proceso básico de autenticación para asegurar el acceso a los recursos de la API.

Herramientas Utilizadas

- Postman: Se utilizó Postman para realizar pruebas exhaustivas de la API RESTful que gestiona las operaciones CRUD para los clientes. Postman permitió verificar la correcta implementación de los endpoints y asegurarse de que las solicitudes GET, POST, PUT y DELETE fueran manejadas adecuadamente por el servidor.
- PHP: PHP fue utilizado para la creación de la API RESTful. Se implementaron las funcionalidades que permiten interactuar con la base de datos, realizando las operaciones de creación, lectura, actualización y eliminación de clientes.
- MySQL: MySQL se usó como el sistema de gestión de bases de datos para almacenar la información de los clientes. Las tablas de la base de datos permiten almacenar y recuperar los datos necesarios para las operaciones CRUD.
- HTML: HTML fue utilizado para estructurar la página web y proporcionar la interfaz de usuario (UI). Se diseñaron formularios de entrada de datos, botones de acción y áreas de visualización de los clientes, proporcionando una estructura básica de la aplicación.
- CSS: CSS se empleó para mejorar la apariencia visual de la interfaz de usuario. Se aplicaron estilos como colores, tamaños, márgenes, bordes redondeados y efectos de hover para los botones. Además, se utilizó CSS para asegurarse de que la página se viera atractiva y fácil de usar.
- JavaScript: JavaScript fue utilizado para manejar la lógica de interacción del frontend con la API. Se usaron funciones como fetch() para enviar solicitudes HTTP a la API, procesar los datos recibidos y actualizar dinámicamente la lista de clientes en la página sin necesidad de recargarla.

Métodos HTTP

- POST: Se utiliza para crear nuevos registros de clientes en la base de datos.
- GET: Permite recuperar los datos de los clientes almacenados en la base de datos.
- PUT: Se utiliza para actualizar los registros de los clientes existentes en la base de datos.
- DELETE: Permite eliminar los registros de clientes de la base de datos.

- HTML y CSS: La interfaz de usuario de la aplicación se desarrolló utilizando HTML para estructurar la página web y CSS para aplicar estilos básicos, asegurando que la aplicación sea visualmente atractiva y fácil de usar.

Objetivos Del Proyecto:

Este desarrollo tiene como objetivo demostrar una implementación básica de una API RESTful utilizando PHP y MySQL, aplicando operaciones CRUD sobre una base de datos de clientes. La autenticación básica se ha implementado para asegurar el acceso a los recursos de la API y garantizar la seguridad de las operaciones realizadas.

Además, se ha desarrollado una interfaz web básica en HTML y CSS, que permite interactuar con la API de manera fácil e intuitiva, mostrando los datos de los clientes y permitiendo la creación, actualización y eliminación de registros.

Trámite De La Actividad Por Realizar

1. Crear un ejercicio de API REST en PHP que realice operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre un listado de clientes.
2. Implementar un sistema de autenticación básica utilizando PHP y MySQL sin frameworks ni librerías.
3. El sistema debe usar los métodos HTTP POST, PUT, DELETE y GET.
4. Montar una interfaz básica de usuario utilizando HTML y aplicarle estilos con CSS.
5. Documentar todo el proceso.

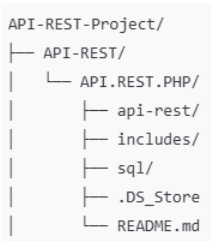
Endpoints De La API REST

Los siguientes endpoints están disponibles para realizar las operaciones CRUD:

1. POST /create_product - Crea un nuevo cliente.
2. GET /get_all_products - Obtiene la lista de todos los clientes.
3. PUT /update_product - Actualiza los detalles de un cliente.
4. DELETE /delete_product - Elimina un cliente.

Estructura De Carpetas

El proyecto sigue una estructura de carpetas que facilita su organización:



- **api-rest/**: Contiene los archivos PHP que manejan la lógica de las operaciones CRUD.
- **includes/**: Contiene archivos de configuración, como la conexión a la base de datos.
- **sql/**: Carpeta donde se guardan los scripts SQL para la base de datos.

Desarrollo Del Proyecto

Este proyecto tiene como objetivo el desarrollo de una aplicación web de tipo CRUD (Crear, Leer, Actualizar, Eliminar) para la gestión de clientes. La plataforma permitirá a los usuarios interactuar con una base de datos de clientes, facilitando la adición, edición, visualización y eliminación de registros a través de una interfaz amigable.

El sistema estará respaldado por una API RESTful que permitirá la interacción entre el frontend y el backend. A través de este proyecto, se busca implementar buenas prácticas en el desarrollo de aplicaciones web, como la separación entre la lógica de negocio y la presentación, y el manejo adecuado de solicitudes HTTP (GET, POST, PUT, DELETE).

El desarrollo se lleva a cabo utilizando tecnologías modernas como HTML, CSS, y JavaScript en el frontend, y PHP para la creación de la API REST. Se utilizarán conceptos fundamentales de desarrollo web, como la validación de formularios, el manejo de datos a través de solicitudes HTTP y la integración de bases de datos para almacenar la información de los clientes.

Este proyecto se plantea como una solución simple y eficiente para la gestión de información, con el fin de brindar una herramienta útil para organizaciones que requieren un sistema ágil para administrar datos de contacto y detalles importantes de sus clientes. A continuación, inicio su desarrollo.

Configuración del entorno

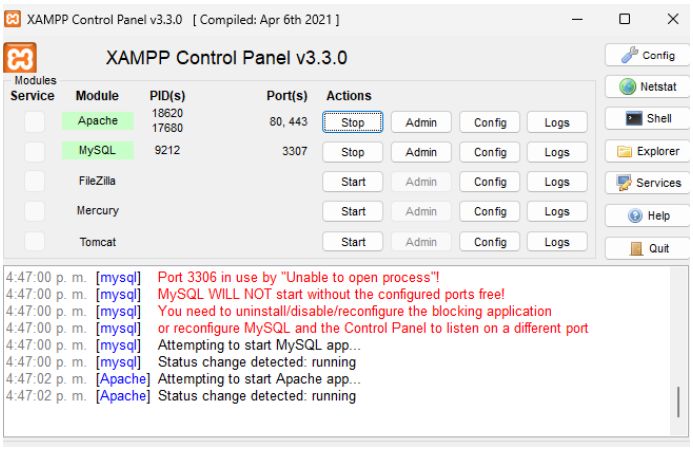
Requisitos previos:

1. **Instalar XAMPP**: Se debe tener instalado XAMPP. XAMPP incluye Apache (para servir archivos PHP) y MySQL (para la base de datos).

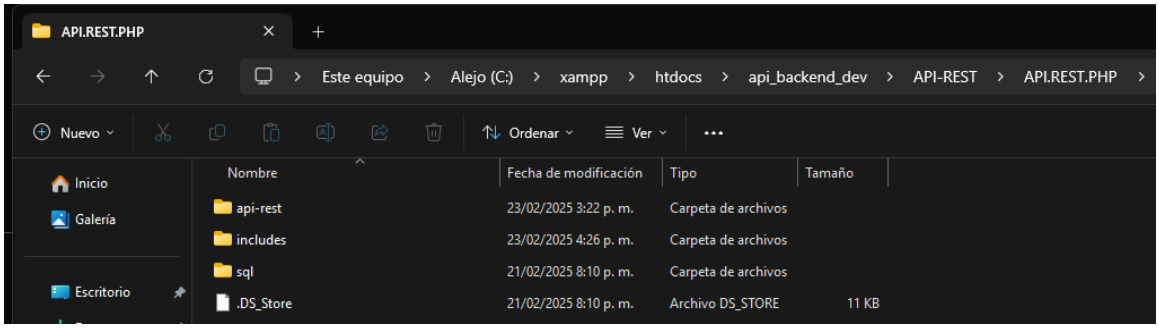
2. **Iniciar Apache y MySQL**:

    - Abrir el **panel de control de XAMPP**.

    - Iniciar **Apache** y **MySQL**.



3. **Subir los archivos del proyecto** a la carpeta htdocs en XAMPP, en mi caso la ruta es: C:\xampp\htdocs\api_backend_dev\API-REST.



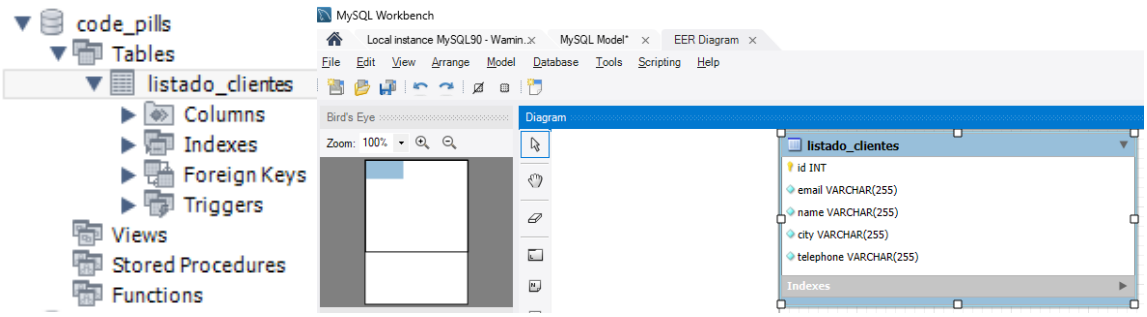
4. **Acceder al proyecto**: En el navegador, ingresar a http://localhost/ seguido de la ruta de ubicación donde se guardó el proyecto (por ejemplo, http://localhost/api_backend_dev/api-rest/API.REST.PHP/api-rest/index.html).

5. Ahora que el entorno esta configurado se procede con el desarrollo.

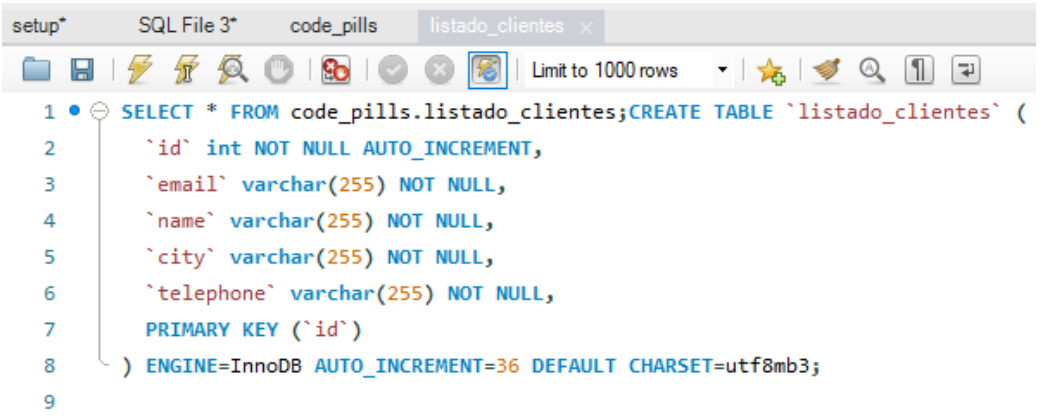
Base de datos:

Para la realización de este proyecto como primera medida se procedió con la Configuración de la Base de Datos en MySQL Workbench que es una herramienta gráfica que nos permite interactuar con nuestras bases de datos MySQL, facilitando la creación y gestión de tablas, así como la ejecución de consultas SQL.

Entonces lo primero con lo que procedí fue con la instalación de la base de datos remitida en GitHub de nombre “setup.sql” en la herramienta MySQL Workbench. Este proceso instalo la base de datos code_pills con una única tabla de nombre listado_clientes cuyas imágenes son las siguientes.



Luego se ejecutó el script SQL para crear la tabla de clientes.



Este código define una tabla con las siguientes características:

- id: Un campo numérico único para identificar a cada cliente.
- email: Un campo de texto que almacena la dirección de correo electrónico del cliente.
- name: El nombre del cliente.
- city: La ciudad en la que reside el cliente.
- telephone: El número de teléfono del cliente.

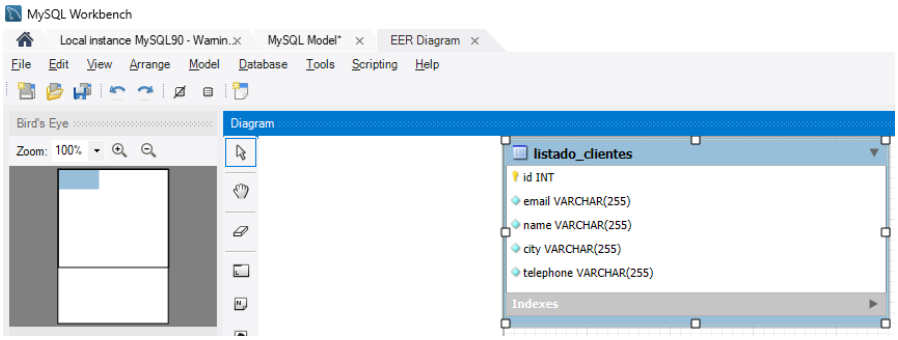
Después de ejecutar el script de creación de la tabla, se verificó su existencia mediante el siguiente comando:

SHOW TABLES;

Para asegurarme que la tabla fue creada correctamente, use el siguiente comando para describir su estructura:

DESCRIBE listado_clientes;

Adicional procedí a verificar su estructura en el mapa de tablas EER:



Y como efectivamente todo estaba correcto procedí entonces con el desarrollo en PHP.

Desarrollo en PHP (Sublime)

Para este proceso primero determine la estructura del proyecto y en este caso en este proyecto, se creó un CRUD (Crear, Leer, Actualizar, Eliminar) para gestionar la información de los clientes. Los archivos principales de este proyecto se encuentran en el directorio API.REST.PHP, y las funcionalidades están implementadas en el archivo create_product.php, que maneja las solicitudes de tipo POST, PUT, DELETE y GET.

database.class.php

Luego procedí con el desarrollo de conexión con la Base de Datos (database.class.php) para interactuar con la base de datos, y para esto se adaptó el archivo de conexión database.class.php que maneja las operaciones de conexión y desconexión de la base de datos con el siguiente código:

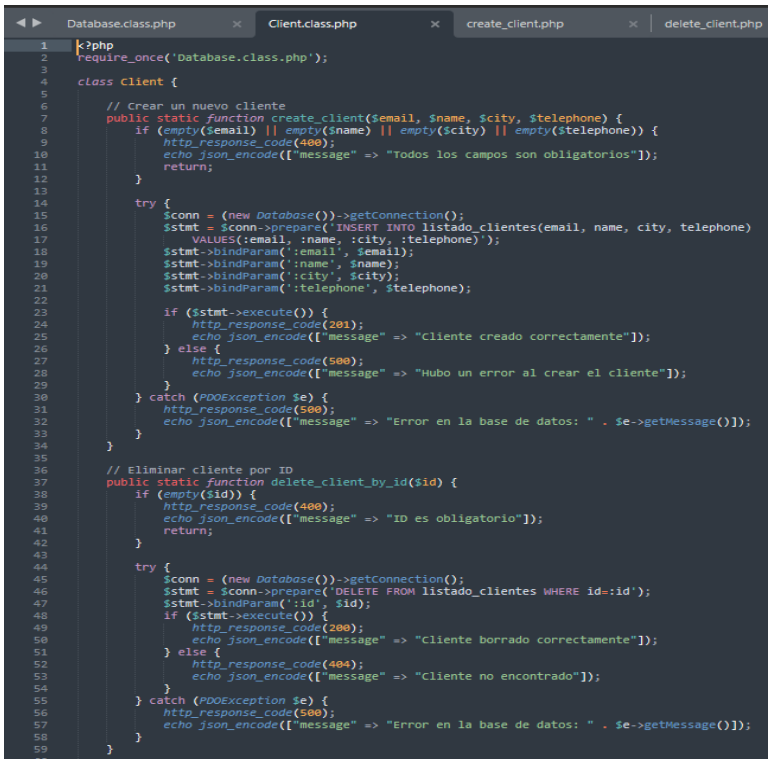
test_connection.php | Database.class.php | Client.class.php | create_client.php | delete_c

```php
<?php
    class Database {
        private $host = 'localhost:3306';
        private $user = 'root';
        private $password = '';
        private $database = 'code_pills'; // El nombre de la base de datos.

        public function getConnection() {
            // Se establece el charset=utf8 para evitar problemas con caracteres especiales.
            $hostDB = "mysql:host=".$this->host.";dbname=".$this->database.";charset=utf8";

            try {
                // Establezco la conexión con los parámetros proporcionados.
                $connection = new PDO($hostDB, $this->user, $this->password);
                // Establezco el modo de error para obtener excepciones en caso de error.
                $connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
                // Configuro el modo de búsqueda para que los resultados sean un array asociativo por defecto.
                $connection->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
                return $connection;
            } catch (PDOException $e) {
                // En caso de error, se muestra un mensaje de error detallado.
                die("ERROR: No se pudo establecer la conexión con la base de datos. Detalles: " . $e->getMessage());
            }
        }
    }
?>
```

Se aclara que este archivo utiliza PDO (PHP Data Objects) para conectar a la base de datos de manera segura.

client.class.php

Se procedió luego con el desarrollo del archivo client.class.php para manejar los métodos POST, GET, PUT, DELETE y para realizar las acciones correspondientes sobre la tabla listado_clientes, a continuación, el código respectivo:

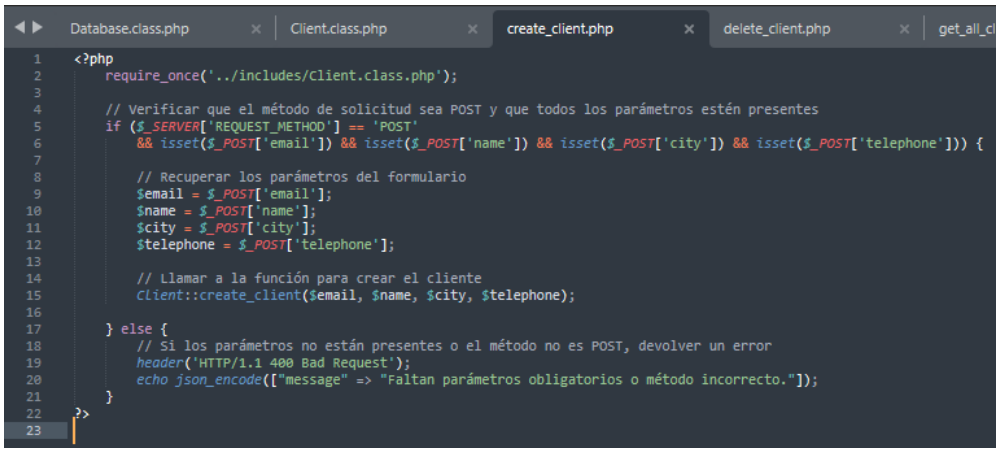



```php
<?php
require_once('Database.class.php');

class Client {

    // Crear un nuevo cliente
    public static function create_client($email, $name, $city, $telephone) {
        if (empty($email) || empty($name) || empty($city) || empty($telephone)) {
            http_response_code(400);
            echo json_encode(["message" => "Todos los campos son obligatorios"]);
            return;
        }

        try {
            $conn = (new Database())->getConnection();
            $stmt = $conn->prepare('INSERT INTO listado_clientes(email, name, city, telephone)
                VALUES(:email, :name, :city, :telephone)');
            $stmt->bindParam(':email', $email);
            $stmt->bindParam(':name', $name);
            $stmt->bindParam(':city', $city);
            $stmt->bindParam(':telephone', $telephone);

            if ($stmt->execute()) {
                http_response_code(201);
                echo json_encode(["message" => "Cliente creado correctamente"]);
            } else {
                http_response_code(500);
                echo json_encode(["message" => "Hubo un error al crear el cliente"]);
            }
        } catch (PDOException $e) {
            http_response_code(500);
            echo json_encode(["message" => "Error en la base de datos: " . $e->getMessage()]);
        }
    }

    // Eliminar cliente por ID
    public static function delete_client_by_id($id) {
        if (empty($id)) {
            http_response_code(400);
            echo json_encode(["message" => "ID es obligatorio"]);
            return;
        }

        try {
            $conn = (new Database())->getConnection();
            $stmt = $conn->prepare('DELETE FROM listado_clientes WHERE id=:id');
            $stmt->bindParam(':id', $id);
            if ($stmt->execute()) {
                http_response_code(200);
                echo json_encode(["message" => "Cliente borrado correctamente"]);
            } else {
                http_response_code(404);
                echo json_encode(["message" => "Cliente no encontrado"]);
            }
        } catch (PDOException $e) {
            http_response_code(500);
            echo json_encode(["message" => "Error en la base de datos: " . $e->getMessage()]);
        }
    }
```

```php
    // Obtener todos los clientes
    public static function get_all_clients() {
        try {
            $conn = (new Database())->getConnection();
            $stmt = $conn->prepare('SELECT * FROM listado_clientes');
            if ($stmt->execute()) {
                http_response_code(200);
                echo json_encode($stmt->fetchAll(PDO::FETCH_ASSOC));
            } else {
                http_response_code(404);
                echo json_encode(["message" => "No se han encontrado clientes"]);
            }
        } catch (PDOException $e) {
            http_response_code(500);
            echo json_encode(["message" => "Error en la base de datos: " . $e->getMessage()]);
        }
    }

    // Actualizar cliente por ID
    public static function update_client($id, $email, $name, $city, $telephone) {
        if (empty($id) || empty($email) || empty($name) || empty($city) || empty($telephone)) {
            http_response_code(400);
            echo json_encode(["error" => true, "message" => "Todos los campos son obligatorios"]);
            exit();  // 🔴 Detiene la ejecución después de la respuesta
        }

        try {
            $conn = (new Database())->getConnection();
            $stmt = $conn->prepare('UPDATE listado_clientes SET email=:email, name=:name, city=:city, telephone=:telephone WHERE id=:id');
            $stmt->bindParam(':email', $email);
            $stmt->bindParam(':name', $name);
            $stmt->bindParam(':city', $city);
            $stmt->bindParam(':telephone', $telephone);
            $stmt->bindParam(':id', $id);

            $stmt->execute();

            if ($stmt->rowCount() > 0) {
                http_response_code(200);
                echo json_encode(["success" => true, "message" => "Cliente actualizado correctamente"]);
            } else {
                http_response_code(404);
                echo json_encode(["error" => true, "message" => "No se pudo actualizar el cliente. Verifique que el ID existe."]);
            }

            exit(); // 🔴 Asegura que solo se envía una respuesta y detiene la ejecución

        } catch (PDOException $e) {
            http_response_code(500);
            echo json_encode(["error" => true, "message" => "Error en la base de datos: " . $e->getMessage()]);
            exit(); // 🔴 Detiene la ejecución tras la respuesta
        }
    }


}
?>
```

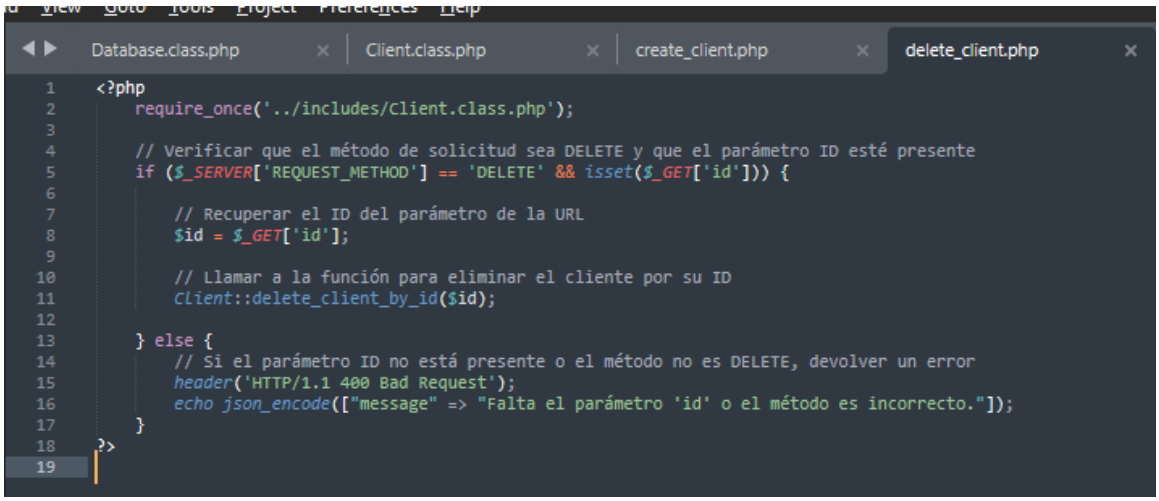
créate_client.php

Se procedió entonces con el desarrollo del archivo créate_client.php, este código PHP está diseñado para manejar una solicitud POST que reciba datos relacionados con un cliente, como email, name, city y telephone, a continuación, el código:



delete_client.php

Se procedió entonces con el desarrollo del archivo delete_client.php, está diseñado para manejar una solicitud DELETE para eliminar un cliente de la base de datos mediante su ID. A continuación, el código:



get_all_client.php

Se procedió entonces con el desarrollo del archivo get_all_client.php, este código PHP está diseñado para manejar una solicitud GET que se utiliza para obtener todos los clientes de la base de datos o de algún otro origen de datos. A continuación, el código:

```php
<?php
    require_once('../includes/Client.class.php');

    // Verificar que el método de solicitud sea GET
    if ($_SERVER['REQUEST_METHOD'] == 'GET') {

        // Llamar a la función para obtener todos los clientes
        Client::get_all_clients();

    } else {
        // Si el método no es GET, devolver un error
        header('HTTP/1.1 405 Method Not Allowed');
        echo json_encode(["message" => "Método no permitido. Solo se permite GET."]);
    }
?>
```

update_client.php

Se procedió entonces con el desarrollo del archivo update_client.php, donde este código PHP maneja una solicitud HTTP de tipo PUT, que es utilizada para actualizar un recurso existente en el servidor en este caso el de un cliente. A continuación, el código:

```php
<?php
require_once('../includes/Client.class.php');

if ($_SERVER['REQUEST_METHOD'] == 'PUT') {
    parse_str(file_get_contents("php://input"), $_PUT); // Capturar datos PUT

    if (isset($_PUT['email']) && isset($_PUT['name']) && isset($_PUT['city']) && isset($_PUT['telephone'])) {
        $id = $_GET['id'] ?? null;
        $email = $_PUT['email'];
        $name = $_PUT['name'];
        $city = $_PUT['city'];
        $telephone = $_PUT['telephone'];

        if ($id) {
            Client::update_client($id, $email, $name, $city, $telephone);
            echo json_encode(["message" => "Cliente actualizado correctamente"]);
        } else {
            echo json_encode(["error" => true, "message" => "ID del cliente no proporcionado."]);
        }
    } else {
        echo json_encode(["error" => true, "message" => "Faltan parámetros requeridos."]);
    }
} else {
    echo json_encode(["error" => true, "message" => "Método no permitido."]);
}
```

debug_log.txt

Se desarrollo el archivo debug_log.txt para registrar mensajes de depuración (debugging) durante el desarrollo o el funcionamiento de una aplicación. Estos registros pueden ser útiles para identificar errores, realizar un seguimiento del comportamiento de la aplicación y solucionar problemas.

```
{"debug_received_data":{"id":"7","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
{"debug_received_data":{"id":"8","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"9","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"9","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"9","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"9","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"7","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"7","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"9","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"11","email":"nuevo@email.com","name":"Julian Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Resultado de update_client: null
{"debug_received_data":{"id":"11","email":"nuevo@email.com","name":"andres Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Ejecutando consulta UPDATE para ID: 11
✅ Cliente con ID 11 actualizado correctamente.
Resultado de update_client: null
{"debug_received_data":null}
{"debug_received_data":{"id":"15","email":"nuevo@email.com","name":"Andras Camilo Barrios","city":"Bogot\u00e1","telephone":"3001234567"}}
Ejecutando consulta UPDATE para ID: 15
✅ Cliente con ID 15 actualizado correctamente.
```

test_connection.php

Se desarrollo el archivo test_connection.php, que establece una conexión a la base de datos MySQL utilizando la extensión mysqli, donde:

- $host: Especifica la dirección del servidor donde se encuentra la base de datos. En este caso, está configurado como 'localhost', lo que significa que la base de datos se encuentra en el mismo servidor que el script PHP.
- $usuario: Es el nombre de usuario utilizado para autenticar la conexión a la base de datos. Aquí se usa el usuario 'root', que es el predeterminado en muchas instalaciones de MySQL (aunque se recomienda cambiarlo por razones de seguridad).
- $contraseña: Es la contraseña asociada al usuario. En este caso, está vacía (''), lo que significa que no se está utilizando una contraseña. Esto también es común en instalaciones locales o de prueba de MySQL.
- $base_de_datos: Es el nombre de la base de datos a la que se desea conectar. Aquí se usa 'code_pills', pero es importante asegurarse de que esa base de datos realmente exista en el servidor de MySQL.
- new mysqli(): Aquí se crea una nueva instancia de la clase mysqli, que representa una conexión a una base de datos MySQL. Los parámetros que se pasan son:
    - $host: La dirección del servidor de la base de datos (en este caso, 'localhost').
    - $usuario: El nombre de usuario para la base de datos (aquí 'root').
    - $contraseña: La contraseña del usuario (en este caso, vacía).
    - $base_de_datos: El nombre de la base de datos a la que se conecta (en este caso 'code_pills').
    - 3306: El puerto que se usa para la conexión MySQL. Este es el puerto predeterminado para MySQL.

A continuación, el código:

index.html

Por último, se desarrolló el archivo index.html, Este código es una interfaz de usuario para un sistema CRUD (Crear, Leer, Actualizar, Eliminar) de clientes. Está construido con HTML, CSS y JavaScript. Aquí hay una explicación rápida de su funcionamiento:

- HTML: Define la estructura de la página con un formulario para ingresar información del cliente (correo electrónico, nombre, ciudad y teléfono) y botones para guardar o editar la información. Hay un botón para mostrar u ocultar la lista de clientes guardados.
- CSS: Estiliza el formulario y la lista de clientes, añadiendo un fondo de imagen, un diseño centrado, y efectos de estilo para los botones y la lista de clientes.
- JavaScript:
    - Funciones CRUD:
        - fetchClients(): Obtiene y muestra la lista de clientes desde el servidor utilizando la API.
        - Mostrar/Ocultar lista: El botón "Mostrar Clientes" alterna la visibilidad de la lista.
        - Formulario: Al enviar el formulario, dependiendo de si se está creando o actualizando un cliente, se envía una solicitud POST o PUT al servidor.
        - Eliminar y Editar: Los botones de eliminar y editar permiten modificar o borrar un cliente de la base de datos, enviando solicitudes DELETE o PUT.
- API: Se realiza una comunicación con un backend mediante fetch, enviando datos como parámetros en las solicitudes para realizar las operaciones CRUD en una base de datos. En resumen, este código permite interactuar con un sistema backend para manejar clientes a través de una interfaz web sencilla.

Operación Create (POST) Postman:

En este caso utilicé la herramienta Postman que es muy útil para hacer pruebas de APIs, enviar solicitudes HTTP y revisar las respuestas. Para esto los pasos fueron:

1. Iniciar Postman y configurar las solicitudes

Primero, abro Postman, crear y configuro las distintas solicitudes HTTP (GET, POST, PUT, DELETE). A continuación, voy a explicar cómo probar cada uno de los métodos CRUD con esta herramienta.

2. Crear un Cliente (POST)

Paso 1: Configuración de la solicitud POST

- Abro Postman y selecciono el método POST.
- En la barra de URL, ingreso la dirección del endpoint de creación de cliente: http://localhost/api_backend_dev/API-REST/API.REST.PHP/api-rest/create_client.php

Paso 2: Agregar los parámetros

- Clic en la pestaña Body de la solicitud.
- Selecciono x-www-form-urlencoded como tipo de cuerpo (esto es equivalente a enviar datos de formulario).
- Luego, agrego los siguientes parámetros (según el código que tienes en la interfaz):
    - email: el correo electrónico del cliente.
    - name: el nombre del cliente.
    - city: la ciudad del cliente.
    - telephone: el teléfono del cliente.

Paso 3: Enviar la solicitud

- Clic en el botón “Send”. El cliente se crea con éxito, se recibe una respuesta del servidor que indica que el cliente se ha guardado correctamente ("Cliente creado correctamente").

3. Obtener Todos los Clientes (GET)

Paso 1: Configuración de la solicitud GET

- Cambio el método a GET.
- En la barra de URL, ingreso la dirección de tu endpoint para obtener todos los clientes: http://localhost/api_backend_dev/API-REST/API.REST.PHP/api-rest/get_all_client.php

Paso 2: Enviar la solicitud

- Clic en Send. La respuesta es una lista de todos los clientes registrados en el sistema, como un objeto JSON con los detalles de cada cliente.

4. Actualizar un Cliente (PUT)

Paso 1: Configuración de la solicitud PUT

- Cambia el método a PUT.
- En la barra de URL, ingreso la dirección del endpoint para actualizar el cliente. http://localhost/api_backend_dev/API-REST/API.REST.PHP/api-rest/update_client.php?id=1

Paso 2: Agregar los parámetros de actualización

- En la pestaña Body, selecciono x-www-form-urlencoded.
- Luego, agregó los parámetros que deseo actualizar, por ejemplo:
    - email: nuevo correo electrónico.
    - name: nuevo nombre.
    - city: nueva ciudad.
    - telephone: nuevo teléfono.

Paso 3: Enviar la solicitud

- Clic en “Send”. La respuesta es "Cliente actualizado correctamente".

5. Eliminar un Cliente (DELETE)

Paso 1: Configuración de la solicitud DELETE

- Cambia el método a DELETE.
- En la barra de URL, ingresa la dirección de tu endpoint para eliminar el cliente, incluyendo el ID del cliente a eliminar: http://localhost/api_backend_dev/API-REST/API.REST.PHP/api-rest/delete_client.php?id=1

Paso 2: Enviar la solicitud

Clic en "Send". La respuesta debe confirmar que el cliente fue eliminado correctamente, como un mensaje en JSON indicando que la operación fue exitosa.

6. Respuestas del Servidor

Para cada solicitud que se realice, la API responderá con un código de estado HTTP y una respuesta en formato JSON. Algunas respuestas comunes incluyen:

- 201 Created: Al crear un cliente con éxito.
- 200 OK: Al obtener la lista de clientes o actualizar un cliente.
- 204 No Content: Al eliminar un cliente con éxito.
- 400 Bad Request: Cuando faltan parámetros requeridos o el método HTTP es incorrecto.
- 404 Not Found: Si el ID proporcionado no corresponde a ningún cliente.

Resumen del flujo en Postman:

- POST: Crear cliente -> Se envían datos en el cuerpo de la solicitud.
- GET: Obtener todos los clientes -> Respuesta JSON con la lista de clientes.
- PUT: Actualizar cliente -> Se envían los nuevos datos de cliente con el ID en la URL.
- DELETE: Eliminar cliente -> Se envía el ID del cliente en la URL.

6. Autenticación (no aplica)

En este caso el proyecto no utiliza autenticación de usuario, pero se puede agregar una sección con:

- **Autenticación mediante sesión**.
- **Autenticación mediante token** (si es una API REST).

7. Frontend

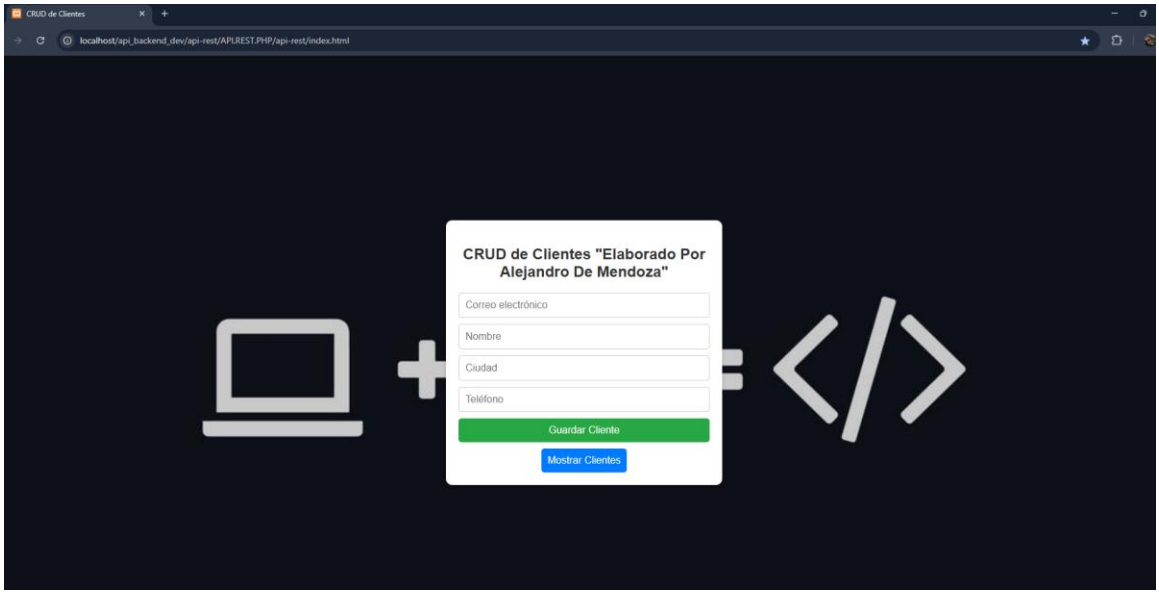
El frontend está construido con HTML y CSS para interactuar con el backend PHP. Este desarrollo se puede ver en el archivo index.html y en su código explicado anteriormente.

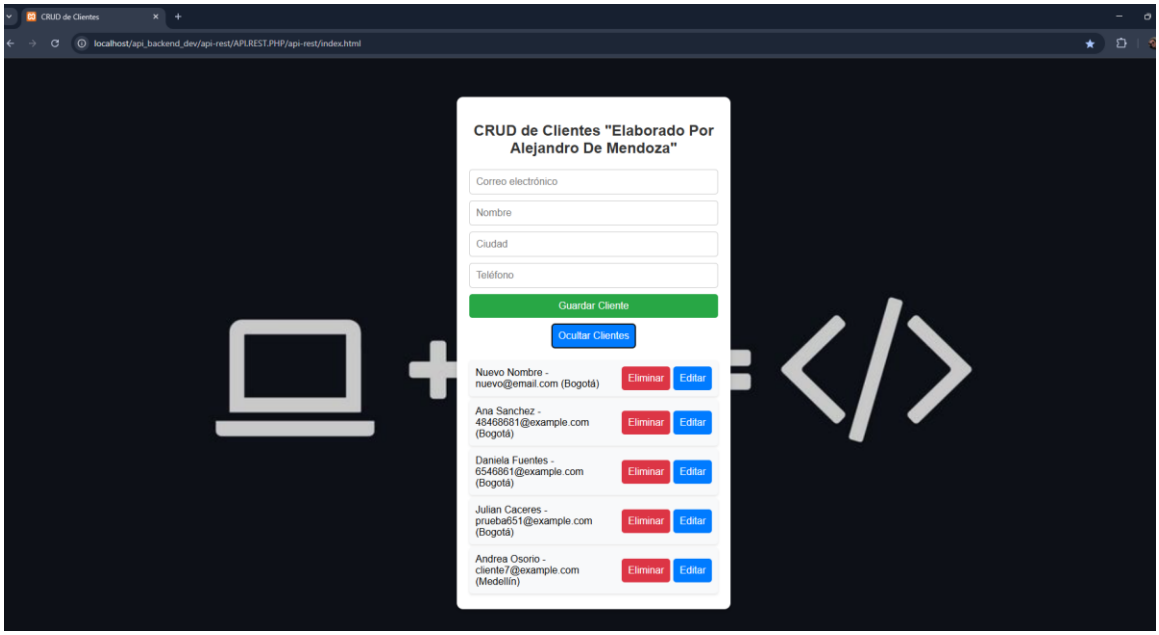
8. Pruebas

Para probar las funcionalidades de la API, se utilizó la herramienta **Postman.**

Revisión Desarrollo En HTML

Como este proyecto tiene como objetivo desarrollar una aplicación web para la gestión de clientes utilizando un enfoque de arquitectura API RESTful. Y las pruebas en Postman permitieron validar que la API interactuaba correctamente con la base de datos, permitiendo crear, leer, actualizar y eliminar registros de clientes. Por ende, una vez confirmada la funcionalidad de la API, el siguiente paso fue crear la interfaz de usuario utilizando tecnologías web como HTML, CSS y JavaScript. Esta interfaz proporciona un formulario intuitivo para la entrada de datos y botones para interactuar con la API, realizando las operaciones CRUD en tiempo real. En cuanto al HTML, se estructuraron las secciones esenciales de la aplicación: un formulario de ingreso de datos, un botón para mostrar la lista de clientes y un área donde se presentarán los registros de los clientes almacenados. Se utilizó CSS para mejorar la presentación visual de la página. La estética de la interfaz incluye una estructura centrada con un diseño limpio, utilizando una imagen de fondo que no interfiere con la legibilidad del contenido. Los botones y campos de entrada tienen un estilo moderno, con bordes redondeados y sombras suaves para mejorar la experiencia visual del usuario. Además, se implementaron efectos interactivos en los botones, como cambios de color al pasar el cursor, para hacer que la interfaz sea más dinámica y atractiva. Con JavaScript, se manejaron las interacciones entre el frontend y la API. El uso de fetch() permitió realizar las solicitudes HTTP a la API, pasando datos a través de formularios y recuperando información para mostrarla en la interfaz de usuario. Cada vez que un cliente es agregado, editado o eliminado, la lista de clientes se actualiza automáticamente sin necesidad de recargar la página por lo que se asegura que el sistema sea interactivo, ágil y fácil de usar. A través de la combinación de la funcionalidad robusta del backend y la interfaz intuitiva del frontend. A continuación, la imagen del desarrollo web:





Conclusión

Este proyecto demuestra cómo crear una API REST sencilla utilizando PHP y MySQL, con autenticación básica, sin el uso de frameworks ni librerías adicionales cubren todos los aspectos necesarios para un CRUD funcional en una API REST.

Y adicionalmente, este proyecto permite la gestión básica de clientes con un API RESTful implementado en PHP y una interfaz frontend para la interacción del usuario. Es una base que puede ser ampliada con más características como autenticación, validación de formularios y más funcionalidades.

¡¡¡Mil gracias y feliz día!!!

Alejandro De Mendoza

Celular: +573112687118

Bogotá