

El resultado de ejecución API Rest desde archivo cliente.php (como en el anunciado de la tarea), captura:

localhost/cliente.php

localhost/cliente.php

Testing GET /productos/{codigo}

--- Response Start ---
HTTP Status Code: 200

Array
(
 [Nombre producto] => Vestido de verano
 [Descripcion producto] => Vestido bohemio con estampados florales
)

--- Response End ---

Testing GET /productos/stock/{producto_id}

--- Response Start ---
HTTP Status Code: 200

Array
(
 [Tienda nombre] => Zara
 [Stock] => 50.000
)

--- Response End ---

Testing POST /tiendas

--- Response Start ---
HTTP Status Code: 201

Array
(
 [Response] => Tienda creada con exito
)

--- Response End ---

Testing DELETE /tiendas/{tienda_id}


--- Response Start ---
HTTP Status Code: 200

Array
(
 [message] => Store with ID 72 deleted successfully
)

--- Response End ---

Alexey Suzdalenko: Perfil públic

fpadistancia.educantabria.es/use...



Alexey Suzdalenko

Editar perfil

Detalles de usuario

Miscelánea

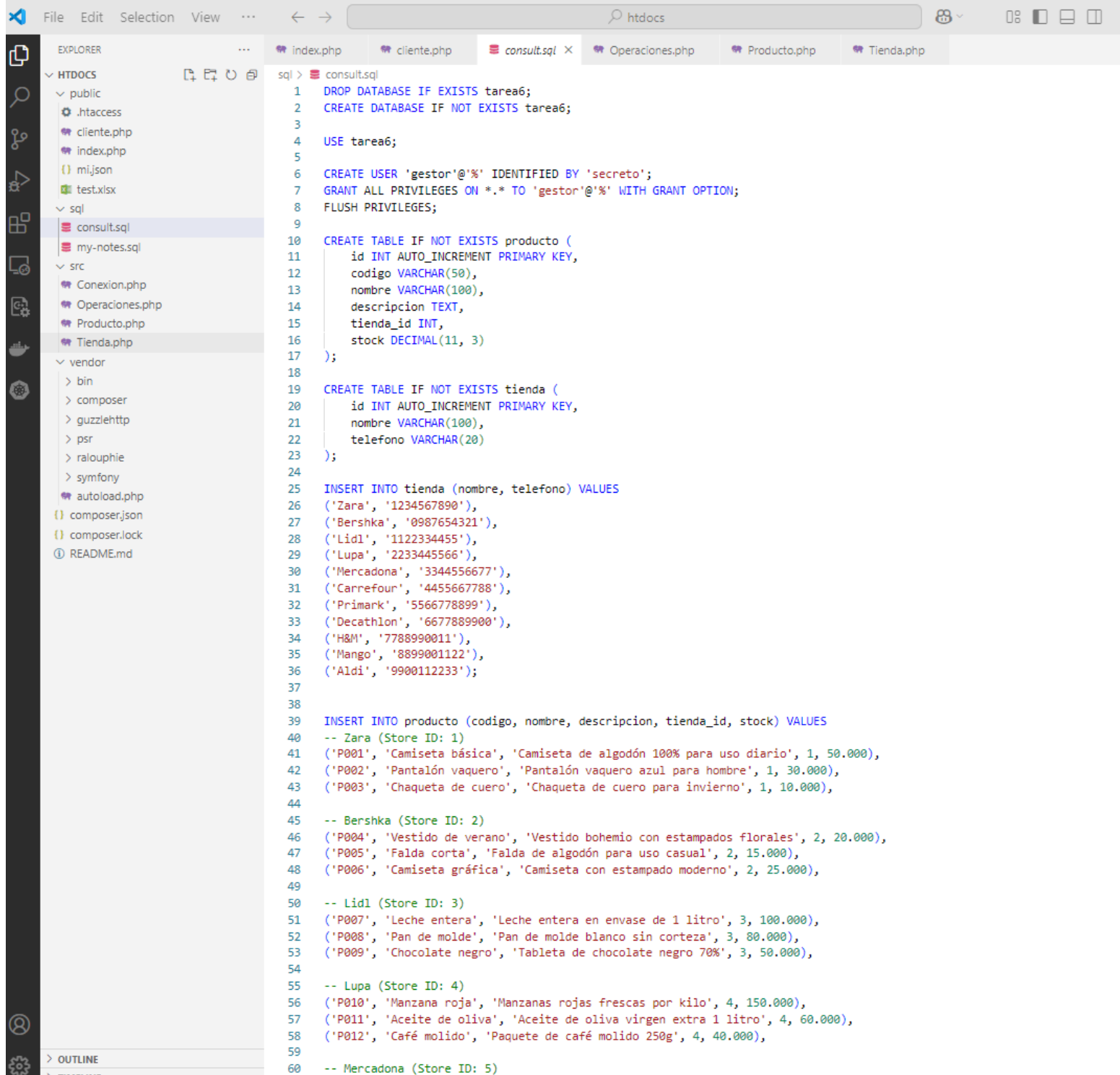
Mensajes en for

Foros de discusi

Planes de apren

Informes

Para llegar a este resultado (completar la tarea), lo primero he preparado la base de datos archivo “consult.sql” el que crea la base de datos “tarea6” usuario “gestor” y tablas producto y tienda, rellenándolas de datos de ejemplo



el resultado de la ejecución del archivo “consult.sql” podemos ver el la consola de mysql, tablas producto y tienda captura:

C:\WINDOWS\System32\cmd.exe - mysql -u root -p

id	nombre	telefono
1	Zara	1234567890
2	Bershka	0987654321
3	Lidl	1122334455
4	Lupa	2233445566
5	Mercadona	3344556677
6	Carrefour	4455667788
7	Primark	5566778899
8	Decathlon	6677889900
9	H&M	7788990011
10	Mango	8899001122
11	Aldi	9900112233

11 rows in set (0.001 sec)



Alexey Suzdalenko

Editar perfil

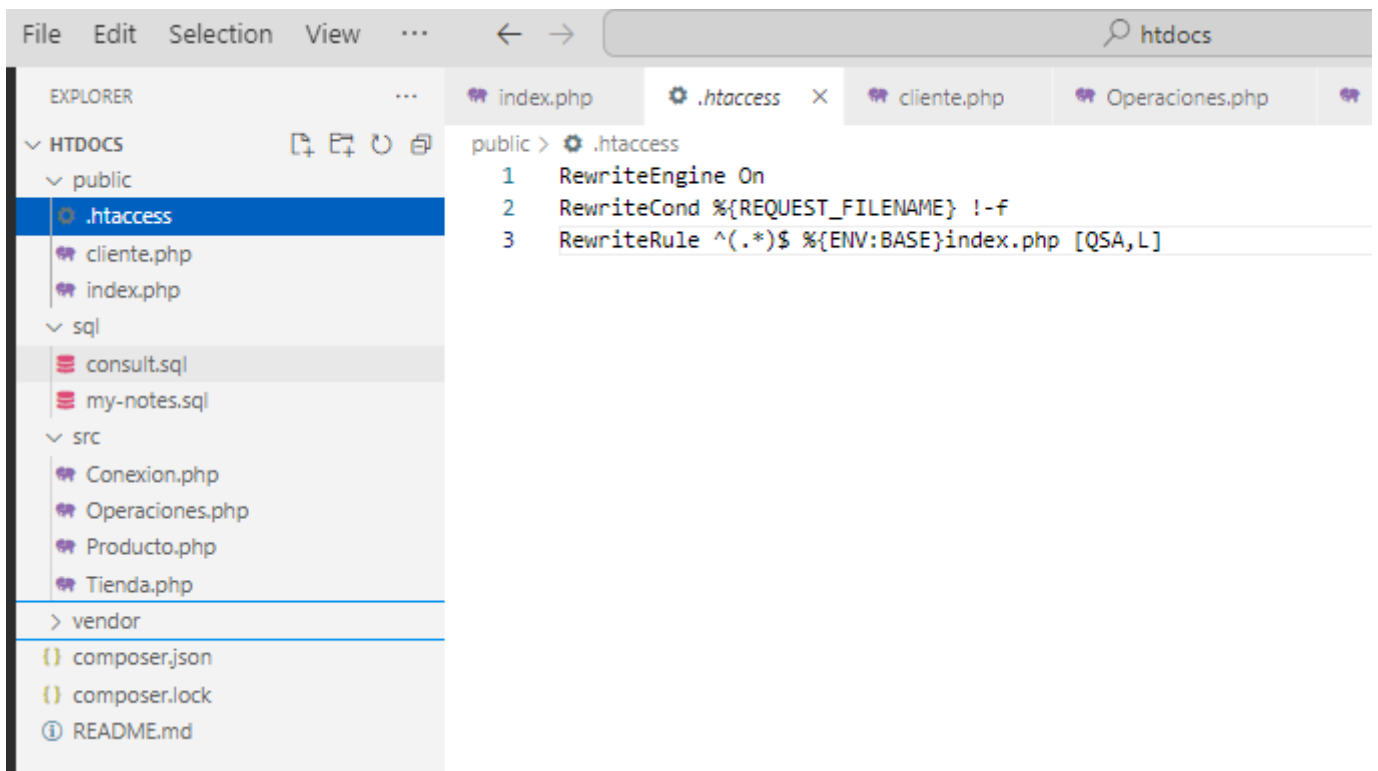
id	codigo	nombre	descripcion	tienda_id	stock
1	P001	Camiseta básica	Camiseta de algodón 100% para uso diario	1	50.000
2	P002	Pantalón vaquero	Pantalón vaquero azul para hombre	1	30.000
3	P003	Chaqueta de cuero	Chaqueta de cuero para invierno	1	10.000
4	P004	Vestido de verano	Vestido bohemio con estampados florales	2	20.000
5	P005	Falda corta	Falda de algodón para uso casual	2	15.000
6	P006	Camiseta gráfica	Camiseta con estampado moderno	2	25.000
7	P007	Leche entera	Leche entera en envase de 1 litro	3	100.000
8	P008	Pan de molde	Pan de molde blanco sin corteza	3	80.000
9	P009	Chocolate negro	Tableta de chocolate negro 70%	3	50.000
10	P010	Manzana roja	Manzanas rojas frescas por kilo	4	150.000
11	P011	Aceite de oliva	Aceite de oliva virgen extra 1 litro	4	60.000
12	P012	Café molido	Paquete de café molido 250g	4	40.000
13	P013	Arroz blanco	Arroz blanco de grano largo 1 kg	5	200.000
14	P014	Tomate frito	Tomate frito natural en bote de cristal	5	50.000
15	P015	Galletas integrales	Paquete de galletas integrales con avena	5	75.000
16	P016	Detergente líquido	Detergente para ropa 3 litros	6	70.000
17	P017	Pañal higiénico	Paquete de 12 rollos de pañal higiénico	6	90.000

Para que la ruta absoluta inicial sea: "localhost/servicios/unidad06/..." y se trabaje desde "/public/index.php" he modificado el archivo "C:\xampp1\apache\conf\httpd.conf" añadiéndole:

```
<VirtualHost *:80>
    DocumentRoot "C:\xampp1\htdocs\public"
    ServerName localhost

    <Directory "C:\xampp1\htdocs\public">
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

y dentro de la carpeta public aparte de crear archivos "cliente.php" y "index.php", se crea archivo ".htaccess con el siguiente contenido:



Una vez creados la base datos y la carpeta public, he ejecutado los siguientes comandos de composer para instalar el esqueleto de la aplicación y Guzzle php, para ello:

1. `composer init`
`suzdalenko/task`
`namespace Suzdalenko\Task;`
2. `composer require guzzlehttp/guzzle`
3. `composer dump-autoload`
4. `composer dump-autoload --optimize`

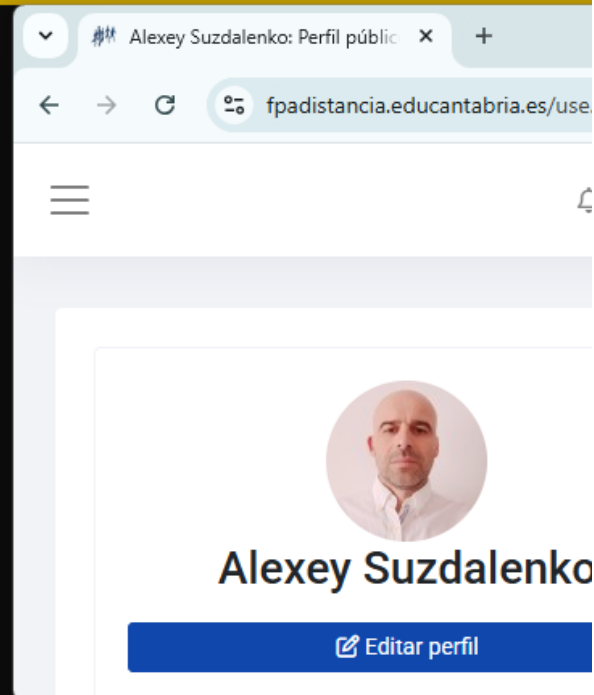
```
C:\WINDOWS\System32\cmd.exe

"email": "alexey.saron@gmail.com"
}
],
"require": {}
}

Do you confirm generation [yes]? yes
Composer could not detect the root package (suzdalenko/task06) version, defaulting to '1.0.0'. See https://getcomposer.org/root-version
Generating autoload files
Generated autoload files
PSR-4 autoloading configured. Use "namespace Suzdalenko\Task06;" in src/
Include the Composer autoloader with: require 'vendor/autoload.php';

E:\_download>composer require guzzlehttp/guzzle
Composer could not detect the root package (suzdalenko/task06) version, defaulting to '1.0.0'. See https://getcomposer.org/root-version
./composer.json has been updated
Composer could not detect the root package (suzdalenko/task06) version, defaulting to '1.0.0'. See https://getcomposer.org/root-version
Running composer update guzzlehttp/guzzle
Loading composer repositories with package information
Updating dependencies
Lock file operations: 8 installs, 0 updates, 0 removals
- Locking guzzlehttp/guzzle (7.9.2)
- Locking guzzlehttp/promises (2.0.4)
- Locking guzzlehttp/psr7 (2.7.0)
- Locking psr/http-client (1.0.3)
- Locking psr/http-factory (1.1.0)
- Locking psr/http-message (2.0)
- Locking ralouphie/getallheaders (3.0.3)
- Locking symfony/deprecation-contracts (v3.5.1)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 8 installs, 0 updates, 0 removals
- Downloading symfony/deprecation-contracts (v3.5.1)
- Downloading psr/http-message (2.0)
- Downloading psr/http-client (1.0.3)
- Downloading ralouphie/getallheaders (3.0.3)
- Downloading psr/http-factory (1.1.0)
- Downloading guzzlehttp/psr7 (2.7.0)
- Downloading guzzlehttp/promises (2.0.4)
- Downloading guzzlehttp/guzzle (7.9.2)
- Installing symfony/deprecation-contracts (v3.5.1): Extracting archive
- Installing psr/http-message (2.0): Extracting archive
- Installing psr/http-client (1.0.3): Extracting archive
- Installing ralouphie/getallheaders (3.0.3): Extracting archive
- Installing psr/http-factory (1.1.0): Extracting archive
- Installing guzzlehttp/psr7 (2.7.0): Extracting archive
- Installing guzzlehttp/promises (2.0.4): Extracting archive
- Installing guzzlehttp/guzzle (7.9.2): Extracting archive
3 package suggestions were added by new dependencies, use `composer suggest` to see details.
Generating autoload files
4 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
Using version ^7.9 for guzzlehttp/guzzle

E:\_download>composer dump-autoload --optimize
Composer could not detect the root package (suzdalenko/task06) version, defaulting to '1.0.0'. See https://getcomposer.org/root-version
Generating optimized autoload files
Generated optimized autoload files containing 104 classes
```



lo que se consigue con esto es la instalación de Guzzle → una biblioteca php para hacer peticiones GET, POST, PUT.. y etc. y de esta forma se crea el esqueleto del proyecto con la carpeta src/ de la cual se incluyen los archivos php automaticamente solo haciendo `require '../vendor/autoload.php'`:

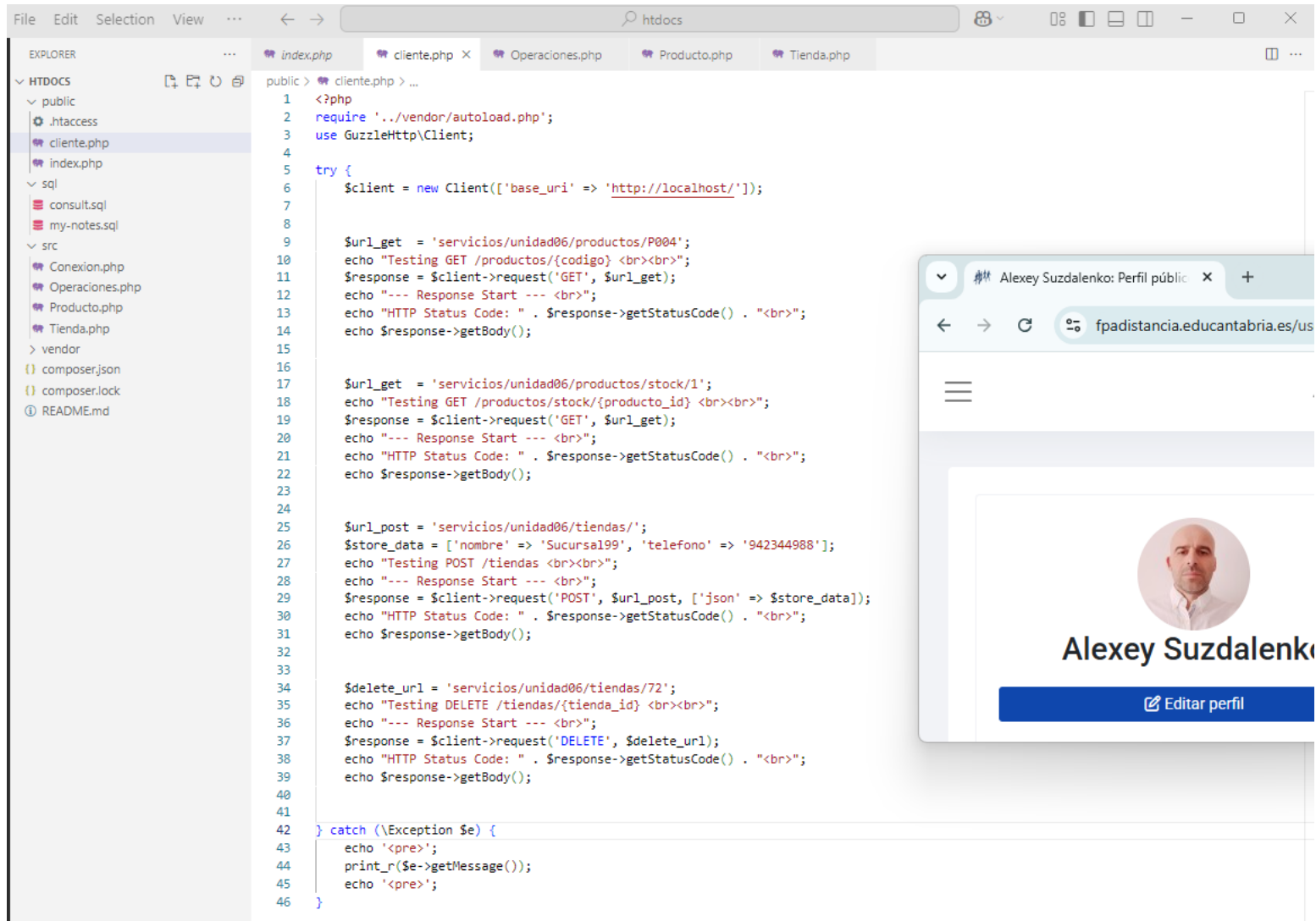
La idea es que cuando ejecutamos <http://localhost/cliente.php>, en el que usamos la biblioteca Guzzle lo que vamos hacer es una serie de REQUEST (peticiones) GET, POST y DELETE, y todos ellos hacia index.php, las rutas correspondientes son:

GET <http://localhost/servicios/unidad06/productos/P004>

GET <http://localhost/servicios/unidad06/productos/stock/1>

POST <http://localhost/servicios/unidad06/tiendas/>

DELETE <http://localhost/servicios/unidad06/tiendas/72>

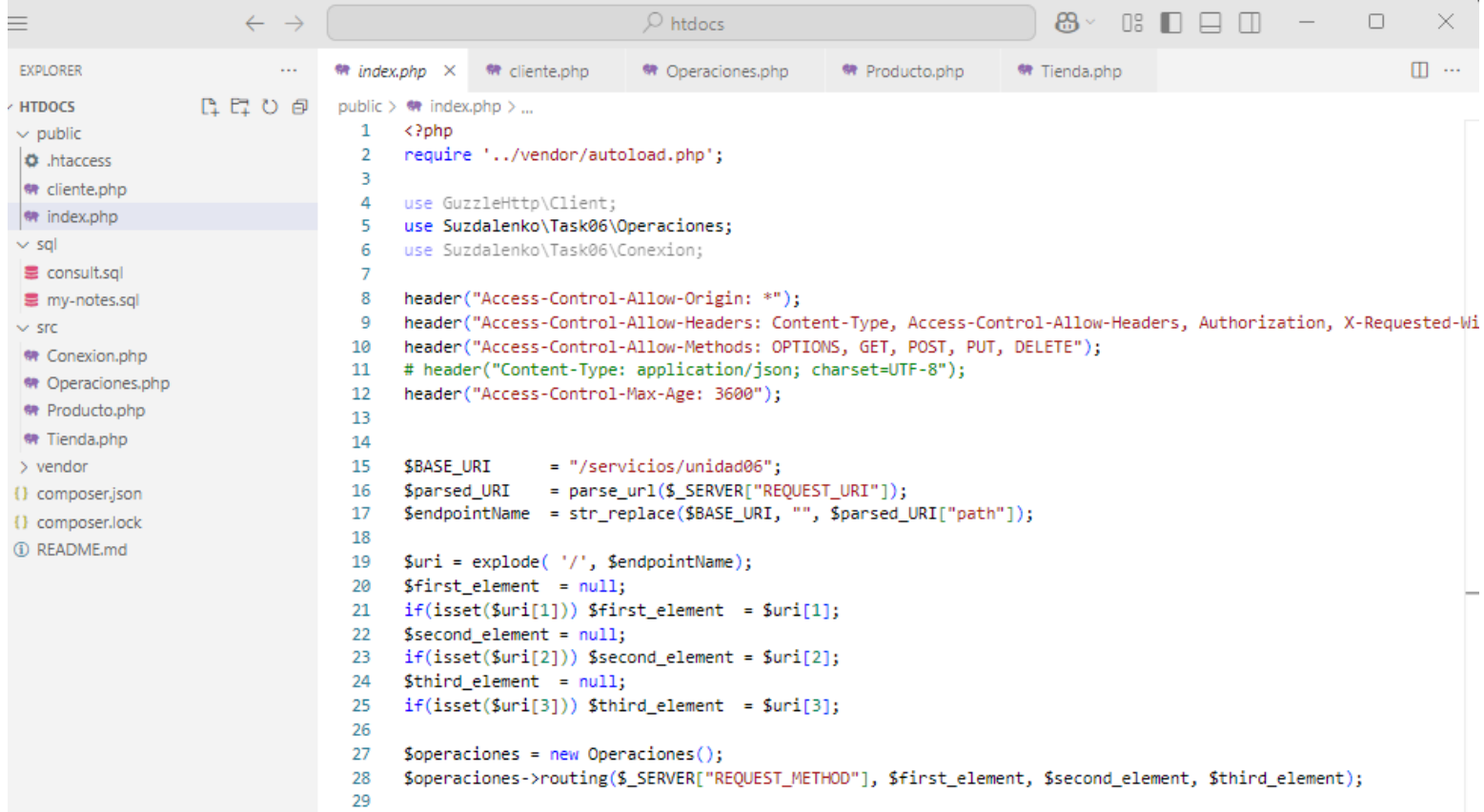


En index.php estamos preparados para recibir estas peticiones y con cada una de ellas los que hacemos es parseamos la url de las peticiones para saber si vamos a trabajar con productos o con tiendas

```
$BASE_URI      = "/servicios/unidad06";
$parsed_URI    = parse_url($_SERVER["REQUEST_URI"]);
$endpointName  = str_replace($BASE_URI, "", $parsed_URI["path"]);

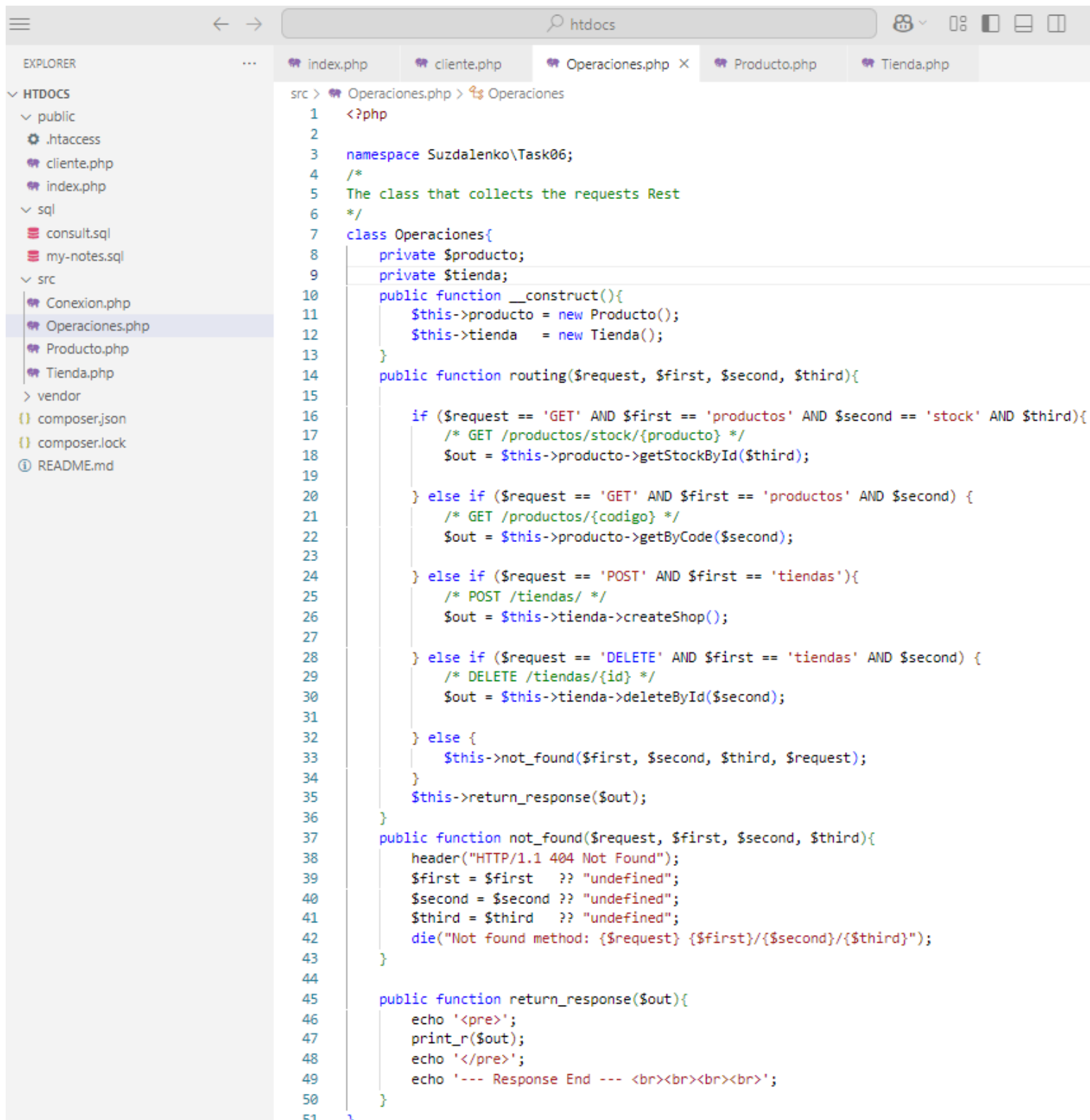
$uri = explode( '/', $endpointName);
$first_element  = null;
if(isset($uri[1])) $first_element  = $uri[1];
$second_element = null;
if(isset($uri[2])) $second_element = $uri[2];
$third_element  = null;
if(isset($uri[3])) $third_element  = $uri[3];
```

y también sabemos el tipo de petición que no llega gracias a \$_SERVER["REQUEST_METHOD"]; Instanciamos la clase "Operaciones" y le pasamos toda esta información a su método "routing" el que se va encargar de llamar diferente tipo de funcionalidad.



```
1 <?php
2 require '../vendor/autoload.php';
3
4 use GuzzleHttp\Client;
5 use Suzdalenko\Task06\Operaciones;
6 use Suzdalenko\Task06\Conexion;
7
8 header("Access-Control-Allow-Origin: *");
9 header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
10 header("Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT, DELETE");
11 # header("Content-Type: application/json; charset=UTF-8");
12 header("Access-Control-Max-Age: 3600");
13
14
15 $BASE_URI      = "/servicios/unidad06";
16 $parsed_URI    = parse_url($_SERVER["REQUEST_URI"]);
17 $endpointName  = str_replace($BASE_URI, "", $parsed_URI["path"]);
18
19 $uri = explode( '/', $endpointName);
20 $first_element = null;
21 if(isset($uri[1])) $first_element = $uri[1];
22 $second_element = null;
23 if(isset($uri[2])) $second_element = $uri[2];
24 $third_element = null;
25 if(isset($uri[3])) $third_element = $uri[3];
26
27 $operaciones = new Operaciones();
28 $operaciones->routing($_SERVER["REQUEST_METHOD"], $first_element, $second_element, $third_element);
29
```

A la hora de instancia la clase Operaciones en su constructor instanciamos clases Producto y Tienda (estas clases y sus funciones se encargan de trabajar con la base de datos), muy importante el método routing() ya que este método depende de los parámetros de la url de la petición llama una clase u otra y a un método u otro. Si el if del método de routing() no encuentra ninguna coincidencia en la ruta => estamos en cuenta que esta ruta no es válida por lo que devolvemos el mensaje "Not found method"



```
1 <?php
2
3 namespace Suzdalenko\Task06;
4 /*
5  The class that collects the requests Rest
6  */
7 class Operaciones{
8     private $producto;
9     private $tienda;
10    public function __construct(){
11        $this->producto = new Producto();
12        $this->tienda = new Tienda();
13    }
14    public function routing($request, $first, $second, $third){
15
16        if ($request == 'GET' AND $first == 'productos' AND $second == 'stock' AND $third){
17            /* GET /productos/stock/{producto} */
18            $out = $this->producto->getStockById($third);
19
20        } else if ($request == 'GET' AND $first == 'productos' AND $second) {
21            /* GET /productos/{codigo} */
22            $out = $this->producto->getByCode($second);
23
24        } else if ($request == 'POST' AND $first == 'tiendas'){
25            /* POST /tiendas/ */
26            $out = $this->tienda->createShop();
27
28        } else if ($request == 'DELETE' AND $first == 'tiendas' AND $second) {
29            /* DELETE /tiendas/{id} */
30            $out = $this->tienda->deleteById($second);
31
32        } else {
33            $this->not_found($first, $second, $third, $request);
34        }
35        $this->return_response($out);
36    }
37    public function not_found($request, $first, $second, $third){
38        header("HTTP/1.1 404 Not Found");
39        $first = $first ?? "undefined";
40        $second = $second ?? "undefined";
41        $third = $third ?? "undefined";
42        die("Not found method: {$request} {$first}/{ $second}/{ $third}");
43    }
44
45    public function return_response($out){
46        echo '<pre>';
47        print_r($out);
48        echo '</pre>';
49        echo '---- Response End --- <br><br><br><br>';
50    }
51 }
```

Las clases Producto y Tienda como lo hemos dicho se encargan en ejecutar consultas hacia la base datos buscando el producto por su código, la conexión en si a la base datos se obtiene desde la clase “Conexion.php” haciendo una instancia de esta clase que esta escrita en el modo de singleton lo que permite es una vez instanciada esta clase por mas veces que se vuelva a instanciar siempre devuelve la misma conexión, de esta forma solo se crea una conexión a la base datos por petición (o por ejecución de código).


```

public function getByCode($code){
    $statement = "SELECT nombre AS 'Nombre producto', descripcion AS 'Descripcion producto' FROM producto WHERE codigo = ?";
    try {
        $statement = $this->conn->prepare($statement);
        $statement->execute([$code]);
        $result = $statement->fetch(\PDO::FETCH_ASSOC);
        if (!$result) {
            $result = ['Producto no encontrado'];
        }
        return $result;
    } catch (\PDOException $e) {
        header('HTTP/1.1 500 Internal Server Error');
        echo "Error en la base de datos: " . $e->getMessage();
    }
}

```

Clase producto busca el stock y el nombre de la tienda en el método getStockById(\$id)

```

public function getStockById($id){
    $statement = "SELECT t.nombre AS 'Tienda nombre', IFNULL(SUM(stock), 0) AS Stock
    FROM producto p
    LEFT JOIN tienda t ON t.id = p.tienda_id
    WHERE p.id = ?
    GROUP BY t.id
    ";
    try {
        $statement = $this->conn->prepare($statement);
        $statement->execute([$id]);
        $result = $statement->fetch(\PDO::FETCH_ASSOC);
        if (!$result) {
            $result = ['Producto no encontrado'];
        }
        return $result;
    } catch (\PDOException $e) {
        header('HTTP/1.1 500 Internal Server Error');
        echo "Error en la base de datos: " . $e->getMessage();
    }
}

```

Clase Tienda método CreateShop() crea una linea nueva en la tabla tienda:

```

public function createShop(){
    $string_json = file_get_contents('php://input');
    $object_json = json_decode($string_json, true);

    $statement = "INSERT INTO tienda (nombre, telefono) VALUES (:nombre, :telefono)";
    try {
        $statement = $this->conn->prepare($statement);
        $statement->execute(['nombre' => $object_json['nombre'], 'telefono' => $object_json['telefono'] ]);
        $lastInsertId = $this->conn->lastInsertId();
        http_response_code(201);
        $result = ['Response' => 'Tienda creada con exito'];
        if (!$lastInsertId) {
            $result = ['Response' => 'Error en la creacion de tienda'];
        }
        return $result;
    } catch (\PDOException $e) {
        die($e->getMessage());
    }
}

```

Y el metodo deleteById(\$id) de la clase tienda elimina la linea en la base datos por el id de la linea, que lógicamente llega desde la url (ejemplo <http://localhost/servicios/unidad06/tiendas/72>) en este caso el 72 es el id para eliminar la linea, siempre y cuando el método sea "DELETE".

```

public function deleteById($id){
    $statement = "DELETE FROM tienda WHERE id = ?";
    $stmt = $this->conn->prepare($statement);
    if ($stmt->execute([$id])) {
        if ($stmt->rowCount() > 0) {
            http_response_code(200);
            $result = ['message' => "Store with ID $id deleted successfully"];
        } else {
            http_response_code(404);
            $result = ['message' => 'Error deleting the store'];
        }
    } else {
        http_response_code(500); // Internal Server Error
        $result = ['message' => ' Internal Server Error'];
    }
    return $result;
}

```

Resumen:

Crear la estructura del proyecto e instalar todas las dependencias necesarias con Composer: SI
 Programar correctamente el index.php SI
 Crear y gestionar los endpoint en Operaciones.php SI, en mi caso es con if, else if, else
 Conexión con la bbdd en fichero independiente SI
 Crear de forma correcta productos.php y tiendas.php SI
 Crear la estructura de directorios según lo pedido en el enunciado Si
 Control adecuado de errores SI
 Crear de forma correcta cliente.php para testar la ejecución del servicio REST SI
 El diseño del código es adecuado y el código está comentado , pienso que SI.