CRUD

Un crud es es el acrónimo de las 4 operaciones que se pueden hacer en la base de datos

C: crear
R: leer
U: modificar
D: borrar

Ahora vamos a hacer nuestro propio CRUD en PHP con altos estándares arquitectónicos (Arquitectura limpia):

/project-root
|-- assets/
|-- config/
|-- controllers/
|-- includes/
| |-- header.php
| |-- footer.php
| |-- TEMPLATES.php
|-- js/
| |-- scripts.js
|-- models/
|-- repositories/
|-- services/
|-- index.php

Descripción de la funcionalidad de las capas

Capa de configuración:

Se responsabiliza de gestionar la conexión con la base de datos y las credenciales.

Capa de controladores:

Contiene los controladores que manejan las solicitudes HTTP y las respuestas.

Capa de includes:

Contiene todas las vistas que deben de ser renderizadas incluyendo los elementos comunes como el HEADER y el FOOTER.

Capa de Modelos:

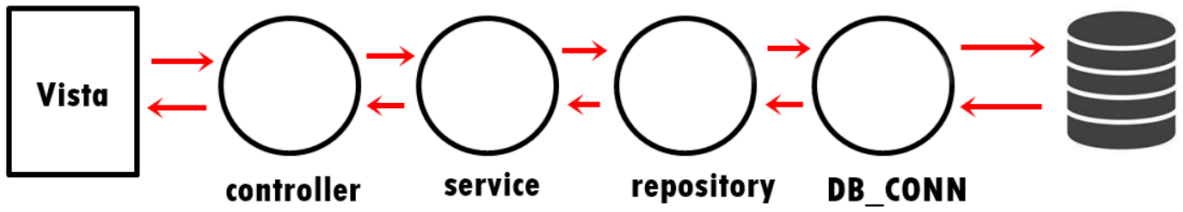
Contiene las abstracciones de los modelos de la lógica de negocio. representa a la estructura de los datos.

Capa de repositorios:

Es un paso antes de la conexión con la base de datos contiene todas las consultas SQL, interactúa con la base de datos y las clases que manejan el acceso a datos.

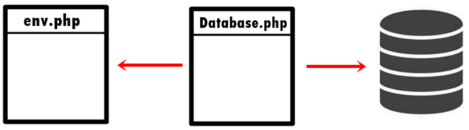
Capa de servicios:

Contiene las clases de servicios que manejan la lógica del negocio. Esta se conecta con el repositorio para poder solucionar las operaciones.



Conexión a la base de datos

Se necesitan 2 archivos uno de environment el cual contiene todas las credenciales para conectarse y uno de conexión a la base de datos el cual se encarga de retornar la conexión a la base de datos.



config > env.php

```php
<?php
return [
    'host' => 'localhost',
    'dbname' => 'exampledb',
    'username' => 'root',
    'password' => 'kmzwa8awaa'
];
```

config > Database.php

```php
<?php

class Database {
    private $conn;

    public function getConnection() {
        $this->conn = null;

        try {
            $config = include __DIR__ . '/env.php';
            $this->conn = new PDO("mysql:host=" . $config['host'] . ";dbname=" . $config['dbname'], $config['username'], $config['password']);
            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        } catch (PDOException $exception) {
            echo "Error de conexión: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
?>
```

R:Obtener registros de la base de datos

Mi lógica es la siguiente… lo que yo quiero es tener un formulario que consuma por GET y que tenga una barra de búsqueda la cual al presionar un botón muestre los registros que coinciden con las siguientes reglas:

- Si se ingresa un número se busca por el ID
- Si se ingresa una palabra se busca que contenga el nombre o que contenga esa palabra en la descripción.

1 - Hay que crear una vista que cumpla con una barra de búsqueda y un botón:

```html
<form method="GET" action="READ.php">
    <input type="text" name="q" placeholder="Buscar por ID/NAME/DESCRIPTION" required>
    <button type="submit">Buscar</button>
</form>
```

2 - Crear un script de php el cual contenga la llamada a un servicio y sea capaz de conectarse a un repositorio que tenga todas nuestras consultas personalizadas:

```php
<?php
if (isset($_GET['q'])) {
    require_once __DIR__ . '/../controllers/ExampleController.php';
    $exampleController = new ExampleController();
    $examples = $exampleController->getExampleByQuery($_GET['q']);

    if ($examples) {
        echo '<br>';
        echo 'TOTAL FOUND: ' . count($examples).'<br>';
        echo '<br>';
        for ($i = 0; $i < count($examples); $i++) {
            $example = $examples[$i];

            echo 'Product ID:'. $example->id . '<br>';
            echo  $example->name . '<br>';
            echo  $example->description . '<br>';
            echo  '<br>';

        }
    } else {
        echo '<p>Not found data.</p>';
    }
}
?>
```

3 - Crear el servicio el cual solo va a ser una conexión con el repositorio:

services > ExampleService.php

```php
class ExampleService {

    public function getExampleByQuery($q) {
        $data = [];

        $example = $this->repository->getById($q);

        if ($example != null){
            $data[] = $example;
        }

        $example = $this->repository->getByLikeName($q);

        if($example != null){
            for ($i = 0; $i < count($example); $i++) {
                $data[] = $example[$i];
            }
        }

        $example = $this->repository->getByLikeDescription($q);

        if($example != null){
            for ($i = 0; $i < count($example); $i++) {
                $data[] = $example[$i];
            }
        }

        return $data;
    }
```

4 - Crear las llamadas en el repositorio:

itories > ExampleRepository.php

```php
class ExampleRepository {

    public function getById($id) {
        $query = "SELECT id, NAME, DESCRIPTION FROM examples where id = :id";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->execute();

        $row = $stmt->fetch(PDO::FETCH_ASSOC);


        if ($row) {
            return new Example($row['id'], $row['NAME'], $row['DESCRIPTION']);
        } else {
            return null;
        }
    }
```

repositories > ExampleRepository.php

```php
5   class ExampleRepository {
25
26      public function getByLikeName($q) {
27          $query = "SELECT id, NAME, DESCRIPTION FROM examples WHERE NAME LIKE :q";
28          $stmt = $this->conn->prepare($query);
29          $likeQuery = '%' . $q . '%';
30          $stmt->bindParam(':q', $likeQuery);
31          $stmt->execute();
32
33          $examples = [];
34
35          while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
36              $examples[] = new Example($row['id'], $row['NAME'], $row['DESCRIPTION']);
37          }
38
39          return $examples;
40      }
41
```

Espero que el lector entienda que para generar el método que retorne una coincidencia en la descripción del ejemplo solo basta con copiar y pegar el método y hacer las respectivas modificaciones.

C:Crear un nuevo registro en la base de datos

Mi lógica es la siguiente: yo quiero un formulario que tenga 2 atributos el name y el description y será un formulario que tenga un método POST debido a vamos a “crear” ahora bien yo espero que el lector se de cuenta que no es buena práctica asignar al ID por que por lo general el usuario no tiene cómo saber cuál es el próximo ID.

1 -> Crear el formulario que va a contener nuestros datos nombre y descripción además del botón para enviar la información al servidor para que sea almacenada.

```
<h1>Create Example</h1>
<form action="CREATE.php" method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>

    <label for="description">Description:</label>
    <input type="text" id="description" name="description" required>

    <input type="submit" value="Create">
</form>
```

2 -> Crear el script que nos va a aceptar la petición post

```
<?php
    require_once __DIR__ . '/../controllers/ExampleController.php';
    $exampleController = new ExampleController();
    if ($_SERVER["REQUEST_METHOD"] == "POST") {

        $status = $exampleController->createNewExample($_POST['name'], $_POST['description']);

        if($status == -1){
            echo "<h1>ERROR INSERTING NEW EXAMPLE</h1>";
        }else{
            echo "<h1>The new Example is insert in DB </h1>";
            echo "<p>CODE:" . $status . "</p>";
        }

    }

?>
```

Advertencia: no se puede acceder a los atributos del $_POST sin envolverlo en un if que pregunte el método $_SERVER["REQUEST_METHOD"] == "POST"

3 -> Crear el link hacia el servicio en el controlador:

```
public function createNewExample($name, $description){
    return $this->service->createNewExample($name, $description);
}
```

4 -> Crear el link hacia el repositorio en el servicio:

```php
public function createNewExample($name, $description){
    return $this->repository->createNewExample($name, $description);
}
```

5 -> Crear en el repositorio el método que nos retornará el nuevo ID:

```php
public function getNextId() {
    $query = "SELECT COUNT(*) + 1 AS next_id FROM examples";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    return $row['next_id'];
}
```

6 -> Crear en el repositorio en método para insertar el nuevo example:

```php
public function createNewExample($name, $description){
    $id = $this->getNextId();

    $query = "INSERT INTO examples (id, NAME, DESCRIPTION) VALUES (:id, :name, :description)";
    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':id', $id);
    $stmt->bindParam(':name', $name);
    $stmt->bindParam(':description', $description);

    if ($stmt->execute()) {
        return $id;
    } else {
        return -1;
    }
}
```

U: Modificar los ejemplos

Mi lógica es la siguiente: voy a crear varios formularios que muestran los atributos de ejemplos: id,name, description.

Solo va a ser modificable el nombre y la descripción. El vital secreto está en que va a tener un input oculto el cual va a controlar cual es el elemento que se va a modificar:

1 -> Crear la vista:

```php
<?php
    require_once __DIR__ . '/../controllers/ExampleController.php';
    $exampleController = new ExampleController();
    $examples = $exampleController->getAllExamplesInArray();

    for ($i = 0; $i < count($examples); $i++){
        $example = $examples[$i];
        echo '<form action="UPDATE.php" method="POST" onsubmit="return confirmUpdate();">';
        echo '<p>ID: ' . htmlspecialchars($example->id) . '</p>';
        echo '<label for="name-' . htmlspecialchars($example->id) . '">Name:</label>';
        echo '<input type="text" id="name-' . htmlspecialchars($example->id) . '" name="name" value="' . htmlspecialchars($example->name) . '" required>';
        echo '<label for="description-' . htmlspecialchars($example->id) . '">Description:</label>';
        echo '<input type="text" id="description-' . htmlspecialchars($example->id) . '" name="description" value="' . htmlspecialchars($example->description) . '" required>';
        echo '<input type="hidden" name="id" value="' . htmlspecialchars($example->id) . '">';
        echo '<input type="submit" value="Update">';
        echo '</form><br>';
    }
?>
```

2 -> Crear un script que nos reciba esa solicitud POST

```php
<?php
    require_once __DIR__ . '/../controllers/ExampleController.php';
    $exampleController = new ExampleController();
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $id = $_POST['id'];
        $name = $_POST['name'];
        $description = $_POST['description'];

        $exampleController->updateExample($id, $name, $description);
        // WARINIG: not update current values in view
    }
?>
```

3 -> Crear el link en el controlador hacia el servicio:

```php
public function updateExample($id, $name, $description){
    $this->service->updateExample($id, $name, $description);
}
```

4 -> Crear el link en el servicio hacia el repositorio:

```php
public function updateExample($id, $name, $description){
    $this->repository->updateExample($id, $name, $description);
}
```

5 - Crear el método en el repositorio que haga el UPDATE

```php
public function updateExample($id, $name, $description) {
    $query = "UPDATE examples SET NAME = :name, DESCRIPTION = :description WHERE id = :id";
    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':id', $id);
    $stmt->bindParam(':name', $name);
    $stmt->bindParam(':description', $description);
    $stmt->execute();
}
```

D: borrar un elemento

Para este ejemplo no creo que se necesite detallar el formulario o el sistema de links entre controlador, servicio y repositorio … voy a pasar directo a el borrar en el repo:

```php
public function deleteExample($id){
    $query = "DELETE FROM examples WHERE id = :id";
    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':id', $id, PDO::PARAM_INT);
    $stmt->execute();
}
```