PHP Data Objects
-
PDO

1. Descrição

O PDO é uma extensão do PHP que fornece uma interface de programação de aplicativos (API) para acessar e trabalhar com diferentes bancos de dados relacionais de maneira uniforme. Ele foi introduzido no PHP 5.1 e tem como objetivo oferecer uma camada de abstração para operações de banco de dados, permitindo que os desenvolvedores escrevam código mais portável e seguro.

Principais características e conceitos do PDO:

1. Prevenção de Injeção de SQL: O PDO oferece suporte a prepared statements, uma técnica que ajuda a prevenir ataques de injeção de SQL. Prepared statements parametrizam as consultas SQL, separando os dados dos comandos SQL e evitando a execução maliciosa de código SQL.

2. Portabilidade de Banco de Dados: Uma das principais vantagens do PDO é sua capacidade de funcionar com vários sistemas de gerenciamento de banco de dados (SGDBs). Você pode usar o PDO para se conectar a bancos de dados como MySQL, PostgreSQL, SQLite, SQL Server, entre outros, sem precisar alterar drasticamente o código.

3. Tratamento de Erros Consistente: O PDO oferece um tratamento de erros consistente e flexível. Você pode configurar o modo como os erros são tratados, permitindo que você escolha entre exceções ou códigos de erro tradicionais.

4. Conexões Seguras e Reutilizáveis: O PDO permite a criação de conexões seguras e reutilizáveis com o banco de dados. Isso é feito através da criação de uma instância da classe PDO com as informações de conexão.

5. Consulta de Dados: O PDO oferece métodos para executar consultas SQL, recuperar resultados e iterar sobre esses resultados. Os resultados podem ser obtidos em diferentes formatos, como arrays associativos.

2. Aplicação no Trabalho

Quando se trata de aplicar o PDO em conjunto com Ajax em um ambiente web, geralmente seguimos uma arquitetura onde o frontend (Ajax) faz solicitações para o backend (PHP) e o backend utiliza o PDO para interagir com o banco de dados.

Com o frontend em Javascript, com uma biblioteca Ajax, e um backend em PHP usando PDO para interagir com o banco de dados PostgreSQL, temos a seguinte estrutura (Exemplo com as rotas de manipulação de usuários).

Javascript:

```
_listeners: function _listeners(){
    $(home.BOTAO_CADASTRAR_USUARIO).click(function(){
        let form = $(home.FORM_USUARIO).serializeArray();
        home._submit(home.URL_AJAX_USUARIO_POST, form, home.REQUEST_SET_USUARIO, function(retorno) {
            console.log(retorno);
        });
    });

    $(home.BOTAO_CADASTRAR_VACINA).click(function(){
        let form = $(home.FORM_VACINA).serializeArray();
        home._submit(home.URL_AJAX_VACINA_POST, form, home.REQUEST_SET_VACINA, function(retorno) {
            console.log(retorno);
        });
    });

    $(home.BOTAO_CADASTRAR_ALERGIA).click(function(){
        let form = $(home.FORM_ALERGIA).serializeArray();
        home._submit(home.URL_AJAX_ALERGIA_POST, form, home.REQUEST_SET_ALERGIA, function(retorno) {
            console.log(retorno);
        });
    });
    // home._getDados(home.URL_AJAX_USUARIO_GET, home.REQUEST_GET_USUARIOS)
},

_submit: function _submit(link, formulario, request, callback){
    $.ajax({
        url: link+request,
        method: 'POST',
        dataType: "json",
        async: false,
        data: formulario,
        success: function(data){
            callback(data);
        }
    });
},
```

Rotas AJAX em PHP:

```php
<?php

include_once("interfaces/usuarios.php");

header('Content-Type: application/json; charset=utf-8');

// Verifica se a requisição POST ou GET contém o parâmetro 'tipo'.
$getRequisicao = isset($_POST['tipo']) && !empty($_POST['tipo']) ? $_POST['tipo'] : NULL;
$postRequisicao = isset($_GET['tipo']) && !empty($_GET['tipo']) ? $_GET['tipo'] : NULL;

// Determina o tipo de requisição com base nos parâmetros encontrados.
$tipoRequisicao = is_null($getRequisicao) ? $postRequisicao : $getRequisicao;

$usuarios = new Usuarios();

// Executa diferentes ações com base no tipo de requisição.
switch ($tipoRequisicao) {
    case 'set-usuario':
        echo json_encode($usuarios->setUsuario($_POST));
        break;
    case 'upt-usuario':
        echo json_encode($usuarios->uptUsuario($_POST));
        break;
    case 'get-usuarios':
        echo json_encode($usuarios->getUsuarios($_GET));
        break;
    case 'get-usuario':
        echo json_encode($usuarios->getUsuario($_GET['id']));
        break;
    case 'del-usuario':
        echo json_encode($usuarios->delUsuario($_POST));
        break;
    default:
        echo json_encode("Erro AJAX: rota não encontrada.");
    break;
}

```

Backend em PHP e uso do PDO:

```php
1 reference
private function setUsuario($dados)
{
    try {
        if (!isset($dados) || !is_array($dados)) {
            throw new InvalidArgumentException("Dados invalidos.");
        }

        $stmt = $this->pdoPGS->prepare(self::SQL_INSERT_NOVO_USUARIO);
        $stmt->bindValue(':nome',               strval($dados['nome']), PDO::PARAM_STR);
        $stmt->bindValue(':data_nascimento',    strval($dados['data_nascimento']));
        $stmt->bindValue(':sexo',               strval($dados['sexo']), PDO::PARAM_STR);
        $stmt->bindValue(':logradouro',         strval($dados['logradouro']), PDO::PARAM_STR);
        $stmt->bindValue(':numero',             intval($dados['numero']), PDO::PARAM_INT);
        $stmt->bindValue(':setor',              strval($dados['setor']), PDO::PARAM_STR);
        $stmt->bindValue(':cidade',             strval($dados['cidade']), PDO::PARAM_STR);
        $stmt->bindValue(':uf',                 strval($dados['uf']), PDO::PARAM_STR);

        if (!$stmt->execute()){
            throw new PDOException("Erro PDO. Detalhes: " . $stmt->errorInfo()[2]);
        } else {
            return [
                'ERRO' => false,
                'MENSAGEM' => 'Usuario cadastrado com sucesso.',
                'DADOS' => []
            ];
        }

    } catch (Throwable $e){
        return [
            'ERRO' => true,
            'MENSAGEM' => $e->getMessage(),
            'DADOS' => []
        ];
    }
}
```

Declaração do PDO, importação no backend, e instanciação:

htdocs > app > ajax > interfaces > usuarios.php > ...

```php
<?php

include_once (__DIR__ . '/../../includes/Connection.php');

6 references | 0 implementations
class Usuarios
{
    //
    ------------------------------------------------------------
    ------
    // CONSULTAS SQL

    // INSERT
    1 reference
    const SQL_INSERT_NOVO_USUARIO = "INSERT INTO usuarios
    (nome, data_nascimento, sexo, logradouro, numero, setor,
    cidade, uf) VALUES (:nome, :data_nascimento, :sexo,
    :logradouro, :numero, :setor, :cidade, :uf)";
    // UPDATE
    1 reference
    const SQL_UPDATE_USUARIO = "UPDATE usuarios SET
    logradouro = :logradouro, numero = :numero, setor =
    :setor, cidade = :cidade, uf = :uf WHERE id = :id";
    // DELETE
    1 reference
    const SQL_DELETE_USUARIO = "DELETE FROM usuarios WHERE
    id = :id";
    // SELECT
    1 reference
    const SQL_SELECT_USUARIOS = "SELECT * FROM usuarios";

    1 reference
    const SQL_SELECT_USUARIO = "SELECT nome,
    data_nascimento, sexo, logradouro, numero, setor,
    cidade, uf FROM usuarios WHERE id = :id";
    //
    ------------------------------------------------------------
```

htdocs > app > includes > Connection.php > ...

```php
<?php

9 references | 0 implementations
class Connection
{
    1 reference
    const host = 'persistencia';
    1 reference
    const port = 5432;
    1 reference
    const dbname = 'postgres';
    1 reference
    const user = 'postgres';
    1 reference
    const password = '12345';

    4 references | 0 overrides
    public static function connect()
    {
        try {
            $dsn = 'pgsql:host='.self::host.';port='.
            self::port.';dbname='.self::dbname.';user='.
            self::user.';password='.self::password;
            $pdo = new PDO($dsn);
            return $pdo;
        } catch (PDOException $e) {
            echo "Erro de conexão: " . $e->getMessage();
        }
    }
}
```