Implementación de un Proyecto CRUD

Usando PHP Slim Framework y Composer

Tecnologías Web II

PHP

Introducción

- **Objetivo:** Desarrollar una aplicación web básica con funcionalidad **CRUD**.
- **Tecnologías Clave:**
  - **PHP Slim Framework:** Micro-framework para construir APIs y aplicaciones web.
  - **Composer:** Herramienta de gestión de dependencias para PHP.
  - **Base de Datos:** Usaremos **MySQL/MariaDB** (o SQLite, si prefieres).

Paso 1: Inicialización con Composer

- Composer es fundamental para la gestión de librerías.
- Crea un archivo `composer.json` y descarga Slim.
- **Comando Clave:**
- `composer require slim/slim:4.x`
- `composer require slim/psr7` (Para manejar peticiones y respuestas)

Estructura Inicial de Archivos:

```
.
├── vendor/
├── composer.json
├── composer.lock
└── public/
└── index.php
```

Paso 2: Configuración Básica de Slim

- Incluir el **autoloader** de Composer para cargar las dependencias.
- Crear y configurar la instancia de la aplicación (`App`).
- Definir las rutas básicas de prueba.

Paso 2: Configuración Básica de Slim

Ejemplo (**public/index.php**):

```
require __DIR__ . '/../vendor/autoload.php';

use Slim\Factory\AppFactory;
$app = AppFactory::create();
// Middleware de ruteo
$app->addRoutingMiddleware();
$app->get('/', function ($request, $response, $args) {
    $response->getBody()->write("¡Hola, CRUD Slim!");
    return $response;
});
$app->run();
```

Paso 3: Conexión a la Base de Datos

- Usaremos **Eloquent ORM** o **PDO** puro para la persistencia.
- **Recomendación:** Instalar un ORM ligero: `composer require illuminate/database`
- La conexión debe configurarse e inyectarse, típicamente a través de un **Container** (Contenedor de Dependencias).
- Esto mantiene la lógica de conexión separada del código de las rutas.

Configuración de la Conexión (ejemplo PDO):

- Definir las credenciales (`host`, `db`, `user`, `pass`).
- Crear un objeto de conexión **PDO** y pasarlo a las clases que lo necesiten.

Paso 4: Implementación de las Operaciones CRUD

Las operaciones CRUD se mapean a los métodos **HTTP** correspondientes:

1. **CREATE (Crear):**
   - Método: `POST`
   - Ruta: `/items`
   - Lógica: Recibe datos del cuerpo (`JSON`), inserta en la DB.
2. **READ (Leer/Obtener):**
   - Método: `GET`
   - Rutas: `/items` (todos) y `/items/{id}` (uno específico).
   - Lógica: Consulta la DB y retorna la lista/objeto en formato `JSON`.

Paso 4: Implementación de las Operaciones CRUD (Cont.)

3. **UPDATE (Actualizar):**
    - Método: `PUT` o `PATCH`
    - Ruta: `/items/{id}`
    - Lógica: Recibe datos, busca el registro por `id`, y actualiza los campos.
4. **DELETE (Borrar):**
    - Método: `DELETE`
    - Ruta: `/items/{id}`
    - Lógica: Busca el registro por `id` y lo elimina de la DB.

Estructura de la Ruta DELETE (Ejemplo):

```
$app->delete('/items/{id}', function ($request, $response, $args) {
    // Lógica para borrar el registro $args['id']
    // ...
    return $response->withStatus(204); // No Content
});
```

Paso 5: Organización del Código (Patrón MVC)

- Para proyectos más grandes, es crucial separar las responsabilidades.
- **Separación de Capas (Modelo, Vista, Controlador):**
  - **Controladores (Controllers):** Gestionan la lógica de la petición (`Request`) y la respuesta (`Response`).
  - **Modelos (Models):** Manejan la interacción con la Base de Datos.
  - **Vistas (Views):** (Opcional, si no es una API) Generan el HTML final.
- **Ventajas:** Código más legible, testeable y mantenible.

Conclusión

- Hemos sentado las bases para un **API CRUD** robusto usando:
  - **Composer** para la gestión de dependencias.
  - **Slim Framework** para el ruteo y manejo de peticiones.
- La metodología paso a paso permite una implementación limpia y modular.

Próximos Pasos:

- Implementación de **Validación de Datos**.
- Adición de **Autenticación** y **Autorización** (Tokens JWT).
- Despliegue en un servidor de producción.