

Manual Técnico de CodeIgniter 4

Arquitectura MVC, Enrutamiento Avanzado y Flujo de Trabajo Práctico

1. Introducción a CodeIgniter 4

CodeIgniter 4 es un framework de desarrollo de aplicaciones web en PHP robusto, rápido y con una huella (footprint) excepcionalmente pequeña. Diseñado para desarrolladores que necesitan una forma sencilla y elegante de crear aplicaciones web con todas las funciones, implementa de forma nativa el patrón de arquitectura **Modelo-Vista-Controlador (MVC)** y un sistema de enrutamiento flexible y seguro.

2. El Ciclo de Vida y Arquitectura MVC

El patrón MVC separa la lógica de negocio de la interfaz de usuario, facilitando el mantenimiento, la escalabilidad y las pruebas del software:

- **Modelo (Model):** Gestiona los datos de la aplicación, las reglas de negocio y la comunicación con la base de datos.
- **Vista (View):** Representa la capa de presentación. Es el código HTML/CSS/JS final que se renderiza en el navegador del usuario.
- **Controlador (Controller):** Actúa como intermediario. Recibe la petición del usuario a través de las rutas, invoca al modelo para procesar datos y selecciona la vista adecuada para mostrar la respuesta.

Estructura del Directorio de Trabajo (Carpeta /app)

Config/: Configuración de la aplicación (incluye Routes.php).

Controllers/: Clases controladoras que manejan el flujo.

Models/: Clases que interactúan con las tablas de la base de datos.

Views/: Archivos de plantilla HTML y código embebido PHP.

3. Configuración del Sistema de Rutas

En CodeIgniter 4, el enrutamiento automático (Auto-Routing) está desactivado por defecto por razones de seguridad y rendimiento. Esto significa que cada URL que la aplicación deba responder debe ser declarada explícitamente en el archivo de configuración de rutas ubicado en:

app/Config/Routes.php

3.1. Sintaxis Básica de una Ruta

La estructura general para definir una ruta mapea un método HTTP y un patrón de URL hacia un Controlador y un método específico:

```
$routes->get('url-visible', 'NombreControlador::metodo');
```

3.2. Rutas con Parámetros Dinámicos

Para capturar segmentos de la URL y pasarlos como argumentos a los métodos del controlador, utilizamos marcadores de posición (placeholders):

- (:num): Empareja cualquier segmento que sea estrictamente numérico.
- (:any): Empareja cualquier secuencia de caracteres permitidos en una URL.

Definición de la Ruta	Ejemplo de URL Real	Mapeo Interno
<code>\$routes->get('productos', 'Productos::index');</code>	<code>/productos</code>	Controlador: Productos, Método: <code>index()</code>
<code>\$routes->get('producto/(:num)', 'Productos::detalle/\$1');</code>	<code>/producto/125</code>	Controlador: Productos, Método: <code>detalle(125)</code>
<code>\$routes->get('categoria/(:any)', 'Productos::categoria/\$1');</code>	<code>/categoria/electronica</code>	Controlador: Productos, Método: <code>categoria('electronica')</code>

4. Ejemplo Práctico: Gestión de un Catálogo de Productos

A continuación, se detalla la implementación completa paso a paso de un módulo de productos utilizando Modelos, Vistas, Controladores y Rutas personalizadas.

Paso 1: Configurar las Rutas (app/Config/Routes.php)

Abrimos el archivo de rutas y añadimos los siguientes endpoints al final del archivo:

```
// Ruta para listar todos los productos
$routes->get('catalogo', 'CatalogoController::index');

// Ruta para ver el detalle de un producto específico por ID numérico
$routes->get('catalogo/ver/(:num)', 'CatalogoController::verDetalle/$1');
```

Paso 2: Crear el Modelo (app/Models/ProductoModel.php)

El modelo interactúa con la base de datos. CodeIgniter 4 provee una clase base muy potente que simplifica los accesos CRUD básicos.

```

<?php

namespace App\Models;

use CodeIgniter\Model;

class ProductoModel extends Model
{
    protected $table      = 'productos';
    protected $primaryKey = 'id';

    // Campos de la tabla permitidos para inserción/actualización segura
    protected $allowedFields = ['nombre', 'descripcion', 'precio', 'stock'];

    // Método personalizado para emular datos si no hay base de datos conectada
    public function obtenerProductosFicticios()
    {
        return [
            ['id' => 1, 'nombre' => 'Laptop Pro 15', 'precio' => 1200.00, 'descripcion' =>
            'Ideal para desarrollo.'],
            ['id' => 2, 'nombre' => 'Monitor 4K 27', 'precio' => 350.00, 'descripcion' =>
            'Panel IPS de alta definición.'],
            ['id' => 3, 'nombre' => 'Teclado Mecánico RGB', 'precio' => 95.00,
            'descripcion' => 'Switches mecánicos táctiles.'],
        ];
    }
}

```

Paso 3: Crear el Controlador (app/Controllers/CatalogoController.php)

El controlador se encarga de recibir los parámetros de la ruta, consultar al modelo y pasar los datos filtrados a la vista correspondiente.

```

<?php

namespace App\Controllers;

use App\Models\ProductoModel;

class CatalogoController extends BaseController
{
    public function index()
    {
        $model = new ProductoModel();

        // Obtenemos los datos (puedes usar $model->findAll() si tienes BD configurada)
        $data['titulo'] = 'Nuestro Catálogo de Productos';
        $data['productos'] = $model->obtenerProductosFicticios();

        // Cargamos la vista pasándole el arreglo de datos
        return view('catalogo/lista_view', $data);
    }

    public function verDetalle($id)
    {
        $model = new ProductoModel();
        $productos = $model->obtenerProductosFicticios();

        // Buscamos el producto coincidente con el ID capturado por la ruta
        $productoEncontrado = null;
        foreach ($productos as $prod) {
            if ($prod['id'] == $id) {
                $productoEncontrado = $prod;
                break;
            }
        }

        // Si no existe el producto, lanzamos un error 404 de página no encontrada
        if ($productoEncontrado === null) {
            throw \CodeIgniter\Exceptions\PageNotFoundException::forPageNotFound("El
producto con ID $id no existe.");
        }

        $data['producto'] = $productoEncontrado;
        return view('catalogo/detalle_view', $data);
    }
}

```

Paso 4: Crear las Vistas (HTML + PHP)

Las vistas reciben el arreglo de variables mapeadas de forma asociativa directamente como variables independientes (ej. \$titulo, \$productos).

Vista de Lista: Guardar en app/Views/catalogo/lista_view.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title><?= esc($titulo) ?></title>
</head>
<body>
    <h1><?= esc($titulo) ?></h1>
    <ul>
        <?php foreach ($productos as $item): ?>
            <li>
                <strong><?= esc($item['nombre']) ?></strong> - $<?= esc($item['precio']) ?
                <br>
                | <a href="<?= base_url('catalogo/ver/' . $item['id']) ?>">Ver Detalles</a>
            </li>
        <?php endforeach; ?>
    </ul>
</body>
</html>
```

Vista de Detalle: Guardar en app/Views/catalogo/detalle_view.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title><?= esc($producto['nombre']) ?></title>
</head>
<body>
    <h1><?= esc($producto['nombre']) ?></h1>
    <p><strong>Precio:</strong> $<?= esc($producto['precio']) ?></p>
    <p><strong>Descripción:</strong> <?= esc($producto['descripcion']) ?></p>
    <hr>
    <a href="<?= base_url('catalogo') ?>">← Volver al catálogo</a>
</body>
</html>
```

Buenas Prácticas de Seguridad Incorporadas

Siempre use la función global `esc()` al renderizar datos dinámicos dentro de elementos HTML. Esto previene vulnerabilidades de Cross-Site Scripting (XSS) al sanitizar los caracteres especiales de manera automática.

5. Resumen del Flujo de Ejecución

Cuando un cliente web solicita una URL en producción, el framework opera bajo la siguiente secuencia matemática y lógica de resolución:

Cliente ightarrow URL ightarrow Routes.php ightarrow Controller::method() ightarrow Model ightarrow View ightarrow Respuesta HTML

1. El usuario accede a `http://tuservidor.com/catalogo/ver/2`.
2. El sistema de **Rutas** captura el segmento `catalogo/ver/2`, verifica que el valor 2 cumple la regla numérico (: num) y delega el control a `CatalogoController` inyectando el valor 2 al argumento.
3. El **Controlador** instancia al `ProductoModel` para recuperar los datos específicos del ítem correspondiente.
4. El Controlador procesa los resultados y los despacha a la **Vista** estructurada.
5. La vista genera el flujo de salida HTML limpio devuelto de vuelta al navegador.