Explicación de la Implementación de un Framework MVC Similar a CodeIgniter

Carlos Arturo Moreno Susatama
Universidad XYZ
carlos.moreno@xyz.edu

3 de diciembre de 2024

Resumen

Este documento describe el funcionamiento y la estructura de un framework MVC básico, similar a CodeIgniter. El propósito de este framework es separar la lógica de la aplicación, haciendo que la gestión de los controladores, vistas y modelos sea más eficiente. En este trabajo se detallan las carpetas del framework y cómo interactúan entre sí para crear una aplicación web escalable.

1. Introducción

Un **framework MVC** es un conjunto de herramientas que permite la creación de aplicaciones web siguiendo el patrón de diseño Modelo-Vista-Controlador (MVC). Este patrón ayuda a separar la lógica de la aplicación en tres componentes: el Modelo (que gestiona la base de datos), la Vista (que maneja la presentación) y el Controlador (que recibe las peticiones del usuario y coordina la interacción entre el Modelo y la Vista).

Uno de los frameworks más populares que sigue este patrón es **CodeIgniter**. CodeIgniter es un framework PHP ligero y rápido que ofrece una estructura básica y fácil de usar para desarrollar aplicaciones web. En este trabajo, se presentará una implementación simplificada de un framework MVC, similar a CodeIgniter, detallando la función de cada carpeta y archivo dentro del sistema.

2. Estructura del Framework

La estructura del framework se organiza en carpetas que facilitan la separación de responsabilidades. A continuación, se explica la función de cada carpeta y los archivos dentro de ellas.

2.1. Carpeta `application`

La carpeta `application` contiene todos los componentes principales de la aplicación, como los controladores, modelos y vistas.

2.1.1. Controladores

Los controladores se encargan de recibir las peticiones del usuario y coordinar la interacción entre el modelo y la vista. Un ejemplo de un controlador es el archivo `UserController.php`, que maneja las peticiones relacionadas con la autenticación de usuarios.

Ejemplo de `UserController.php`:

```
class UserController extends Controller {
    public function login() {
        $this->load->view('login');
    }

    public function register() {
        $this->load->view('register');
    }
}
```

2.1.2. Modelos

Los modelos son responsables de interactuar con la base de datos. En este caso, el modelo `User.php` gestiona las consultas a la base de datos relacionadas con los usuarios.

Ejemplo de `User.php`:

```
class User extends Model {
    public function getUserData($id) {
        $query = $this->db->query("SELECT␣*␣FROM␣users␣WHERE␣id␣=␣?
            ", [$id]);
        return $query->fetch();
    }
}
```

2.1.3. Vistas

Las vistas son responsables de la presentación de la información al usuario. Por ejemplo, `login.php` muestra el formulario de inicio de sesión.

Ejemplo de `login.php`:

```
<form method="POST" action="/user/login">
    <input type="text" name="username" placeholder="Username"
        required>
    <input type="password" name="password" placeholder="Password"
        required>
    <button type="submit">Login</button>
</form>
```

2.2. Carpeta `config`

La carpeta `config` contiene archivos de configuración, como `database.php`, que establece la conexión con la base de datos.

Ejemplo de `database.php`:

```php
class Database {
    protected $user = "root";
    protected $pass = "";
    protected $dbname = "paraclase";
    protected $conn = null;

    public function __construct() {
        $this->conn  = new mysqli($this->url, $this->user, $this->pass, $this->dbname);
        if ($this->conn->connect_error) {
            die("Conexion fallida: " . $this->conn->connect_error);
        }
    }

    public function query($sql, $params = []) {
        $stmt = $this->conn->prepare($sql);
        if ($stmt === false) {
            die("Error en la preparaci n: " . $this->conn->error);
        }
        if ($params) {
            $stmt->bind_param(str_repeat('s', count($params)), ... $params);
        }
        $stmt->execute();
        return $stmt;
    }
}
```

2.3. Carpeta `public`

La carpeta `public` contiene los archivos accesibles públicamente, como imágenes, hojas de estilo (CSS) y scripts de JavaScript. Estos archivos se cargan en las vistas para mejorar la interacción con el usuario.

2.4. Carpeta `views`

La carpeta `views` contiene las plantillas HTML que el usuario verá. Dentro de esta carpeta se encuentran las vistas como `welcome.php` y `404.php`.

Ejemplo de `welcome.php`:

```php
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bienvenido</title>
</head>
<body>
    <h1>Bienvenido a Mi Framework</h1>
    <p>Tu framework est  configurado y listo para construir algo incre ble.</p>
    <a href="/user/login">Iniciar sesi n</a>
</body>
```

```
</html>
```

2.5. Carpeta `errors`

La carpeta `errors` contiene las vistas para manejar los errores. Por ejemplo, `404.php` se utiliza para mostrar una página de error cuando el usuario accede a una ruta inexistente.

Ejemplo de `404.php`:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>Error 404</title>
</head>
<body>
    <h1>Error 404: P gina no encontrada</h1>
    <p>La p gina que buscas no existe.</p>
</body>
</html>
```

3. Conclusión

Este framework MVC básico sigue el patrón Modelo-Vista-Controlador, separando las responsabilidades de cada componente para mejorar la organización y la escalabilidad de la aplicación. Al utilizar controladores para gestionar la lógica de negocio, modelos para interactuar con la base de datos y vistas para mostrar la interfaz de usuario, se facilita la mantenibilidad y extensión de la aplicación. Además, este enfoque modular y flexible es ideal para aplicaciones de cualquier tamaño.