Documentación Técnica: Refactorización a Arquitectura MVC

1. Introducción y Objetivos

Esta documentación detalla la reestructuración del proyecto "ProyConstruccionSoftware" hacia un patrón de diseño **Modelo-Vista-Controlador (MVC)**.

Objetivos de la refactorización:

- **Separación de Responsabilidades:** Desacoplar la lógica de negocio (PHP) de la interfaz de usuario (HTML/JS).
- **Seguridad:** Centralizar el punto de entrada (`index.php`) para evitar el acceso directo a scripts sensibles.
- **Mantenibilidad:** Organizar el código en capas lógicas para facilitar la corrección de errores y la adición de nuevas funcionalidades.
- **Escalabilidad:** Permitir que el equipo de desarrollo trabaje en paralelo (Backend vs Frontend) sin conflictos.

---

2. Estructura del Proyecto

La nueva estructura divide claramente el código de la aplicación (`app/`) de los archivos públicos accesibles por el navegador (`public/`).

```
/Code
├── app/ # LÓGICA PRIVADA (No accesible vía URL directa)
| ├── Config/ # Configuración global (BD, constantes)
```

```
| ├── Controllers/ # Controladores: Manejan las peticiones
| | └── Api/ # Controladores específicos para respuestas JSON
| ├── Core/ # Núcleo del framework (Router, Database, etc.)
| ├── Models/ # Modelos: Acceso a datos y lógica de negocio
| └── Views/ # Vistas: Plantillas HTML/PHP
|
├── public/ # ACCESO PÚBLICO (Raíz del servidor web)
| ├── assets/ # Recursos estáticos (CSS, imágenes, librerías)
| ├── js/ # Lógica Frontend específica de cada vista
| ├── styles/ # Estilos CSS específicos
| ├── partials/ # Fragmentos HTML reutilizables (sidebar, header)
| ├── index.php # PUNTO DE ENTRADA ÚNICO
| └── .htaccess # Reglas de redirección para el Router
|
└── vendor/ # Dependencias de Composer (si aplica)
```

---

3. Componentes Clave

3.1. Punto de Entrada (`public/index.php`)

Es el único archivo PHP que el servidor web ejecuta directamente.

- Inicia la sesión (`session_start`).
- Carga el Autoloader (para usar clases sin `require` manuales).

- Instancia el `Router` y define las rutas válidas.

3.2. Enrutador (`app/Core/Router.php`)

Intercepta la URL solicitada y decide qué Controlador ejecutar.

- **Web Routes:** Devuelven vistas HTML (ej. `/dashboard -> DashboardController::index`).
- **API Routes:** Devuelven datos JSON (ej. `/api/materias -> Api\MateriaController::index`).

3.3. Controladores (`app/Controllers/`)

Son los “directores de orquesta”. Reciben la petición, llaman al Modelo, y devuelven una respuesta.

- **Base:** `AuthController` , `DashboardController` .
- **API:** `Api\ActividadController` , `Api\MateriaController` .

3.4. Modelos (`app/Models/`)

Contienen la lógica “dura” y el acceso a la base de datos.

- Ejemplo: `Calculadora.php` contiene toda la lógica matemática de ponderaciones.
- Ejemplo: `Actividad.php` contiene las consultas SQL para leer/guardar actividades.

---

4. Guía para Desarrolladores Backend

¿Cómo crear una nueva funcionalidad API?

1. **Crear el Modelo (si es necesario):**

- Archivo: `app/Models/NuevaEntidad.php`
- Clase: `class NuevaEntidad { ... }`
- Debe contener métodos para interactuar con la BD (`obtenerTodos` , `guardar` , etc.).

2. **Crear el Controlador API:**

- Archivo: `app/Controllers/Api/NuevaEntidadController.php`
- Clase: `class NuevaEntidadController extends Controller { ... }`
- Métodos típicos: `index()` (GET), `store()` (POST), `delete()` (DELETE).
- **Importante:** Usar `$this->json(['status' => 'success', ...])` para responder.

3. **Registrar la Ruta:**

- Archivo: `public/index.php`
- Agregar: `$router->get('/api/nueva-entidad', [App\Controllers\Api\NuevaEntidadController::class, 'index']);`

Convenciones

- **Nombres de Clases:** PascalCase (ej. `ActividadController`).
- **Métodos:** camelCase (ej. `obtenerPorMateria`).
- **Retorno API:** Siempre devolver JSON con estructura estándar:

```
{
"status": "success"  |  "error",
"data": { ... },
"message": "Opcional"
```

```
}
```

---

5. Guía para Desarrolladores Frontend

¿Cómo crear una nueva vista?

1. **Crear el archivo de Vista:**

- Archivo: `app/Views/mi-nueva-vista.php`
- Estructura: HTML estándar.
- **Importante:** Usar la variable `$baseUrl` inyectada por PHP para cargar recursos:

```
<link rel="stylesheet" href="<?php echo $baseUrl; ?>assets/css/estilo.css

<script src="<?php echo $baseUrl; ?>js/mi-script.js"></script>
```

2. **Crear el JavaScript (Cliente):**

- Archivo: `public/js/mi-nueva-vista.js`
- Lógica: `fetch` a la API para obtener datos y manipular el DOM.

3. **Consumir la API:**

- Usar rutas relativas seguras o la variable global `BASE_URL`.
- Ejemplo seguro:

```
const baseUrl = globalThis.BASE_URL || '../';

const url = `${baseUrl}api/mi-entidad`;
```

```
fetch(url).then(...)
```

Estructura de Archivos Estáticos

- **CSS Global:** `public/assets/css/` (layout, sidebar, tablas).
- **CSS Específico:** `public/styles/` (estilos únicos de una vista).
- **JS Reutilizable:** `public/assets/js/` (sidebar.js, ui-helpers.js).
- **JS Específico:** `public/js/` (lógica de negocio de una vista).

---

6. Flujo de Trabajo Típico (Ejemplo: “Ver Detalles”)

1. **Usuario** navega a `/mis-calificaciones/detalle`.
2. **Router** detecta la ruta y llama a `CalificacionesController::detalle()`.
3. **Controlador** carga la vista `app/Views/mis-calificaciones-detalle.php`.
4. **Vista** se renderiza en el navegador (HTML vacío + Estructura).
5. **Navegador** carga `public/js/mis-calificaciones-detalle.js`.
6. **JS** hace una petición `fetch` a `/api/actividades`.
7. **Router** detecta `/api/actividades` y llama a `Api\ActividadController::index()`.
8. **Controlador API** llama a `Calculadora::obtenerMateriaConProgreso()`.
9. **Modelo** consulta la BD, calcula promedios y devuelve un array.
10. **Controlador API** devuelve el array como JSON.
11. **JS** recibe el JSON y actualiza el HTML (DOM) para mostrar los dato