📖 Guía del Proyecto MVC en PHP

Esta guía tiene como objetivo documentar todo lo necesario para que cualquier miembro del equipo pueda entender, instalar y trabajar en el proyecto.

✨ Tecnologías utilizadas

- PHP 8+
- MySQL (MariaDB)
- Composer
- Bootstrap 5 (vía CDN)
- Arquitectura MVC
- Router personalizado
- Controladores separados para Web y API
- Autoload PSR-4 con Composer

⚙️ Instalación del entorno

1. Clona el repositorio

```
git clone https://github.com/usuario/proyecto.git
cd proyecto
```

2. Instala las dependencias

```
composer install
```

3. Crea el archivo `.env`

```
DB_HOST=localhost
DB_NAME=mi_base
DB_USER=root
DB_PASS=
APP_ENV=local
```

4. Carga el archivo `.env` desde `index.php`

```
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();
```

5. Ejecuta `composer dump-autoload`

```
composer dump-autoload
```

🔷 Estructura del proyecto

```
project/
├── public/              ← Punto de entrada (index.php)
├── src/
│   ├── Controllers/
│   │   ├── Web/
│   │   └── Api/
│   ├── Models/
│   ├── Services/
│   ├── View.php         ← Motor de plantillas simple
│   ├── Database.php     ← Conexión PDO a MySQL
│   └── helpers.php      ← Funciones globales (jsonResponse, etc.)
├── views/
│   ├── layouts/
│   └── ...              ← Vistas por controlador
├── .env
```

```
├── composer.json
└── README.md
```

🔄 Router personalizado (MiniRouter)

- Se encuentra en `src/MiniRouter.php`
- Detecta si la ruta es `/api/...` para usar el namespace `App\Controllers\Api`
- Si no, usa `App\Controllers\Web`
- Devuelve JSON si `Accept: application/json` o si la ruta comienza con `/api`

✨ Vistas y layout

- Las vistas están en `/views`
- El layout principal está en `/views/layouts/main.php`
- Puedes pasar variables desde el controlador a la vista con `View::render()`
- El `content` de cada vista se inserta dentro del layout

💡 Models vs Services

Models

- Representan una entidad de base de datos
- Acceden directamente a la DB usando PDO
- Ejemplo: `User::find(1)`

Services

- Encapsulan la lógica de negocio
- Reutilizan Models
- Ejemplo: `UserService::getUserWithStatus($id)`

🏠 Controladores Web y API

- Web: `App\Controllers\Web\`
- API: `App\Controllers\Api\`

Ejemplo Web:

```
View::render('user/show', ['user' ⇒ $user]);
```

Ejemplo API:

```
jsonResponse(['data' ⇒ $user]);
```

💡 Tips importantes

- Ejecuta `composer dump-autoload` cada vez que agregues nuevas clases
- No subas el archivo `.env` al repositorio
- Usa `App\Services\...` para la lógica y `App\Models\...` para la DB
- Carga `helpers.php` desde `index.php`

🚀 Probar API desde JavaScript

```
fetch('/api/user/1', {
  headers: {
    'Accept': 'application/json'
  }
})
  .then(res ⇒ res.json())
  .then(data ⇒ console.log(data));
```

🚫 Manejo de errores 404

- Si la ruta es `/api/...` o espera JSON, se responde con:

```
{"error": "Ruta no encontrada"}
```

- Si es Web, se muestra un mensaje HTML 404.

✅ En resumen

> La estructura está pensada para separar responsabilidades, permitir pruebas fáciles, y mantener escalabilidad.

Para cualquier duda, revisar este documento o contactar al desarrollador principal del proyecto.