Plan de Migración: Frontend React a PHP Nativo (No-Framework)

Objetivo: Reemplazar el frontend React (SPA) por una arquitectura tradicional Server-Side Rendering (SSR) utilizando PHP puro, manteniendo la lógica de negocio existente.

Restricción: No se permite el uso de frameworks (Laravel, Symfony, React, Vue, etc.). Todo debe ser PHP estándar.

1. Análisis de Arquitectura

Estado Actual (Híbrido - A Eliminar)

- **Frontend:** React + Vite (Puerto 3001)
- **Backend:** PHP API REST (Puerto 8080)
- **Comunicación:** JSON vía HTTP (Axios)

Estado Deseado (Monolito PHP)

- **Frontend + Backend:** PHP único (Puerto 8080)
- **Patrón:** MVC (Model-View-Controller) simplificado.
- **Vistas:** Archivos `.php` en `public/views/` con HTML + PHP incrustado.
- **Controladores:** Nuevos controladores "Web" que renderizan HTML en lugar de JSON.
- **Sesiones:** `$_SESSION` nativa de PHP (reemplazando JWT en localStorage para la web).

2. Estrategia de Implementación

Fase 1: Estructura de Vistas (Scaffolding)

Crear la estructura de directorios para las vistas HTML:

```
hr-portal/
├── public/
│   └── views/
│       ├── layouts/       # Header, Footer, Navbar
│       ├── auth/          # Login
│       ├── dashboard/     # Panel principal
│       ├── employees/     # CRUD Empleados
│       └── errors/        # 404, 500
```

Fase 2: Adaptación de Rutas (Router)

Modificar `config/routes.php` para separar rutas de API (que mantendrá la App Móvil) de las rutas Web.

```
// Rutas Web (HTML)
$router->get('/login', [WebAuthController::class, 'loginForm']);
$router->post('/login', [WebAuthController::class, 'login']);
$router->get('/dashboard', [DashboardController::class, 'index']);

// Rutas API (JSON - Existentes)
$router->post('/api/auth/login', [ApiAuthController::class, 'login']);
```

Fase 3: Implementación de Controladores Web

Crear nuevos controladores en `src/Controllers/Web/` que orquesten la lógica y carguen las vistas.

- `WebAuthController`: Manejo de sesión y formulario de login.
- `DashboardController`: Vista principal.
- `WebEmployeeController`: Listado y gestión de empleados (HTML).

Diferencia Clave:

- `ApiController` -> Retorna `Response::json($data)`
- `WebController` -> Retorna `Response::html($viewPath, $data)`

Fase 4: Migración de Componentes React a PHP

Replicar la funcionalidad de las páginas React usando PHP/HTML:

Página React	Vista PHP	Funcionalidad
`LoginPage.jsx`	`views/auth/login.php`	Formulario POST, manejo de errores
`Dashboard.jsx`	`views/dashboard/index.php`	Resumen, gráficos (Chart.js via CDN si permitido)
`EmployeeList.jsx`	`views/employees/index.php`	Tabla HTML, paginación PHP
`EmployeeForm.jsx`	`views/employees/form.php`	Crear/Editar empleado

Fase 5: Manejo de Estado y Sesión

- **Login:** Al autenticar, guardar usuario en `$_SESSION`.

- **Middleware:** Crear un `WebAuthMiddleware` que verifique `isset($_SESSION['user_id'])` y redirija a `/login` si no.
- **Flash Messages:** Sistema simple para mostrar “Empleado guardado correctamente” tras redirección.

3. Plan de Ejecución

1. **Crear Directorios:** `mkdir -p public/views/{layouts,auth,dashboard,employees}`.
2. **Crear Layout Base:** `header.php` y `footer.php` con estilos CSS básicos (copiados de la maqueta actual).
3. **Controlador Web Base:** Clase abstracta para métodos comunes (`render()`, `redirect()`).
4. **Migrar Login:**
    - Ruta GET /login -> View form.
    - Ruta POST /login -> Validar -> Session -> Redirect Dashboard.
5. **Migrar Dashboard:** Ruta protegida que muestra el nombre del usuario.
6. **Migrar Empleados:** Listado simple conectado a la BBDD existente.
7. **Eliminación:** Borrar carpeta `web/` una vez validado el funcionamiento.

4. Estilos y Assets

- Mover CSS/imágenes de `web/src/assets` a `public/assets/`.
- Usar CSS plano o Bootstrap (vía CDN o local) para mantener diseño limpio sin build tools.

Autor: Gemini Agent **Fecha:** 24 Enero 2026