# Memoria Técnica del Proyecto DEW

## Introducción

El presente documento detalla el desarrollo técnico del proyecto DEW. El objetivo principal ha sido implementar una aplicación web para la gestión de registros y datos relacionados con el compostaje, utilizando JavaScript para el frontend en una arquitectura de SPA.

## Desarrollo de las vistas y funcionalidades

El proyecto incluye la creación de vistas dinámicas y funcionalidades esenciales para la gestión de datos desde el frontend. Las principales características desarrolladas son:

- Carga de datos desde una API REST utilizando funciones asíncronas con fetch.

* + Archivos como registros.js, ciclosBolos.js, y composteras.js incluyen funciones para consultar datos desde la API y presentarlos en tablas dinámicas.
  + Ejemplo: La función consulta(url) realiza llamadas a las rutas de la API con autenticación utilizando tokens.

- Generación dinámica de tablas y formularios con datos obtenidos de la API.

* + Funciones como generarTablaBolos() y generarTablasRegistrosCiclo() crean tablas interactivas con datos recibidos de la API.
  + Estas tablas incluyen enlaces clicables que permiten profundizar en registros específicos.
  + Se generan formularios dinámicos para el registro de datos en formulario.js.
  + Ejemplo: La función generarFormularioBolo() permite crear un nuevo bolo con datos como nombre y descripción.

- Navegación y paginación controladas mediante botones dinámicos.

* + Los botones btnAnterior y btnSiguiente permiten navegar entre páginas de datos, actualizando las vistas dinámicamente.
  + Ejemplo: manejarBotones(meta) controla el estado de los botones según la paginación.

- Modularización del código para mejorar su mantenimiento.

Estas funcionalidades están implementadas en archivos JavaScript como registros.js, ciclosBolos.js y composteras.js, siguiendo los principios de diseño modular y buenas prácticas.

**Relación con los contenidos de las primeras cuatro unidades de DEW:**

* **Unidad 1:** Uso de eventos en JavaScript (addEventListener) para manejar la interacción del usuario.
* **Unidad 2:** Manipulación del DOM para la creación dinámica de elementos HTML como tablas y formularios.
* **Unidad 3:** Peticiones HTTP utilizando fetch para la integración con la API.
* **Unidad 4:** Modularización del código en múltiples archivos JavaScript, mejorando la mantenibilidad.

## Aspecto visual final del front

El aspecto visual del proyecto se desarrolló utilizando Tailwind CSS, garantizando un diseño responsivo y atractivo. Los elementos clave del diseño incluyen:

- Formularios y tablas adaptables a dispositivos móviles.

* + Se utiliza Tailwind CSS para garantizar que los elementos, como tablas y formularios, sean visualmente atractivos y adaptables a diferentes tamaños de pantalla.
  + Ejemplo: Clases como max-w-md y overflow-x-auto aseguran que los formularios y tablas sean usables en dispositivos móviles.

- Botones interactivos con estados visuales (hover).

* + Las tablas incluyen enlaces interactivos y resaltados, mejorando la experiencia del usuario.
  + Formularios con validaciones visuales, como focus:outline-none, mejoran la accesibilidad.

- Colores y estilos consistentes, asegurando una experiencia de usuario agradable.

* + Los colores, fuentes y espaciados siguen un diseño limpio.
  + Botones como Siguiente, Atrás y Enviar tienen estados visuales que cambian al pasar el ratón (hover).

## Implementación de otras funcionalidades

Se han implementado diversas funcionalidades adicionales para enriquecer la experiencia del usuario y mejorar la seguridad del proyecto:

- Autenticación mediante tokens (Bearer Token) para proteger las rutas de la API.

* + La API utiliza Bearer Token para proteger las rutas.
  + Ejemplo: El sessionStorage almacena el token del usuario para su uso en todas las llamadas HTTP.

- Gestión de estados dinámicos, como la ocupación de composteras.

* + Se controlan estados como "compostera ocupada" para habilitar o deshabilitar botones.
  + Ejemplo: generarComposteras() verifica si una compostera está ocupada antes de permitir el registro.

- Lógica para la finalización automática de ciclos y bolos.

* + La lógica en insertarRegistros() permite cerrar ciclos y bolos automáticamente según las acciones realizadas.

- Edición de datos directamente desde las tablas generadas.

* + La aplicación permite editar datos directamente desde las tablas.

## Problemas encontrados

Durante el desarrollo, se enfrentaron los siguientes problemas y se implementaron las soluciones respectivas:

- Manejo de errores al consultar la API, como el manejo de respuestas 404 y errores de red.

- Ajustes en el diseño responsivo para garantizar la correcta visualización en todos los dispositivos.

- Coordinación de estados dinámicos entre múltiples vistas, resuelto mediante funciones centralizadas.

## Trabajo pendiente

Se identificaron las siguientes áreas de mejora para el futuro desarrollo del proyecto:

- Implementación de filtros avanzados para los registros.

- Optimización de las consultas a la API para mejorar el rendimiento.

- Creación de vistas adicionales para ampliar la funcionalidad del sistema.

## Instrucciones para desplegar y probar el proyecto

1. Clonar el repositorio desde GitHub.  
2. Instalar las dependencias necesarias con npm y composer.  
3. Configurar las variables de entorno en el archivo .env.  
4. Ejecutar las migraciones y seeders con php artisan migrate --seed.  
5. Iniciar el servidor local con php artisan serve.