**Documentación de Avance del Proyecto ERP Modular para PYMEs**

**Lenguaje y entorno**

* **Lenguaje principal**: PHP 8.3.14
* **Entorno local**: MAMP en macOS

**Fase 1: Estructura inicial y conexión a base de datos**

**Objetivo**

Establecer una arquitectura MVC simple en PHP sin frameworks.

**Acciones realizadas**

* Se creó index.php como punto de entrada.
* Se añadió core/Model.php y core/Controller.php como clases base.
* Se configuró la conexión PDO en config/database.php.
* Se preparó .htaccess para URLs limpias.

**Problemas encontrados**

* Ninguno relevante en esta fase.

**Fase 2: Módulo de autenticación (login/logout)**

**Objetivo**

Permitir el inicio y cierre de sesión con validación de credenciales contra la base de datos.

**Acciones realizadas**

* Se creó la tabla users.
* Se creó UserModel.php con método login() y verificación password\_verify().
* Se creó AuthController.php con métodos login() y logout().
* Se añadió vista app/views/auth/login.php.
* Se implementó validación de formulario en cliente (JS) y servidor (PHP).
* Se creó DashboardController.php con vista protegida para usuarios logueados.
* Se implementó helper core/Auth.php con la función requireLogin() reutilizable.
* Se reorganizó Model.php para usar la clase Database::connect().
* Se creó el script crear\_usuario.php para insertar usuarios con contraseña encriptada correctamente.

**Problemas encontrados y solución**

1. **Error “Controlador no encontrado”**:
   * ✔️ Solucionado: el archivo del controlador no estaba bien ubicado o tenía nombre incorrecto.
2. **Error en require() de la vista**:
   * ✔️ Solucionado: el sistema intentaba cargar app/views/login.php cuando debía ser app/views/auth/login.php. Se corrigió la ruta con \_\_DIR\_\_ . '/../app/views/$view.php' en el método view() del Controller.
3. **Faltaba protección de acceso tras login**:
   * ✔️ Solucionado: se creó un middleware simple para redirigir al login si no hay sesión.
4. **No había redirección tras login**:
   * ✔️ Solucionado: se añadió una pantalla dashboard.php como punto de acceso tras iniciar sesión.
5. **Falta validación de inputs**:
   * ✔️ Solucionado: se implementó validación tanto en PHP (servidor) como en JavaScript (cliente) para email y contraseña.
6. **Sin pantalla de destino tras login**:
   * ✔️ Solucionado: se creó un DashboardController y una vista simple protegida que saluda al usuario logueado.
7. **Se requería vista protegida post-login**:
   * ✔️ Solucionado: se creó DashboardController con protección por sesión. Si no hay sesión, redirige al login. Se generó la vista dashboard/index.php con saludo personalizado y opción de cerrar sesión.
8. **Protección repetida en controladores privados**:
   * ✔️ Solucionado: se creó el helper core/Auth.php con la función requireLogin(), que inicia sesión si es necesario y redirige al login si no hay usuario. Se usará al principio de todos los métodos que deban protegerse.
9. **Falta de cierre de sesión seguro**:
   * ✔️ Solucionado: se completó el método logout() en AuthController para destruir correctamente la sesión y redirigir al formulario de login.
10. **Login fallido con credenciales correctas**:

* ✔️ Solucionado: se detectó que las contraseñas estaban siendo insertadas sin password\_hash(). Se creó un script crear\_usuario.php para generar usuarios con contraseñas cifradas. Se reestructuró database.php como clase Database reutilizable. Tras esto, se validó el login correctamente con el usuario prueba1@gmail.com y la contraseña prueba.

**🧾 Actualización completa para la documentación**

Aquí tienes la sección **completa y ampliada** con el apartado de problemas y soluciones, como marca tu estructura:

**Módulo extra: Registro de usuarios desde el panel admin**

**Objetivo**

Permitir que los usuarios con rol admin puedan crear nuevos usuarios desde un formulario accesible tras login.

**Características implementadas**

* Formulario de creación de usuarios disponible solo para administradores.
* Acceso protegido mediante la función requireAdmin() definida en core/Auth.php.
* El formulario permite ingresar nombre, email, contraseña y seleccionar rol.
* Los roles se obtienen de forma **dinámica** desde la tabla users usando SELECT DISTINCT role, lo que permite un sistema escalable y autogestionado sin necesidad de tabla roles.
* Si la tabla users está vacía, se devuelven los roles por defecto ['admin', 'empleado', 'contable'].
* Inserción segura usando password\_hash() y validación básica de campos.

**Archivos modificados/creados**

* app/controllers/UserController.php: controlador con métodos create() y store().
* app/models/UserModel.php: se agregó getRoles() y createUser().
* app/views/users/create.php: formulario HTML para crear nuevos usuarios.
* core/Auth.php: se agregó función requireAdmin() reutilizable.

**Problemas encontrados y soluciones aplicadas**

* **❓ Replanteamiento del diseño inicial del sistema de roles**: inicialmente se propuso crear una tabla roles, pero se descartó para evitar mantenimiento extra y mejorar la escalabilidad.
  + ✅ **Solución aplicada**: UserModel::getRoles() lee los roles únicos directamente desde la tabla users, lo que hace el sistema más autónomo.
* ⚠️ No se detectaron errores técnicos en la implementación. Todo funcionó correctamente tras decidir la estrategia de carga dinámica.

**Resultado**

* El formulario ya está disponible para admins y funcional.
* El sistema es totalmente escalable: cualquier rol nuevo que exista en la base de datos aparecerá automáticamente.
* Los usuarios se crean con contraseñas hasheadas usando password\_hash().

**Estado: ✅ Completado**

**🧩 Mejora de arquitectura: enrutamiento dinámico en index.php**

**🎯 Objetivo**

Eliminar la necesidad de definir manualmente una acción por defecto en cada controlador, utilizando una solución genérica que detecta el primer método público del controlador si no se especifica action por URL.

**🛠️ Acciones realizadas**

* Se reemplazó el bloque de lógica del archivo index.php.
* Ahora:
  + Si action está presente en la URL, se ejecuta normalmente.
  + Si no se pasa action, se detecta automáticamente el **primer método público disponible** en el controlador cargado.
* Esto elimina la necesidad de definir propiedades como $defaultAction o de usar condicionales fijos (index, login, etc.).

**✅ Ventajas**

* 🔁 **Totalmente escalable**: funciona con cualquier cantidad de controladores.
* ♻️ **Reutilizable**: puedes aplicar este patrón en otros proyectos MVC fácilmente.
* 🧼 **Elimina duplicidad y hardcodeo**: sin ifs por controlador o método.
* 🧠 **Enrutamiento inteligente**: cada controlador define su flujo con su propio orden de métodos, sin configuración adicional.

**🧾 Estado: ✅ Completado y funcional**

**✅ Cierre del módulo: Registro de logs de acceso (login/logout)**

**🎯 Objetivo**

Registrar automáticamente en base de datos los accesos al sistema (inicios y cierres de sesión) por parte de los usuarios, con el objetivo de auditar la actividad básica del sistema ERP.

**🛠️ Acciones realizadas**

* Se creó la tabla logs con los campos:
  + user\_id (relación con usuarios)
  + action (login o logout)
  + ip\_address (IP del usuario)
  + created\_at (fecha y hora automática)
* Se implementó el modelo LogModel.php con la función add($userId, $action)
* Se integró el log:
  + Al iniciar sesión correctamente (login)
  + Al cerrar sesión correctamente (logout)
* Se ajustó el sistema de rutas para que la sesión estuviera disponible antes de destruirse.

**🐞 Problemas encontrados**

1. **Undefined index al cerrar sesión**  
   🔍 El sistema intentaba registrar el logout tras vaciar $\_SESSION, lo que causaba un error.  
   ✅ Solución: se guardó el user\_id antes de vaciar la sesión.
2. **Error de lógica al registrar "logout" durante login**  
   🔍 Se insertaba incorrectamente la acción "logout" al iniciar sesión.  
   ✅ Solución: se corrigió la acción y se puso "login" en el sitio correcto.

**✅ Resultado final**

* Los accesos se registran correctamente en la tabla logs.
* Cada entrada incluye usuario, IP, acción y timestamp.
* Se asegura que las acciones se registran solo si hay sesión válida.

**📌 Estado: Completado y validado correctamente  
  
Acceso rápido a creación de usuarios desde el dashboard**

**-------------------------------------------------------**

**Objetivo:**

**---------**

**Mejorar la usabilidad permitiendo acceso directo a la vista de creación de usuarios desde el panel de control.**

**Requisitos:**

**-----------**

**- Solo usuarios con rol 'admin' deben ver este acceso.**

**Cambios realizados:**

**-------------------**

**- Se añadió un botón condicional en `app/views/dashboard/index.php` que verifica el rol del usuario antes de mostrarlo.**

**Problemas encontrados:**

**----------------------**

**Ninguno.**

**Soluciones aplicadas:**

**----------------------**

**- Verificación de rol mediante $\_SESSION['user']['role'].**

**- Renderizado condicional del botón según el rol.  
  
Controlador EmpresaController – versión dinámica**

**------------------------------------------------**

**Objetivo:**

**---------**

**Eliminar la dependencia de definir manualmente los campos de formularios en el controlador.**

**Cambios realizados:**

**-------------------**

**- Se integró `getColumnNames()` desde el modelo base para obtener los campos válidos de la tabla.**

**- En la acción `store()`, los datos se recogen dinámicamente del $\_POST con un bucle automático.**

**Ventajas:**

**---------**

**- Escalable a cualquier tabla con muchos campos.**

**- Reutilizable como plantilla base para otros controladores.**

**- Reduce errores de mantenimiento al cambiar columnas en BBDD.**

**Resultado:**

**----------**

**✅ Controlador limpio, dinámico y 100% modular.**

**Implementación del botón de edición en la vista de empresas**

**------------------------------------------------------------**

**Objetivo:**

**---------**

**Permitir que los administradores puedan editar cualquier empresa desde el listado principal, accediendo a la vista de edición mediante un enlace directo.**

**Cambios realizados:**

**-------------------**

**- Se añadió una columna "Acciones" en la vista `app/views/empresas/index.php`, visible solo para usuarios con rol 'admin'.**

**- Por cada empresa listada, se genera automáticamente un botón "✏️ Editar" que enlaza a:**

**index.php?controller=empresa&action=edit&id={ID}**

**Ventajas:**

**---------**

**- Mejora la navegabilidad y usabilidad para el flujo de edición.**

**- Es modular: se adapta automáticamente a cualquier entidad que incluya un campo ID.**

**- Respeta los permisos: solo los administradores pueden acceder a funciones críticas como editar.**

**Resultado:**

**----------**

**✅ Sistema preparado para conectar la acción de edición con el método `edit()` del controlador.**

**Módulo: Edición Inline en Listado de Empresas**

**--------------------------------------------**

**Objetivo:**

**---------**

**Permitir a los usuarios con privilegios modificar los registros de la tabla `empresas` directamente desde la vista de listado, sin necesidad de redirigir a otro formulario.**

**Cambios implementados:**

**-----------------------**

**1. \*\*Botón "✏️ Editar" por cada fila\*\***

**- Convierte cada celda editable en un `<input>` para modificar su valor directamente.**

**- Activa el modo edición por fila con JavaScript.**

**- Oculta el botón editar y muestra botones "💾 Guardar" y "❌ Cancelar".**

**2. \*\*Controlador `update\_inline()`\*\***

**- Se añadió al `EmpresaController`.**

**- Recibe datos por `POST` mediante `fetch()` y actualiza en base de datos usando el modelo dinámico.**

**3. \*\*Campo `id` no editable\*\***

**- El campo `id` se mantiene visible solo como texto.**

**- No se convierte en input ni se envía en los datos actualizados (más allá de usarse como clave).**

**4. \*\*Botón "❌ Cancelar"\*\***

**- Restaura los valores originales sin enviar nada al servidor.**

**- Revierte tanto el texto mostrado (`<span>`) como el valor dentro del input (`<input>`), evitando que se mantenga un valor no guardado.**

**Ventajas:**

**---------**

**- Edición rápida sin abandonar el contexto visual.**

**- Evita redirecciones y formularios innecesarios.**

**- Experiencia de usuario mejorada.**

**- Todo el sistema es modular, dinámico y reutilizable.**

**Resultado:**

**----------**

**✅ El módulo de edición inline para empresas está completamente operativo, sin errores visibles, con una UX fluida, y siguiendo los estándares del ERP profesional.**

**Validación de campos en edición inline**

**--------------------------------------**

**🧾 DOCUMENTACIÓN – Módulo Edición Inline Empresas (Fase 3)**

**🔹 Objetivo:**  
Permitir la edición directa (inline) de los datos de empresas listadas en el sistema, con validación en cliente, feedback visual y actualización dinámica en la base de datos sin recargar la página.

**🛠️ Implementación**

1. **Conversión a edición inline:**
   * Se reemplazó el sistema de edición mediante formulario por uno inline.
   * Al hacer clic en el botón ✏️, las celdas se convierten en campos <input>, permitiendo modificar los valores directamente sobre la tabla.
2. **Botones de acción por fila:**
   * ✏️ Editar: activa modo edición.
   * 💾 Guardar: valida y envía los cambios.
   * ❌ Cancelar: restaura los valores originales.
3. **Control de campos editables:**
   * El campo id no es editable y se excluye de la lógica de edición.
   * Se impide que campos vacíos sean enviados.
   * Se implementó validación de formato para campos de tipo email.
4. **Modificaciones técnicas:**
   * Se separó el JavaScript en archivos independientes (empresas.js y flash.js) dentro de public/js/.
   * Se añadieron scripts a la vista empresas/index.php.
   * Se creó un sistema de mensajes tipo "toast" en pantalla, sin necesidad de recargar.

**🐞 Problemas detectados y soluciones**

1. **❌ Problema:** El botón de editar dejó de funcionar tras mover el JS a archivo externo.  
   **✅ Solución:** Se identificó que los scripts no estaban en la carpeta correcta (js/). Se movieron a public/js/ y se corrigieron las rutas.
2. **❌ Problema:** El botón de guardar mostraba “error al conectar con el servidor”.  
   **✅ Solución:** Se detectó que la variable $data no se inicializaba correctamente en EmpresaController::update\_inline(). Se corrigió añadiendo la lectura de php://input.
3. **❌ Problema:** Al cancelar la edición, los valores antiguos no se restauraban correctamente.  
   **✅ Solución:** Se usó data-original para guardar temporalmente los valores y restaurarlos al cancelar.
4. **❌ Problema:** La validación de campos no funcionaba (se permitía enviar vacíos o emails mal formateados).  
   **✅ Solución:** Se añadió validación directamente en JS antes de enviar el fetch(). Si falla, se muestra el mensaje flash y no se envía la petición.

**✅ Resultado final**

* Edición inline funcional, intuitiva y visualmente clara.
* Sistema modular y reutilizable tanto en estructura JS como en PHP.
* Preparado para escalar fácilmente a otros módulos (usuarios, productos, etc).

## Cumple el estándar ERP de modularidad, dinamismo, validación y experiencia de usuario profesional. ✅ FUNCIÓN COMPLETADA: Búsqueda en vivo por columnas con AJAX + Paginación persistente

🛠 **Qué se ha hecho:**

* Se ha implementado un sistema de búsqueda por columna (filtros individuales en <thead>) en el módulo de empresas.
* Los filtros se generan dinámicamente según los campos de la tabla en la BBDD.
* Se ha añadido una funcionalidad en el modelo (EmpresaModel) que permite buscar con múltiples condiciones LIKE de forma segura, y cuenta los resultados paginados.
* Se creó un nuevo endpoint AJAX en el controlador (buscar\_ajax) que procesa los filtros y devuelve los resultados en formato JSON.
* El frontend (empresas.js) escucha automáticamente los cambios en los inputs y realiza peticiones AJAX para actualizar los resultados.
* Se añadió una variable global filtrosActivos en JS que guarda el estado de los filtros para que **la paginación no los borre al cambiar de página**.

🔐 **Seguridad y buenas prácticas:**

* Solo se permite filtrar por columnas que están explícitamente validadas en getCamposPermitidos().
* Se usan prepared statements en todas las consultas SQL.
* Se valida que las peticiones AJAX sean de tipo POST y con datos JSON correctamente estructurados.
* Protección de acceso mediante requireLogin().

🧩 **Ventajas del sistema:**

* Altamente escalable: no se requiere redefinir campos si cambian en BBDD.
* Reutilizable: puede aplicarse el mismo patrón a otros módulos del ERP.
* Experiencia de usuario fluida y moderna, sin recargas de página.
* Compatible con la edición inline y otras funciones existentes.

✅ **Resultado final:**

* Búsqueda funcional por múltiples columnas al instante.
* Tabla y paginación que se adaptan a los filtros activos.
* Rendimiento optimizado con AJAX y estructura limpia.

## ✅ FUNCIÓN COMPLETADA: Exportación de empresas a Excel (formato .xls, sin librerías externas)

🛠 **Qué se ha hecho:**

* Se añadió un botón "Exportar a Excel" en la vista de empresas (index.php), que permite exportar los registros actualmente filtrados.
* Se reutilizan los filtros activos del buscador por columnas sin duplicar inputs.
* En JavaScript (empresas.js), se intercepta el envío del formulario y se extraen dinámicamente los filtros del <thead>.
* El backend (EmpresaController::exportar\_excel) recibe los filtros por POST, obtiene los datos con EmpresaModel::buscarEmpresas(), y genera un archivo Excel simulado con cabeceras Content-Type: application/vnd.ms-excel.

🧠 **Detalles técnicos:**

* El archivo se genera como una tabla HTML (<table>) con extensión .xls, que es totalmente compatible con Excel y LibreOffice.
* No se requiere Composer ni ninguna librería externa como PhpSpreadsheet.
* La exportación respeta los filtros activos, por lo que puede exportar el total o subconjuntos filtrados.

🔐 **Seguridad y robustez:**

* La acción solo permite peticiones POST y requiere login.
* Se usa htmlspecialchars() para evitar inyecciones en las celdas.
* Se corrigió un bug que provocaba un TypeError cuando no se enviaban filtros (null): ahora se convierte a un array vacío por defecto ($filtros ?? []), asegurando la compatibilidad con la firma del método.

✅ **Resultado:**

* Ligero, rápido, seguro y listo para entornos productivos.
* 100% escalable y reutilizable en otros módulos del ERP.

✅ FUNCIÓN AÑADIDA: Botón de acceso a módulo de empresas desde el Dashboard

🛠 Qué se ha hecho:

- Se añadió un botón “Ir a Gestión de Empresas” en la vista del dashboard (`dashboard/index.php`).

- El botón está disponible solo para usuarios con rol `admin`, igual que el de crear usuarios.

- Utiliza clases Bootstrap para un estilo coherente con el resto del sistema.

🎯 Qué permite:

- Acceder directamente al listado de empresas desde el panel tras iniciar sesión.

- Mejora la navegación y la productividad del usuario administrador.

✅ Cumple con: modularidad, legibilidad, reutilización y experiencia de usuario.

**🛠️ BUGFIX APLICADO: Restauración del botón “Guardar cambios” y protección de eventos**

🔧 **Problema detectado:**

* El botón de “Guardar” en la edición inline (.btn-guardar) lanzaba un error en consola:

csharp

CopiarEditar

Uncaught ReferenceError: guardarCambios is not defined

Esto ocurría porque la lógica de guardado fue separada pero no encapsulada en una función.

* Además, el formulario de exportar (form-exportar) generaba:

javascript

CopiarEditar

Uncaught TypeError: Cannot read properties of null (reading 'addEventListener')

cuando se accedía a una vista que no lo contenía (por ejemplo, el dashboard).

✅ **Qué se ha hecho:**

1. **Modularización de la lógica de guardado:**
   * Se extrajo la lógica original del addEventListener de .btn-guardar y se encapsuló en una función global reutilizable:  
     function guardarCambios(btn) { ... }
   * Esto resolvió el error y mejoró la legibilidad y reutilización del código.
2. **Protección del addEventListener del formulario de exportar:**
   * Se añadió una comprobación previa con if (formExportar) antes de llamar a .addEventListener, para evitar errores en vistas donde el formulario no existe.

🔐 **Resultado:**

* El sistema de edición inline vuelve a funcionar correctamente.
* El archivo empresas.js ahora es más modular, seguro y escalable.
* Se evita que errores JS rompan funcionalidades en otras vistas del ERP.

✅ Con esto, todo el bloque de edición, filtro, paginación y exportación está sólido.

### 📄 DOCUMENTACIÓN MÓDULO EMPRESAS — Soft Delete y Correcciones JS

#### ✅ Funcionalidad añadida: Eliminación lógica (Soft Delete)

**Objetivo**: Permitir eliminar empresas sin borrarlas físicamente de la base de datos, utilizando el campo eliminado para ocultarlas del listado por defecto, y ofreciendo la opción de verlas.

**Cambios realizados**:

* En el modelo EmpresaModel.php:
  + Se añadió el método softDelete($id) que marca una empresa como eliminada (eliminado = 1).
  + Se adaptaron los métodos getAllPaginado(), contarEmpresas(), buscarEmpresas() y contarEmpresasFiltradas() para tener en cuenta el parámetro $mostrarEliminadas.
* En el controlador EmpresaController.php:
  + Se añadió la acción softDelete() para recibir la petición AJAX de borrado lógico.
  + Se actualizó paginado\_ajax() y buscar\_ajax() para respetar el flag mostrarEliminadas.
* En la vista empresas/index.php:
  + Se incluyó un botón 👁️ Ver eliminadas que alterna entre ver empresas activas y eliminadas.
  + Se añadió un botón de eliminación 🗑️ por cada fila de empresa.

#### ⚙️ Correcciones y mejoras en empresas.js

**Problemas detectados y solucionados**:

1. **🔴 Botón eliminar no funcionaba**:
   * Causa: Los eventos addEventListener no se reasignaban después de la carga dinámica vía AJAX.
   * Solución: Centralizamos los eventos en la función asignarEventosEmpresas() y nos aseguramos de llamarla tras cada recarga de tabla.
2. **🔴 El botón eliminar no mostraba confirm() ni ejecutaba fetch()**:
   * Causa: asignarEventosEmpresas() no se ejecutaba tras DOMContentLoaded, y el selector de .btn-eliminar no encontraba elementos porque se cargaban dinámicamente.
   * Solución: Se garantizó que asignarEventosEmpresas() se ejecute cada vez que se regeneran filas.
3. **🟡 Problema potencial con múltiples bloques DOMContentLoaded**:
   * Se unificó la lógica de arranque dentro de la función inicializarEmpresas() para centralizar toda la inicialización JS.
4. **🟢 Mejora de mantenibilidad**:
   * Se encapsularon las funcionalidades de edición, guardado, cancelación, paginación, filtros y soft delete en funciones reutilizables y ordenadas.

#### 🧪 Verificación final

* ✅ El botón de eliminar aparece correctamente junto a cada empresa.
* ✅ Al hacer clic en "🗑️", se muestra el mensaje de confirmación.
* ✅ Si se confirma, se realiza la petición fetch() correctamente.
* ✅ El registro se marca como eliminado (eliminado = 1) en la BBDD.
* ✅ El botón 👁️ Ver eliminadas permite alternar entre empresas activas y eliminadas.
* ✅ Los filtros y la paginación funcionan con ambas vistas (activas o eliminadas).
* ✅ El mensaje visual mostrarMensaje() confirma el éxito o fallo de la operación.

### ✅ Módulo: Recuperación de Empresas Eliminadas (Soft Delete + Restaurar)

**Objetivo:**  
Permitir a los administradores restaurar empresas previamente eliminadas mediante el sistema de soft delete, con control visual desde la tabla de gestión de empresas y backend protegido.

#### 🧩 Funcionalidad implementada

* **Soft Delete**: al hacer clic en el botón 🗑️, la empresa no se elimina de la BBDD, sino que se marca con el campo eliminado = 1.
* **Filtro de visualización**: botón toggle que permite cambiar entre empresas activas y eliminadas (mostrarEliminadas).
* **Restauración de empresas**: si se visualizan eliminadas, aparece el botón ♻️ "Restaurar", que permite revertir el soft delete (eliminado = 0).

#### 🛠️ Cambios en el sistema

1. **Modelo (EmpresaModel)**:
   * Se añadieron los métodos softDelete($id) y restaurar($id).
   * Se adaptaron los métodos de paginación y búsqueda para contemplar mostrarEliminadas.
2. **Controlador (EmpresaController)**:
   * Métodos softDelete() y restaurar() reciben POST en JSON.
   * Se actualizó paginado\_ajax() y buscar\_ajax() para incluir/eliminar registros según mostrarEliminadas.
3. **Vista (empresas/index.php)**:
   * Se añadió un botón con id="toggle-eliminadas" para alternar la vista.
   * Se adaptó el frontend para visualizar registros eliminados.
4. **JS (empresas.js)**:
   * Se integró la variable mostrarEliminadas globalmente.
   * Se adaptó actualizarTablaEmpresas() para mostrar un botón u otro según el estado.
   * Se implementó la función de restauración vía fetch y btn-restaurar.

#### ⚠️ Problemas encontrados y soluciones aplicadas

1. **🔁 El botón "Restaurar" no se mostraba correctamente**  
   **Causa:** el valor de mostrarEliminadas se cambiaba tras pulsar el botón toggle, pero no se usaba adecuadamente en empresas.js.  
   **Solución:** se declaró mostrarEliminadas como variable global y se utilizó directamente en actualizarTablaEmpresas() para condicionar los botones.
2. **🧠 El evento del botón "Restaurar" no funcionaba tras recargar la tabla**  
   **Causa:** los botones se generaban dinámicamente con innerHTML, pero no se reasignaban los listeners.  
   **Solución:** se centralizó toda la lógica de eventos en asignarEventosEmpresas() y se llamó a esta función siempre después de actualizarTablaEmpresas().
3. **🌀 Caché del navegador impidiendo ver cambios recientes en JS**  
   **Causa:** el navegador servía una versión antigua del archivo empresas.js.  
   **Solución:** limpieza manual de la caché y uso de versionado en el <script src="empresas.js?v=2"> para evitar futuros problemas.

#### 🧪 Verificación

* Se testearon los siguientes flujos:
  + Eliminar una empresa ➝ no se borra, pasa a eliminado = 1.
  + Pulsar "👁️ Ver eliminadas" ➝ aparece el listado correcto.
  + Pulsar "♻️ Restaurar" ➝ la empresa vuelve a estar activa.
  + Cambio de vista y paginación funciona en ambos modos.

**🧾 Documentación – Mejora de estilo y usabilidad (Fase de Pulido)**

**🔧 Área: Mejora visual y experiencia de usuario (UX/UI)**

**🎯 Objetivo:**

Elevar el aspecto visual del ERP y mejorar la experiencia de uso mediante un estilo unificado, botones de navegación y componentes visuales consistentes en todas las vistas.

**✅ 1. Definición de estilo visual unificado**

* Se ha creado el archivo /public/css/estilos.css para aplicar un estilo global coherente y profesional.
* Se ha mantenido Bootstrap como base de diseño, añadiendo personalización avanzada mediante variables, colores definidos y ajustes visuales en:
  + Tipografía general (Segoe UI, Roboto)
  + Botones (btn-primary, btn-outline-\*, btn-danger)
  + Tablas (table, thead, filas editables)
  + Formularios (input, select, textarea)
  + Mensajes flash (.alert-success, .alert-danger)
  + Contenedores (.card, .container)

**✅ 2. Mejora de la vista principal de empresas (empresas/index.php)**

* Se ha envuelto el contenido principal dentro de un <div class="container"> para centralizar y espaciar correctamente la vista.
* Se han aplicado clases de Bootstrap a:
  + Tabla (table table-bordered table-hover)
  + Inputs de búsqueda (form-control form-control-sm)
  + Botones de acción (crear, editar, eliminar, restaurar) con btn btn-sm y estilos visuales claros.
* Se ha estilizado la tabla editable inline para mejorar la legibilidad.
* Se ha colocado un botón **“⬅️ Volver al dashboard”** arriba para facilitar la navegación global.

**✅ 3. Mejora de la vista crear usuario (users/create.php)**

* El formulario se ha envuelto en una card con padding (p-4) y sombra (shadow-sm) para destacarlo del fondo.
* Se han reemplazado los input planos por form-control de Bootstrap.
* Se ha agregado también un botón **“⬅️ Volver al dashboard”** como en empresas.
* Se ha utilizado alert-danger para mostrar errores de forma visual.

**✅ 4. Mejora de la vista dashboard/index.php**

* Se ha reorganizado la vista del panel principal en una tarjeta visual (card p-5 text-center) con mensaje de bienvenida.
* Los botones de acción (Crear nuevo usuario, Gestión de empresas, Cerrar sesión) se han unificado con clases visuales (btn-lg, btn-outline-dark, btn-primary, btn-danger).
* Se ha dejado preparado el dashboard para recibir tarjetas KPI dinámicas en una fila (row con col-md-6), pero se implementarán más adelante.

**🛠️ Problemas encontrados y soluciones**

* ❌ Error 404 al cargar estilos.css:  
  ✅ Se solucionó ajustando correctamente la ruta según el entorno local (/TFG/ERP/public/css/estilos.css).
* ❌ Botones sin consistencia visual:  
  ✅ Se unificó el diseño de botones mediante Bootstrap y estilos.css.
* ❌ Formularios sin estructura clara:  
  ✅ Se reestructuraron con form-label, form-control y tarjetas Bootstrap.

**🧠 Conclusión**

Este bloque de mejoras ha convertido el ERP en una plataforma más limpia, profesional y accesible. El sistema ahora transmite robustez, claridad y facilidad de uso, y está listo para seguir creciendo con KPIs, pedidos, facturas o importadores, manteniendo una base visual sólida.

## 🧾 DOCUMENTACIÓN – Mejora visual: Tarjetas KPI y Modo Oscuro

### 📅 Fecha: [03/06/2025]

### 🔧 Área: Interfaz de usuario – Dashboard y Estilo Global

### ✅ 1. Incorporación de tarjetas KPI dinámicas en el Dashboard

**Objetivo:** Mostrar datos clave del sistema en tiempo real, de forma visual, clara y moderna.

#### 🔹 Elementos implementados:

* Tarjeta con **Empresas activas**.
* Tarjeta con **Usuarios registrados**.
* Cada tarjeta muestra:
  + Icono representativo.
  + Título descriptivo.
  + Número actualizado desde BBDD.
* Valores extraídos dinámicamente desde los modelos EmpresaModel y UserModel.

#### 🔹 Detalles técnicos:

* Se usaron componentes Bootstrap (card, row, col) y clases personalizadas.
* Colores y bordes definidos con variables CSS (--color-primario, --color-exito, etc.).
* Animación de entrada con clase .fade-in.
* Responsive, accesible y fácilmente replicable para más KPIs (pedidos, ingresos, etc.).

#### ✅ Resultado:

* Interfaz profesional y comprensible al instante para el usuario.
* Totalmente integrado con la estructura modular y el estilo general del ERP.

### ✅ 2. Implementación de Modo Oscuro persistente

**Objetivo:** Ofrecer una experiencia más accesible, moderna y adaptable, con preferencia recordada entre sesiones.

#### 🔹 Funcionalidad añadida:

* Botón "🌓 Modo oscuro" en layout general (layout.php).
* Al hacer clic, se añade/quita la clase modo-oscuro en <body>.
* La preferencia se guarda en localStorage, para mantenerse tras recargar o cerrar navegador.

#### 🔹 Cambios en estilos.css:

* Reestructuración de todo el CSS usando **variables (--color-\*)**.
* Se definieron dos bloques:
  + :root (modo claro)
  + body.modo-oscuro (modo oscuro)
* Se ajustaron: body, .container, .card, .table, input, th, td, .editable, etc.
* Se sobreescribió .text-muted y .btn-outline-dark para que respeten el modo oscuro.

#### 🔹 Estilos adicionales:

* Fondo gris oscuro (#121212) para modo oscuro.
* Tarjetas con color de fondo intermedio (#2a2a2a) para destacar sobre fondo general.
* Textos y bordes adaptativos, manteniendo contraste y legibilidad.

### 🛠️ Problemas encontrados y soluciones aplicadas:

| **Problema** | **Solución aplicada** |
| --- | --- |
| Textos text-muted no se veían en modo oscuro | Se sobrescribió .text-muted con --color-texto + opacity |
| Botones btn-outline-dark con texto negro | Se ajustó en modo oscuro para usar color claro y mantener contraste |
| Fondos fijos en tarjetas y tablas (#fff) | Reemplazados por var(--color-card) |
| Herencia incorrecta en texto de KPIs | Se unificó con color: var(--color-texto) globalmente |

### 🧠 Conclusión

Esta fase ha mejorado sustancialmente la presentación del ERP, dotándolo de:

* Un **dashboard profesional** con métricas visibles al instante.
* Un sistema de **modo oscuro real**, coherente y moderno.
* Un estilo global consistente, escalable y adaptable.

Ambas mejoras están alineadas con los principios del proyecto:  
**modularidad, escalabilidad, usabilidad y preparación para producción**.

### 🧱 Reflexión arquitectónica y decisión de implementación

Durante el desarrollo del ERP, y en especial al afrontar la implementación del módulo de pedidos y facturación, se ha identificado una posible mejora estructural: la creación de un controlador y modelo genéricos capaces de operar dinámicamente sobre cualquier tabla de la base de datos. Esta estrategia habría permitido un alto grado de reutilización de código, reduciendo la necesidad de generar múltiples controladores y modelos específicos por entidad (como EmpresaController, PedidoController, etc.).

No obstante, se ha decidido mantener la arquitectura actual para garantizar la coherencia con el sistema ya desarrollado, asegurar la estabilidad del proyecto y cumplir con los plazos establecidos. Esta decisión responde a criterios prácticos, sin comprometer la modularidad ni la escalabilidad del sistema, ya que la estructura MVC adoptada sigue permitiendo un desarrollo limpio y mantenible.

Se documenta este análisis como una **propuesta de mejora futura**, ideal para una versión 2.0 del ERP, en la que se podría introducir un núcleo genérico de control y persistencia que reduzca aún más el acoplamiento y potencie la reutilización en proyectos a gran escala.