Sprint 2 J-J-PROYECT

📋 Información del Proyecto

Autores:

- Jesús Ríos López
- Jaime Gavilán Torrero

Fecha: Sprint 2 - 28 Noviembre 2025
Proyecto: Mini ERP Modular SaaS

🎯 Objetivo del Proyecto

El proyecto J-J-PROYECT es un **sistema ERP (Enterprise Resource Planning) modular** diseñado para pequeñas y medianas empresas. Se trata de una aplicación web que permite gestionar diferentes áreas de negocio de forma integrada: facturación, inventario, control horario, gestión de vacaciones, CRM y más.

El objetivo principal de este primer sprint ha sido establecer las **bases del sistema**, implementando:

- La arquitectura general del proyecto
- La estructura de base de datos
- Los primeros módulos funcionales (Productos y Proveedores)
- La interfaz de usuario básica con navegación entre secciones

---

🏗️ Arquitectura del Sistema

El proyecto sigue una arquitectura de **3 capas** (Modelo-Vista-Controlador) y está diseñado para funcionar como una **Single Page Application (SPA)**:

Estructura del Proyecto

```
J-J-PROYECTO-INTERMODULAR/
├── backend/              # Lógica del servidor
│   ├── api/             # Endpoints de la API REST
│   ├── controlador/     # Controladores (lógica de negocio)
│   ├── modelo/          # Modelos (acceso a datos)
│   ├── config.php       # Configuración de la base de datos
│   ├── database.php     # Conexión a la base de datos
│   └── schema.sql       # Estructura de la base de datos
├── frontend/            # Interfaz de usuario
│   ├── js/             # JavaScript de la aplicación
│   ├── pages/          # Páginas HTML
│   ├── styles/         # Hojas de estilo CSS
│   └── index.html      # Punto de entrada
└── docker-compose.yml   # Configuración de contenedores
```

¿Cómo funciona?

1. **Frontend (Cliente)**: El usuario interactúa con la interfaz web construida con HTML, CSS y JavaScript vanilla. Es una aplicación de una sola página (SPA) que no recarga el navegador al cambiar de sección.
2. **Backend (Servidor)**: Procesa las peticiones del cliente mediante una API REST construida en PHP. Aplica la lógica de negocio y consulta la base de datos.
3. **Base de Datos**: Almacena toda la información del sistema (productos, proveedores, usuarios, etc.) en una base de datos MariaDB.

🛠️ Tecnologías Utilizadas

Backend

- **PHP 8.2**: Lenguaje de programación del servidor
- **MariaDB 11**: Sistema de gestión de base de datos
- **Apache**: Servidor web
- **Patrón MVC**: Organización del código en Modelo-Vista-Controlador
- **API REST**: Comunicación cliente-servidor mediante JSON

Frontend

- **HTML5**: Estructura de las páginas
- **CSS3**: Estilos y diseño visual
- **JavaScript (ES6+)**: Interactividad y lógica del cliente
- **SPA (Single Page Application)**: Navegación sin recargas de página

Infraestructura

- **Docker**: Contenedores para el entorno de desarrollo
- **Docker Compose**: Orquestación de servicios (web, base de datos, phpMyAdmin)

📊 Base de Datos

El sistema utiliza una base de datos relacional con las siguientes tablas principales:

Tabla `proveedor`

Almacena la información de los proveedores de productos:

```
CREATE TABLE proveedor (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    telefono VARCHAR(20),
    email VARCHAR(255),
    direccion VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Campos:

- `id`: Identificador único
- `nombre`: Nombre de la empresa proveedora
- `telefono`: Número de contacto
- `email`: Correo electrónico
- `direccion`: Dirección física
- `created_at`: Fecha de registro

Tabla `productos`

Gestiona el inventario de productos:

```
CREATE TABLE productos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    stock INT NOT NULL DEFAULT 0,
    precio DECIMAL(10, 2) NOT NULL DEFAULT 0.00,
    proveedor INT,
    ubicacionAlmacen VARCHAR(100),
    FOREIGN KEY (proveedor) REFERENCES proveedor(id)
);
```

Campos:

- `id`: Identificador único del producto
- `nombre`: Nombre descriptivo del producto
- `stock`: Cantidad disponible en almacén
- `precio`: Precio unitario del producto
- `proveedor`: Referencia al proveedor (relación con tabla `proveedor`)
- `ubicacionAlmacen`: Ubicación física en el almacén (ej: "A-01", "B-15")

Relación: Un proveedor puede suministrar muchos productos (relación 1 a N).

🔧 Backend: Arquitectura MVC

El backend está organizado siguiendo el patrón **Modelo-Vista-Controlador (MVC)**, separando las responsabilidades:

1. Modelo (`modelo/productos_modelo.php`)

El modelo se encarga de **interactuar con la base de datos**. Define las operaciones de acceso a datos sin preocuparse de la lógica de negocio.

```
class ProductosModelo {
    private PDO $db;
```

```
    public function __construct($db) {
        $this->db = $db;
    }

    // Obtiene productos con paginación
    public function obtenerProductos(int $limit = 10, int $offset = 0): array {
        $sql = "SELECT * FROM productos LIMIT :limit OFFSET :offset";
        $stmt = $this->db->prepare($sql);
        $stmt->bindParam(':limit', $limit, PDO::PARAM_INT);
        $stmt->bindParam(':offset', $offset, PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetchAll();
    }

    // Cuenta el total de productos
    public function contarProductos(): int {
        $sql = "SELECT COUNT(*) as total FROM productos";
        $stmt = $this->db->prepare($sql);
        $stmt->execute();
        return (int) $stmt->fetch()['total'];
    }
}
```

Responsabilidades:

- Ejecutar consultas SQL
- Devolver datos en formato array
- No contiene lógica de negocio

2. Controlador (`controlador/productos_controlador.php`)

El controlador aplica la **lógica de negocio**. Procesa los datos del modelo antes de enviarlos al cliente.

Ejemplo de función:

```
public function listarProductos(int $limit = 10, int $offset = 0): array {
    $productos = $this->modelo->obtenerProductos($limit, $offset);
    $total = $this->modelo->contarProductos();

    return [
        'data' => $productos,
        'pagination' => [
            'page' => floor($offset / $limit) + 1,
            'limit' => $limit,
            'total' => $total,
            'totalPages' => ceil($total / $limit),
            'offset' => $offset
        ]
    ];
}
```

Responsabilidades:

- Procesar datos del modelo
- Aplicar cálculos (como la paginación)
- Formatear respuestas para la API

3. API (`api/productos.php`)

La API es el **punto de entrada** de las peticiones HTTP. Define los endpoints y enruta las solicitudes al controlador adecuado.

```
header('Content-Type: application/json; charset=UTF-8');
header('Access-Control-Allow-Origin: *');

$method = $_SERVER['REQUEST_METHOD'];

if ($method == 'GET') {
    if (isset($_GET['id'])) {
        // Obtener producto específico
        $producto = $controlador->verProducto((int) $_GET['id']);
        echo json_encode($producto);
    } else {
        // Listar todos los productos con paginación
        $page = isset($_GET['page']) ? max(1, (int) $_GET['page']) : 1;
        $limit = isset($_GET['limit']) ? max(1, (int) $_GET['limit']) : 10;
        $offset = ($page - 1) * $limit;

        $result = $controlador->listarProductos($limit, $offset);
        echo json_encode($result);
    }
}
```

Características:

- Responde en formato JSON
- Permite CORS (Cross-Origin Resource Sharing) para desarrollo
- Soporta paginación mediante parámetros `page` y `limit`

🎨 Frontend: Single Page Application (SPA)

El frontend está construido como una **SPA (Single Page Application)**, lo que significa que la página no se recarga al navegar entre secciones. Todo se gestiona con JavaScript.

Router (`js/router.js`)

El router es el **sistema de navegación** de la aplicación. Permite cambiar de vista sin recargar la página.

```
class Router {
    constructor() {
        this.routes = {};
        this.currentRoute = null;
    }

    // Registrar una ruta
    register(path, handler) {
        this.routes[path] = handler;
    }

    // Navegar a una ruta
    navigate(path) {
        if (this.routes[path]) {
            this.currentRoute = path;
            window.location.hash = path;  // Actualiza la URL (#dashboard, #stock,
etc.)
            this.routes[path]();           // Ejecuta la función de la ruta
            this.updateActiveLink(path);   // Marca el enlace activo
        }
    }
}
```

¿Cómo funciona?

1. El usuario hace clic en un enlace (ej: "Stock")
2. El router captura el cambio de hash en la URL (`#stock`)
3. Ejecuta la función asociada a esa ruta
4. Actualiza el contenido de la página sin recargar

Gestión de Vistas (`js/app.js`)

Las vistas se definen como funciones que retornan código HTML:

```
const views = {
  stock: () => `
    <header>
      <h1>Stock / Inventario</h1>
      <p>Control de productos, existencias y movimientos.</p>
    </header>
    <section>
      <div id="productos-container">
        <!-- Aquí se carga la lista de productos -->
      </div>
    </section>
  `
};
```

Conexión con la API

El frontend consume la API mediante `fetch`:

```
async function loadProductos(page = 1, limit = 10) {
    const response = await fetch(
        `http://localhost:8080/backend/api/productos.php?
page=${page}&limit=${limit}`
    );
    const result = await response.json();
    const productos = result.data;

    // Renderizar la tabla de productos
    productos.forEach(producto => {
        // Crear filas de la tabla
    });
}
```

Proceso:

1. Se hace una petición HTTP GET a la API
2. La API devuelve JSON con los productos y datos de paginación
3. JavaScript genera dinámicamente el HTML de la tabla
4. Se insertan los botones de navegación (anterior/siguiente)

⚙ Funcionalidades Implementadas

1. Sistema de Navegación SPA

- Menú lateral con enlaces a diferentes módulos
- Cambio de vista sin recargar la página
- Indicador visual de la sección activa

2. Módulo de Productos

- **Listado paginado**: Muestra productos en tablas con paginación
- **Información mostrada**: ID, nombre, stock, precio, ubicación
- **Control de paginación**: Botones anterior/siguiente
- **Selector de resultados**: 10, 20 o 50 productos por página
- **Contador de registros**: "Mostrando 1-10 de 25"

3. Módulo de Proveedores

- **Listado con paginación**: Similar al de productos
- **Datos mostrados**: ID, nombre, teléfono, email, dirección
- **Carga bajo demanda**: Se cargan al hacer clic en un botón

4. Dashboard

- Vista principal con indicadores clave (KPIs)

- Resumen del estado de módulos activos
- Información de facturación, stock y control horario

5. Vistas de Módulos Futuros

- Plantillas preparadas para: Facturación, Control Horario, Vacaciones, CRM, Compras, Notificaciones
- Descripción de funcionalidades planificadas

Entorno de Desarrollo con Docker

El proyecto utiliza **Docker Compose** para crear un entorno de desarrollo consistente y fácil de configurar.

Servicios Configurados

1. Servicio Web (Apache + PHP)

```
web:
  image: php:8.2-apache
  ports:
    - "8080:80"
  volumes:
    - ./:/var/www/html
```

- **Puerto**: 8080
- **Extensiones**: mysqli, pdo, pdo_mysql (para base de datos)
- **mod_rewrite**: Activado para URLs amigables

2. Servicio de Base de Datos (MariaDB)

```
db:
  image: mariadb:11
  environment:
    - MARIADB_ROOT_PASSWORD=root
    - MARIADB_DATABASE=JJPROYECT
    - MARIADB_USER=app
    - MARIADB_PASSWORD=app
  ports:
    - "3307:3306"
```

- **Puerto**: 3307 (para evitar conflictos con MySQL local)
- **Base de datos**: JJPROYECT
- **Persistencia**: Volumen Docker para no perder datos

3. phpMyAdmin

```
phpmyadmin:
  image: phpmyadmin:5
  ports:
    - "8081:80"
```

- **Puerto**: 8081
- **Función**: Gestión visual de la base de datos

Comandos Útiles

Iniciar el proyecto:

```
docker-compose up -d
```

Ver logs:

```
docker-compose logs -f
```

Detener servicios:

```
docker-compose down
```

Acceder a la base de datos:

```
docker exec -it db-J-J-Proyect mysql -u root -p
```

Datos de Ejemplo

El sistema incluye **datos de prueba** para facilitar el desarrollo y las demostraciones:

Proveedores Precargados

- Herramientas Industriales S.L.
- Tornillería García Hermanos
- Distribuciones Eléctricas del Norte
- Pinturas y Barnices Colores S.A.
- Fontanería Pro Suministros

Productos Precargados (25 productos)

Ejemplo: Taladro percutor, Tornillos autorroscantes, Cable eléctrico, Pintura plástica, Tubo PVC, etc.

Estos datos simulan un **negocio de ferretería**, mostrando cómo el sistema puede gestionar diferentes categorías de productos con sus respectivos proveedores.

Características Destacadas

1. Paginación Eficiente

El sistema implementa paginación tanto en backend como en frontend:

Backend:

```
// Calcula automáticamente offset y total de páginas
$offset = ($page - 1) * $limit;
$totalPages = ceil($total / $limit);
```

Frontend:

```
// Control de navegación
<button data-action="prev" ${page <= 1 ? 'disabled' : ''}>Anterior</button>
<button data-action="next" ${page >= totalPages ? 'disabled' :
''}>Siguiente</button>
```

2. Código Modular y Reutilizable

Cada componente tiene una responsabilidad clara:

- **Modelos**: Solo acceso a datos
- **Controladores**: Solo lógica de negocio
- **API**: Solo enrutamiento
- **Frontend**: Separación entre router y vistas

3. Buenas Prácticas de Seguridad

- **Prepared Statements**: Previene inyección SQL
- **Validación de parámetros**: Comprueba tipos y rangos
- **CORS configurado**: Control de acceso desde el navegador

4. Diseño Responsive

La interfaz se adapta a diferentes tamaños de pantalla mediante CSS flexible.

Próximos Pasos (Sprint 2)

Para el siguiente sprint están planificadas las siguientes mejoras:

Funcionalidades

1. **CRUD completo**: Crear, editar y eliminar productos/proveedores
2. **Sistema de autenticación**: Login y gestión de usuarios
3. **Módulo de facturación**: Crear y gestionar facturas
4. **Control horario**: Fichaje de empleados
5. **Búsqueda y filtros**: Buscar productos por nombre, precio, etc.

Tecnologías

1. **Migración a frameworks modernos**:
    - **Frontend**: Angular para una arquitectura más robusta
    - **Backend**: Laravel para aprovechar su ecosistema y herramientas
2. **Mejoras técnicas**:
    - Autenticación JWT
    - Validación de formularios
    - Manejo de errores mejorado
    - Testing automatizado

🎓 Conclusiones del Sprint 2

Logros Alcanzados

☑ **Arquitectura sólida**: Base MVC bien estructurada
☑ **Frontend funcional**: SPA con navegación fluida
☑ **API REST operativa**: Endpoints de productos y proveedores
☑ **Base de datos diseñada**: Esquema relacional con datos de ejemplo
☑ **Entorno Docker**: Configuración reproducible y portable
☑ **Paginación completa**: Implementada en backend y frontend

Lecciones Aprendidas

- La **separación de responsabilidades** (MVC) facilita el mantenimiento
- Las **SPA mejoran la experiencia de usuario** al evitar recargas
- **Docker simplifica la configuración** del entorno de desarrollo
- La **paginación es esencial** para manejar grandes volúmenes de datos

Estado Actual del Proyecto

El proyecto cuenta con una **base funcional** sobre la cual se pueden construir el resto de módulos. La arquitectura elegida permite escalar fácilmente añadiendo nuevos endpoints, modelos y vistas sin modificar el código existente.

📌 Nota sobre Migración Futura

IMPORTANTE: En el futuro próximo, el proyecto migrará a tecnologías más modernas:

- **Frontend**: Se migrará de JavaScript vanilla a **Angular** para aprovechar:

    - Sistema de componentes reutilizables
    - TypeScript para mayor seguridad de tipos
    - Herramientas de desarrollo avanzadas
    - Gestión de estado más robusta

- **Backend**: Se migrará de PHP vanilla a **Laravel** para aprovechar:

    - Sistema de rutas y middleware avanzado
    - ORM Eloquent para gestión de base de datos
    - Sistema de autenticación integrado
    - Validación de datos más potente
    - Ecosistema de paquetes extenso

Esta migración permitirá acelerar el desarrollo y garantizar un código más mantenible y escalable a largo plazo.

📞 Contacto

Desarrolladores:

- Jesús Ríos López
- Jaime Gavilán Torrero

Proyecto Académico: Desarrollo de Aplicaciones Web
Curso: 2025-26

Documento generado en el Sprint 2 - Noviembre 2025