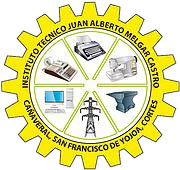
Instituto Gubernamental Técnico



“Juan Alberto Melgar Castro”

**Alumno**:

Dayana Michelle Paz Chávez

**Profesor**:

Lic. Redin José Méndez Meza

**Espacio Curricular:**

Diseño Web

**Trabajo:**

Manual Técnico

**Curso:**

III de BTP en Informática

**Lugar:**

Cañaveral, San francisco de Yojoa Cortes

**Fecha:**

17/06/2025

# Manual Técnico del Proyecto Suministros-Dayana

Este documento proporciona una guía técnica exhaustiva para comprender, configurar, mantener y extender la aplicación web Suministros-Dayana. Está diseñado para desarrolladores, administradores de sistemas y cualquier persona interesada en los detalles técnicos de la implementación.

## 1. Introducción

### 1.1. Propósito del Documento

El propósito de este manual es documentar la arquitectura, la configuración del entorno, la estructura del código, las funcionalidades clave y las consideraciones de seguridad del proyecto Suministros-Dayana. Sirve como una referencia integral para facilitar el desarrollo futuro, la depuración y el mantenimiento del sistema.

### 1.2. Alcance del Proyecto

Suministros-Dayana es una plataforma de e-commerce de productos tecnológicos que abarca las siguientes funcionalidades:

* **Gestión de Usuarios:** Registro, listado, edición y eliminación de usuarios.
* **Catálogo de Productos:** Visualización de productos con filtros y ordenamiento.
* **Mini-Carrito de Compras:** Funcionalidad de añadir, ajustar cantidad y eliminar productos de un carrito temporal basado en el navegador.
* **Proceso de Pago Simplificado:** Simulación de una página de pago que integra el total del carrito.
* **Navegación y Estructura:** Páginas informativas como blog, marcas, servicio técnico, etc.

### 1.3. Audiencia

Este manual está dirigido principalmente a:

* **Desarrolladores:** Para entender la lógica del código, la estructura y cómo realizar nuevas implementaciones o modificaciones.
* **Administradores de Sistemas:** Para configurar el entorno, desplegar la aplicación y solucionar problemas a nivel de servidor.
* **Analistas Técnicos:** Para comprender el flujo de datos y la interacción entre componentes.

## 2. Arquitectura del Sistema

### 2.1. Arquitectura de Alto Nivel (Cliente-Servidor)

El proyecto Suministros-Dayana sigue una arquitectura tradicional de tres capas:

1. **Capa de Presentación (Frontend):** Responsable de la interfaz de usuario. Construida con HTML, CSS y JavaScript, se ejecuta en el navegador del cliente.
2. **Capa de Lógica de Negocio (Backend):** Maneja las reglas de negocio, el procesamiento de datos y la comunicación con la base de datos. Implementada en PHP, se ejecuta en el servidor web (Apache).
3. **Capa de Datos:** Almacena y gestiona la información del sistema. Utiliza una base de datos MySQL.

La comunicación entre el frontend y el backend se realiza a través de solicitudes HTTP (GET, POST), donde el PHP procesa la solicitud, interactúa con la base de datos y genera una respuesta (generalmente HTML para renderizado directo o redirecciones).

### 2.2. Tecnologías y Stack

La siguiente tabla resume las principales tecnologías utilizadas en cada capa:

|  |  |  |
| --- | --- | --- |
| Capa | Tecnología / Lenguaje | Descripción |
| **Frontend** | HTML5 | Estructura y contenido de las páginas web. |
|  | CSS3 | Estilos y diseño visual de la interfaz de usuario. |
|  | JavaScript (ES6+) | Interactividad en el cliente (mini-carrito, filtros, validaciones). |
|  | Bootstrap 5 | Framework CSS para un diseño responsivo y componentes de UI predefinidos. |
|  | FontAwesome | Biblioteca de íconos vectoriales. |
| **Backend** | PHP | Lenguaje de scripting del lado del servidor para la lógica de negocio. |
|  | MySQLi | Extensión de PHP para interactuar de forma segura con la base de datos MySQL. |
| **Base de Datos** | MySQL / MariaDB | Sistema de gestión de bases de datos relacionales. |
| **Servidor Web** | Apache HTTP Server | Servidor web que aloja la aplicación PHP. |

### 2.3. Diagrama de Componentes

graph TD  
 A[Usuario/Navegador] -->|HTTP/HTTPS| B(Servidor Web: Apache)  
 B -->|Procesa .php| C[Motor PHP]  
 C -->|Consultas SQL| D[Base de Datos: MySQL]  
 C -->|Genera HTML/Redirige| B  
 B -->|Envía HTML/CSS/JS| A  
  
 subgraph Frontend (Navegador)  
 A\_html[HTML] --> A\_css[CSS]  
 A\_html --> A\_js[JavaScript]  
 A\_js -->|Manipula DOM| A\_html  
 end  
  
 subgraph Backend (Servidor)  
 B -- Apache --> C\_php[PHP Scripts]  
 C\_php --> D\_db[MySQL Database]  
 end

**(Continúa en la siguiente sección: Configuración del Entorno de Desarrollo)**

## 3. Configuración del Entorno de Desarrollo

Esta sección detalla los pasos necesarios para configurar tu entorno de desarrollo y poner en marcha el proyecto Suministros-Dayana.

### 3.1. Requisitos Previos

Asegúrate de tener instalado un paquete de servidor web local como **XAMPP**, **WAMP Server** (Windows) o **MAMP** (macOS). Estos paquetes incluyen Apache, PHP y MySQL, que son esenciales para el proyecto.

### 3.2. Ubicación del Proyecto

Descomprime o clona el repositorio del proyecto Suministros-Dayana en el directorio de documentos de tu servidor web. Las rutas típicas son:

* **XAMPP (Windows):** C:\xampp\htdocs\Suministros-Dayana\
* **WAMP Server (Windows):** C:\wamp64\www\Suministros-Dayana\
* **MAMP (macOS):** /Applications/MAMP/htdocs/Suministros-Dayana/

Una vez ubicado, la URL para acceder al proyecto en tu navegador será similar a http://localhost/Suministros-Dayana/.

### 3.3. Configuración de la Base de Datos (MySQL)

El proyecto depende de una base de datos MySQL para el almacenamiento de datos de usuarios. Sigue estos pasos para configurarla:

1. **Inicia los servicios de Apache y MySQL:** Abre el panel de control de tu XAMPP/WAMP/MAMP e inicia ambos servicios.
2. **Accede a phpMyAdmin:** Abre tu navegador web y ve a http://localhost/phpmyadmin/. Esta es la interfaz gráfica para gestionar tu base de datos MySQL.
3. **Crea la base de datos bytestore\_db:**
   * En phpMyAdmin, haz clic en la pestaña **"Bases de datos"** o en el enlace **"Nueva"** en el menú de la izquierda.
   * En el campo "Nombre de la base de datos", escribe bytestore\_db.
   * Para la "Cotejación" (Collation), se recomienda seleccionar utf8mb4\_general\_ci o utf8mb4\_unicode\_ci para asegurar la correcta manipulación de caracteres especiales.
   * Haz clic en el botón **"Crear"**.
4. **Crea la tabla users:**
   * Una vez creada la base de datos, selecciónala en el panel izquierdo (haz clic en bytestore\_db).
   * Haz clic en la pestaña **"SQL"** en la parte superior.
   * Copia y pega el siguiente código SQL en el cuadro de texto y haz clic en **"Continuar"** o **"Go"** para ejecutarlo. Esta consulta creará la tabla users con las columnas necesarias para almacenar la información de los usuarios.

* -- Si la tabla 'users' ya existe, puedes eliminarla antes de recrearla (opcional, ¡cuidado con los datos existentes!)  
  -- DROP TABLE IF EXISTS users;  
    
  CREATE TABLE users (  
   id INT(11) NOT NULL AUTO\_INCREMENT PRIMARY KEY,  
   full\_name VARCHAR(255) NOT NULL,  
   email VARCHAR(255) NOT NULL UNIQUE,  
   password\_hash VARCHAR(255) NOT NULL,  
   registration\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
  );
  + **Nota sobre UNIQUE en email:** La columna email tiene la restricción UNIQUE, lo que significa que no se pueden registrar dos usuarios con la misma dirección de correo electrónico. Si intentas registrar un email ya existente, el sistema lo impedirá.

### 3.4. Configuración del Archivo de Conexión a la Base de Datos

El archivo php/db\_config.php es crucial para que la aplicación se conecte a tu base de datos. Abre este archivo en tu editor de código y verifica que los parámetros de conexión ($servername, $username, $password, $dbname) coincidan con la configuración de tu servidor MySQL.

// Contenido de php/db\_config.php  
<?php  
// Habilitar reporte de errores (útil para desarrollo)  
error\_reporting(E\_ALL);  
ini\_set('display\_errors', 1);  
  
// Configuración de la base de datos  
$servername = "localhost";  
$username = "root"; // ¡CAMBIA ESTO SI TU USUARIO ES DIFERENTE!  
$password = ""; // ¡CAMBIA ESTO SI TIENES UNA CONTRASEÑA!  
$dbname = "bytestore\_db";  
  
try {  
 // Crear conexión usando MySQLi orientado a objetos  
 $conn = new mysqli($servername, $username, $password, $dbname);  
  
 // Verificar conexión  
 if ($conn->connect\_error) {  
 throw new Exception("Error de conexión a la base de datos: " . $conn->connect\_error);  
 }  
  
 // Establecer charset a utf8mb4 para soporte completo de caracteres  
 if (!$conn->set\_charset("utf8mb4")) {  
 throw new Exception("Error al establecer el charset: " . $conn->error);  
 }  
  
} catch (Exception $e) {  
 die("Error fatal al conectar a la base de datos: " . $e->getMessage());  
}  
?>

* **Advertencia de Seguridad:** En un entorno de producción, error\_reporting(E\_ALL); y ini\_set('display\_errors', 1); deben ser deshabilitados para evitar la exposición de información sensible a los usuarios finales. Los errores deben ser registrados en archivos de log.

**(Continúa en la siguiente sección: Estructura Detallada del Proyecto)**

## 4. Estructura Detallada del Proyecto

El proyecto Suministros-Dayana está organizado en una estructura de directorios modular para separar claramente las responsabilidades de cada tipo de archivo. A continuación, se detalla la finalidad de cada directorio y de los archivos clave dentro de ellos.

Suministros-Dayana/  
├── blog-post-gaming.html  
├── blog-post-seguridad.html  
├── blog-post-tendencias.html  
├── blog.html  
├── carrito.html # Página de visualización completa del carrito (no mini-carrito)  
├── credito-bytestore.html  
├── credito-solvenza.html  
├── css/ # Archivos de estilos CSS  
│ ├── blog.css  
│ ├── chat-widget.css  
│ ├── credito-bytestore.css  
│ ├── exclusivos-online.css  
│ ├── filter-bar.css  
│ ├── header.css  
│ ├── home.css  
│ ├── marcas.css  
│ ├── mini-cart.css # Estilos para el mini-carrito flotante  
│ ├── productos.css  
│ ├── registro.css # Estilos para la página de registro/usuarios  
│ ├── servicio-tecnico.css  
│ ├── styles.css # Estilos globales  
│ └── ubicacion.css  
├── diagnostico-reparacion.html  
├── editar\_registro.php # Script PHP para editar usuarios  
├── eliminar\_registro.php # Script PHP para eliminar usuarios  
├── exclusivos-online.html  
├── favotitos.html  
├── Imagenes/ # Directorio de imágenes de productos y diseño  
├── index.html # Página de inicio (versión HTML)  
├── index.php # Página de inicio (versión PHP, si aplica)  
├── js/ # Archivos JavaScript  
│ ├── chat-widget.js  
│ ├── main.js  
│ ├── marcas.js  
│ ├── mini-cart.js # Lógica JavaScript para el mini-carrito  
│ ├── productos-filters.js  
│ └── sidebar.js  
├── login.html  
├── mantenimiento-preventivo.html  
├── marcas.html  
├── MANUAL\_TECNICO.md # Manual técnico (versión anterior)  
├── MANUAL\_TECNICO\_EXTENSO.md # Este manual técnico extenso  
├── pago-cuotas.html # Página de pago de cuotas  
├── php/ # Scripts PHP del lado del servidor  
│ ├── db\_config.php # Configuración de la conexión a la base de datos  
│ ├── submit\_login.php  
│ ├── submit\_pago\_cuotas.php  
│ ├── submit\_pago.php  
│ ├── submit\_pre\_aprobacion.php  
│ ├── submit\_preaprobacion.php  
│ ├── submit\_registration.php # Maneja el registro de nuevos usuarios  
│ ├── submit\_service\_request.php  
│ └── submit\_solvenza.php  
├── pre-aprobacion.html  
├── productos.html # Página de listado de productos  
├── README.md # README del proyecto (general)  
├── recuperacion-datos.html  
├── registro.html (para mostrar errores/) # Carpeta de respaldo de registro.html (puede ser eliminada)  
├── registro.php # Página de registro de usuarios y listado (PHP)  
├── servicio-tecnico.html  
├── ubicacion.html  
└── Videos/

### 4.1. Directorio Raíz (Suministros-Dayana/)

Contiene los archivos HTML y PHP principales que representan las páginas directas del sitio web, además de los directorios para los recursos.

* **index.html:** La página de inicio principal del sitio web. Sirve como la puerta de entrada para los usuarios.
* **productos.html:** Muestra el catálogo de productos. Es una página estática que se dinamiza con JavaScript para filtros y la integración del mini-carrito.
* **registro.php:** La página de registro de nuevos usuarios. También incluye la funcionalidad para mostrar, paginar y gestionar (editar/eliminar) los registros de usuarios existentes en la base de datos.
  + **Fragmento de Código - Listado y Paginación de Usuarios (registro.php):**
* <?php  
  // Contenido relevante de registro.php  
  require\_once 'php/db\_config.php';  
    
  $recordsPerPage = 5;  
  $currentPage = isset($\_GET['page']) ? (int)$\_GET['page'] : 1;  
  $offset = ($currentPage - 1) \* $recordsPerPage;  
    
  $usuarios = [];  
  $totalPages = 1;  
    
  try {  
   $totalRecordsStmt = $conn->prepare("SELECT COUNT(\*) FROM users");  
   $totalRecordsStmt->execute();  
   $totalRecords = $totalRecordsStmt->get\_result()->fetch\_row()[0];  
   $totalRecordsStmt->close();  
    
   $totalPages = ceil($totalRecords / $recordsPerPage);  
    
   $stmt = $conn->prepare("SELECT id, full\_name, email, registration\_date FROM users ORDER BY registration\_date DESC LIMIT ?, ?");  
   if ($stmt === false) {  
   throw new Exception("Error al preparar la consulta de usuarios: " . $conn->error);  
   }  
   $stmt->bind\_param("ii", $offset, $recordsPerPage);  
   $stmt->execute();  
   $result = $stmt->get\_result();  
   $usuarios = $result->fetch\_all(MYSQLI\_ASSOC);  
   $stmt->close();  
    
  } catch (Exception $e) {  
   error\_log("Error al recuperar registros: " . $e->getMessage());  
  }  
    
  // La conexión se cierra al final del script en el flujo normal.  
  // En este punto, $conn puede estar ya cerrado si se llegó desde una excepción o al final del script.  
  if ($conn && $conn->ping()) { // Verificar si la conexión está activa antes de intentar cerrar  
   $conn->close();   
  }  
  ?>  
  <!-- ... HTML para mostrar la tabla de usuarios ... -->  
  <tbody>  
   <?php if (!empty($usuarios)): ?>  
   <?php foreach ($usuarios as $usuario): ?>  
   <tr>  
   <td><?php echo htmlspecialchars($usuario['id']); ?></td>  
   <td><?php echo htmlspecialchars($usuario['full\_name']); ?></td>  
   <td><?php echo htmlspecialchars($usuario['email']); ?></td>  
   <td><?php echo htmlspecialchars($usuario['registration\_date']); ?></td>  
   <td>  
   <a href="editar\_registro.php?id=<?php echo $usuario['id']; ?>" class="action-button edit">Editar</a>  
   <a href="php/eliminar\_registro.php?id=<?php echo $usuario['id']; ?>" class="action-button delete" onclick="return confirm('¿Estás seguro de que quieres eliminar este registro?');">Eliminar</a>  
   </td>  
   </tr>  
   <?php endforeach; ?>  
   <?php else: ?>  
   <tr>  
   <td colspan="5" style="text-align: center;">No hay usuarios registrados aún.</td>  
   </tr>  
   <?php endif; ?>  
  </tbody>  
  <!-- ... HTML para la paginación ... -->
* **editar\_registro.php:** Página dedicada a la edición de los datos de un usuario específico. Recibe el ID del usuario a través de la URL, carga sus datos en un formulario y permite la actualización.
  + **Fragmento de Código - Carga y Actualización de Usuario (editar\_registro.php):**
* <?php  
  // Contenido relevante de editar\_registro.php  
  require\_once 'php/db\_config.php';  
    
  $user = null;  
  $message = '';  
  $message\_type = '';  
    
  if (isset($\_GET['id']) && !empty($\_GET['id'])) {  
   $user\_id = $\_GET['id'];  
   try {  
   $sql = "SELECT id, full\_name, email FROM users WHERE id = ? LIMIT 1";  
   $stmt = $conn->prepare($sql);  
   $stmt->bind\_param("i", $user\_id);  
   $stmt->execute();  
   $result = $stmt->get\_result();  
   if ($result->num\_rows === 1) {  
   $user = $result->fetch\_assoc();  
   }  
   $stmt->close();  
   } catch (Exception $e) {  
   error\_log("Excepción al obtener registro para edición: " . $e->getMessage());  
   $message = "Error al cargar los datos del usuario.";  
   $message\_type = "danger";  
   }  
  } else if (isset($\_POST['update\_user'])) {  
   $user\_id = htmlspecialchars(trim($\_POST['user\_id']));  
   $full\_name = htmlspecialchars(trim($\_POST['nombre']));  
   $email = htmlspecialchars(trim($\_POST['email']));  
   $password = htmlspecialchars(trim($\_POST['password']));  
   $confirm\_password = htmlspecialchars(trim($\_POST['confirm\_password']));  
    
   // ... (validaciones de formulario) ...  
    
   try {  
   $sql = "UPDATE users SET full\_name = ?, email = ?";  
   $params = ["ss", $full\_name, $email];  
   if (!empty($password)) {  
   $password\_hash = password\_hash($password, PASSWORD\_DEFAULT);  
   $sql .= ", password\_hash = ?";  
   $params[0] .= "s";  
   $params[] = $password\_hash;  
   }  
   $sql .= " WHERE id = ?";  
   $params[0] .= "i";  
   $params[] = $user\_id;  
    
   $stmt = $conn->prepare($sql);  
   if ($stmt === false) {  
   throw new Exception("Error al preparar la consulta de actualización: " . $conn->error);  
   }  
   call\_user\_func\_array([$stmt, 'bind\_param'], refValues($params)); // Para parámetros dinámicos  
   if ($stmt->execute()) {  
   header("Location: registro.php?status=updated");  
   exit();  
   } else {  
   error\_log("Error al ejecutar actualización: " . $stmt->error);  
   $message = "Error al actualizar el registro.";  
   $message\_type = "danger";  
   }  
   $stmt->close();  
   } catch (Exception $e) {  
   error\_log("Excepción al actualizar registro: " . $e->getMessage());  
   $message = "Error: " . $e->getMessage();  
   $message\_type = "danger";  
   }  
  }  
    
  // ... (recarga de usuario si no hay redirección) ...  
    
  $conn->close();  
    
  function refValues($arr){ /\* ... \*/ }  
  ?>  
  <!-- ... HTML del formulario de edición ... -->
* **pago-cuotas.html:** Página de simulación de pago. Recopila información para un pago de cuotas y muestra el monto total del carrito recuperado de localStorage.
  + **Fragmento de Código - Recuperación del Total del Carrito (pago-cuotas.html):**
* <!-- ... HTML existente ... -->  
  <p class="text-center mb-4" style="font-size: 1.2rem; font-weight: bold;">  
   Monto Total del Carrito: <span id="cartTotalAmount">L0.00</span>  
  </p>  
  <!-- ... HTML del formulario ... -->  
  <script>  
   document.addEventListener('DOMContentLoaded', function() {  
   const totalAmount = localStorage.getItem('byteStoreTotalToPay');  
   const cartTotalAmountSpan = document.getElementById('cartTotalAmount');  
   const montoPagoInput = document.getElementById('monto\_pago');  
    
   if (totalAmount) {  
   cartTotalAmountSpan.textContent = `L${parseFloat(totalAmount).toFixed(2)}`;  
   montoPagoInput.value = parseFloat(totalAmount).toFixed(2);  
   } else {  
   cartTotalAmountSpan.textContent = 'L0.00';  
   montoPagoInput.value = '0.00';  
   }  
   });  
  </script>  
  <!-- ... HTML restante ... -->

### 4.2. Directorio css/

Este directorio contiene todos los archivos CSS que definen el estilo y la apariencia visual de la aplicación. Cada archivo está generalmente dedicado a una sección o componente específico para facilitar la organización y el mantenimiento.

* **styles.css:** Contiene estilos globales aplicables a todo el sitio (tipografías, colores base, resets, etc.).
* **productos.css:** Estilos específicos para la página de listado de productos y las tarjetas de producto.
* **mini-cart.css:** Define la apariencia del mini-carrito lateral, incluyendo animaciones, colores y disposición de los ítems.
  + **Fragmento de Código - Estilo de un Ítem del Carrito (mini-cart.css):**
* /\* Estilo para cada ítem del carrito \*/  
  .cart-item {  
   display: flex;  
   align-items: flex-start;  
   padding: 15px 0;  
   border-bottom: 1px dashed #eee;  
   position: relative;  
  }  
    
  /\* Si la imagen está comentada en JS, el width/height de img ya no aplica directamente a un elemento visual  
  pero estas reglas asegurarían que si se descomentara, la imagen se vería bien. \*/  
  .cart-item img {  
   width: 80px;  
   height: 60px;  
   object-fit: contain;  
   margin-right: 15px;  
   border: 1px solid #eee;  
   border-radius: 5px;  
   flex-shrink: 0;  
  }  
    
  .cart-item .item-details {  
   flex-grow: 1;  
   display: flex;  
   flex-direction: column;  
   color: #333;  
   font-size: 0.95rem;  
  }  
    
  .discount-coupon-tag {  
   background-color: #00bcd4;  
   color: white;  
   padding: 5px 10px;  
   border-radius: 3px;  
   font-size: 0.8rem;  
   font-weight: bold;  
   margin-top: 10px;  
   align-self: flex-start;  
  }
* **registro.css:** Contiene estilos para el formulario de registro y la tabla de listado de usuarios.

### 4.3. Directorio js/

Alberga los archivos JavaScript que proporcionan interactividad en el lado del cliente y lógica de UI dinámica.

* **mini-cart.js:** Es el script principal que gestiona la funcionalidad del mini-carrito. Se encarga de añadir/eliminar productos, actualizar cantidades, calcular totales, y controlar la apertura/cierre del carrito.
  + **Fragmento de Código - addToCart y checkoutBtn (js/mini-cart.js):**
* // Contenido relevante de js/mini-cart.js  
  function addToCart(productData) {  
   const existingItem = cart.find(item => item.id === productData.id);  
    
   if (existingItem) {  
   existingItem.quantity++;  
   } else {  
   cart.push({ ...productData, quantity: 1 });  
   }  
   saveCart();  
   openMiniCart(); // Abre el mini-carrito al añadir producto  
  }  
    
  // Event listener para el botón "Proceder a Pagar"  
  checkoutBtn.addEventListener('click', function() {  
   const totals = calculateTotals();  
   localStorage.setItem('byteStoreTotalToPay', totals.total.toFixed(2));  
   localStorage.setItem('byteStoreCartISV', totals.isv.toFixed(2));  
   localStorage.setItem('byteStoreCartSubtotal', totals.subtotal.toFixed(2));  
    
   location.href = 'pago-cuotas.html';   
  });
* **productos-filters.js:** Contiene la lógica para filtrar y ordenar los productos en la página productos.html.
* **sidebar.js:** Maneja la funcionalidad del menú lateral de categorías.

### 4.4. Directorio php/

Contiene los scripts PHP del lado del servidor que interactúan con la base de datos y manejan la lógica de negocio.

* **db\_config.php:** Contiene la configuración y la lógica para establecer la conexión con la base de datos MySQL. Es incluido por otros scripts PHP que necesitan acceder a la base de datos.
  + (El código de este archivo ya fue incluido en la sección 3.4)
* **submit\_registration.php:** Procesar los datos enviados desde el formulario de registro de usuarios (registro.php). Realiza validaciones, hashea la contraseña y guarda el nuevo usuario en la tabla users.
  + **Fragmento de Código - Lógica de Registro (submit\_registration.php):**
* <?php  
  // Contenido relevante de submit\_registration.php  
  session\_start();  
  require\_once 'db\_config.php';  
    
  if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
   $full\_name = htmlspecialchars(trim($\_POST['nombre']));  
   $email = htmlspecialchars(trim($\_POST['email']));  
   $password = htmlspecialchars(trim($\_POST['password']));  
   $confirm\_password = htmlspecialchars(trim($\_POST['confirm\_password']));  
    
   // ... (validaciones de entrada de datos) ...  
    
   $password\_hash = password\_hash($password, PASSWORD\_DEFAULT);  
    
   // Verificar si el correo ya existe  
   $check\_email\_sql = "SELECT id FROM users WHERE email = ?";  
   if ($stmt = $conn->prepare($check\_email\_sql)) {  
   $stmt->bind\_param("s", $email);  
   $stmt->execute();  
   $stmt->store\_result();  
   if ($stmt->num\_rows > 0) {  
   $stmt->close();  
   $conn->close();  
   header("Location: ../registro.php?error=email\_ya\_registrado");  
   exit();  
   }  
   $stmt->close();  
   }  
    
   // Insertar el nuevo usuario  
   $insert\_sql = "INSERT INTO users (full\_name, email, password\_hash) VALUES (?, ?, ?)";  
   if ($stmt = $conn->prepare($insert\_sql)) {  
   $stmt->bind\_param("sss", $full\_name, $email, $password\_hash);  
   if ($stmt->execute()) {  
   header("Location: ../login.html?registration=success");  
   exit();  
   } else {  
   error\_log("Error al ejecutar INSERT: " . $stmt->error);  
   header("Location: ../registro.php?error=db\_execution\_failed");  
   exit();  
   }  
   } else {  
   error\_log("Error al preparar INSERT SQL: " . $conn->error);  
   header("Location: ../registro.php?error=db\_prepare\_failed");  
   exit();  
   }  
  } else {  
   header("Location: ../registro.php");  
   exit();  
  }  
  ?>
* **eliminar\_registro.php:** Se encarga de procesar la solicitud de eliminación de un usuario. Recibe el ID del usuario y ejecuta una consulta DELETE en la base de datos.
  + **Fragmento de Código - Lógica de Eliminación (eliminar\_registro.php):**
* <?php  
  // Contenido relevante de eliminar\_registro.php  
  require\_once 'db\_config.php';  
    
  if (isset($\_GET['id']) && !empty($\_GET['id'])) {  
   $user\_id = $\_GET['id'];  
   try {  
   $sql = "DELETE FROM users WHERE id = ?";  
   $stmt = $conn->prepare($sql);  
   if ($stmt === false) {  
   throw new Exception("Error al preparar la consulta de eliminación: " . $conn->error);  
   }  
   $stmt->bind\_param("i", $user\_id);  
   if ($stmt->execute()) {  
   header("Location: ../registro.php?status=deleted");  
   exit();  
   } else {  
   error\_log("Error al ejecutar la eliminación: " . $stmt->error);  
   header("Location: ../registro.php?status=error\_delete\_execution");  
   exit();  
   }  
   $stmt->close();  
   } catch (Exception $e) {  
   error\_log("Excepción al eliminar registro: " . $e->getMessage());  
   header("Location: ../registro.php?status=error\_delete\_exception");  
   exit();  
   }  
  } else {  
   header("Location: ../registro.php?status=id\_not\_provided");  
   exit();  
  }  
  $conn->close();  
  ?>
* **Otros submit\_\*.php:** Archivos PHP para manejar el envío de otros formularios (login, pagos, solicitudes de servicio, etc.), interactuando con la base de datos según su propósito específico.

**(Continúa en la siguiente sección: Flujos de Trabajo y Consideraciones)**

## 5. Flujos de Trabajo Clave

Esta sección describe los flujos de interacción más importantes dentro de la aplicación, detallando cómo los componentes del frontend y el backend trabajan juntos para lograr las funcionalidades.

### 5.1. Registro de Usuarios

El flujo de registro permite a los nuevos usuarios crear una cuenta en el sistema. Involucra tanto la interfaz de usuario para la entrada de datos como el procesamiento del lado del servidor para almacenar la información de forma segura.

1. \*\*Interacción del Usuario (Frontend - registro.php):  
   \*\* El usuario accede a registro.php y completa el formulario de registro con su nombre completo, correo electrónico, contraseña y confirmación de contraseña. El formulario usa el método POST para enviar los datos.
2. \*\*Envío y Procesamiento (Backend - php/submit\_registration.php):  
   \*\* Cuando el formulario se envía, los datos son recibidos por php/submit\_registration.php.
   * **Validación de Entrada:** El script primero valida que todos los campos requeridos estén presentes y que las contraseñas coincidan. También verifica si el correo electrónico ya existe en la base de datos para evitar duplicados.
   * **Hashing de Contraseña:** Antes de almacenar la contraseña, se aplica una función de hashing (password\_hash()) para convertir la contraseña de texto plano en una cadena de caracteres ilegible. Esto es crucial para la seguridad, ya que nunca se debe almacenar contraseñas en texto plano.
   * **Almacenamiento en Base de Datos:** Si todas las validaciones son exitosas, el script inserta el nuevo usuario (nombre completo, correo electrónico y hash de contraseña) en la tabla users de la base de datos.
   * **Redirección:** Tras un registro exitoso, el usuario es redirigido a login.html (con un parámetro de éxito en la URL). Si hay errores (ej. email duplicado, validación fallida), se redirige de nuevo a registro.php con un parámetro de error.

* **Código Relevante - Lógica de Procesamiento en php/submit\_registration.php (Resumen):**
* <?php  
  session\_start();  
  require\_once 'db\_config.php';  
    
  if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
   $full\_name = htmlspecialchars(trim($\_POST['nombre']));  
   $email = htmlspecialchars(trim($\_POST['email']));  
   $password = htmlspecialchars(trim($\_POST['password']));  
   $confirm\_password = htmlspecialchars(trim($\_POST['confirm\_password']));  
    
   // 1. Validaciones básicas (campos vacíos, contraseñas coinciden, formato email)  
   if (empty($full\_name) || empty($email) || empty($password) || empty($confirm\_password)) {  
   header("Location: ../registro.php?error=campos\_vacios");  
   exit();  
   }  
   if ($password !== $confirm\_password) {  
   header("Location: ../registro.php?error=passwords\_no\_coinciden");  
   exit();  
   }  
   if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {  
   header("Location: ../registro.php?error=email\_invalido");  
   exit();  
   }  
    
   // 2. Verificar si el correo ya existe en la base de datos  
   $check\_email\_sql = "SELECT id FROM users WHERE email = ?";  
   if ($stmt = $conn->prepare($check\_email\_sql)) {  
   $stmt->bind\_param("s", $email);  
   $stmt->execute();  
   $stmt->store\_result();  
   if ($stmt->num\_rows > 0) {  
   $stmt->close();  
   $conn->close();  
   header("Location: ../registro.php?error=email\_ya\_registrado");  
   exit();  
   }  
   $stmt->close();  
   } else {  
   error\_log("Error al preparar consulta de verificación de email: " . $conn->error);  
   header("Location: ../registro.php?error=db\_prepare\_failed");  
   exit();  
   }  
    
   // 3. Hashear la contraseña  
   $password\_hash = password\_hash($password, PASSWORD\_DEFAULT);  
    
   // 4. Insertar el nuevo usuario en la base de datos  
   $insert\_sql = "INSERT INTO users (full\_name, email, password\_hash) VALUES (?, ?, ?)";  
   if ($stmt = $conn->prepare($insert\_sql)) {  
   $stmt->bind\_param("sss", $full\_name, $email, $password\_hash);  
   if ($stmt->execute()) {  
   header("Location: ../login.html?registration=success");  
   exit();  
   } else {  
   error\_log("Error al ejecutar INSERT: " . $stmt->error);  
   header("Location: ../registro.php?error=db\_execution\_failed");  
   exit();  
   }  
   } else {  
   error\_log("Error al preparar INSERT SQL: " . $conn->error);  
   header("Location: ../registro.php?error=db\_prepare\_failed");  
   exit();  
   }  
  } else {  
   header("Location: ../registro.php");  
   exit();  
  }  
  ?>

### 5.2. Gestión de Usuarios (Listado, Edición y Eliminación)

El archivo registro.php no solo maneja el formulario de registro, sino que también sirve como un panel de administración básico para listar, editar y eliminar usuarios.

1. \*\*Listado de Usuarios (Frontend & Backend - registro.php):  
   \*\*
   * Cuando se carga registro.php, el script PHP se conecta a la base de datos y recupera una lista paginada de usuarios de la tabla users.
   * La paginación se implementa usando parámetros LIMIT y OFFSET en la consulta SQL, basándose en el número de página (page) de la URL.
   * Los datos recuperados se muestran en una tabla HTML. Para cada usuario, se proporcionan botones/enlaces para "Editar" y "Eliminar".

* **Código Relevante - Listado y Paginación en registro.php (Resumen):**
* <?php  
  require\_once 'php/db\_config.php';  
    
  $recordsPerPage = 5; // Número de registros por página  
  $currentPage = isset($\_GET['page']) ? (int)$\_GET['page'] : 1;  
  $offset = ($currentPage - 1) \* $recordsPerPage;  
    
  $usuarios = []; // Array para almacenar los usuarios  
  $totalPages = 1; // Inicializar total de páginas  
    
  try {  
   // Contar el total de registros para la paginación  
   $totalRecordsStmt = $conn->prepare("SELECT COUNT(\*) FROM users");  
   if ($totalRecordsStmt === false) {  
   throw new Exception("Error al preparar la consulta de conteo: " . $conn->error);  
   }  
   $totalRecordsStmt->execute();  
   $totalRecords = $totalRecordsStmt->get\_result()->fetch\_row()[0];  
   $totalRecordsStmt->close();  
    
   $totalPages = ceil($totalRecords / $recordsPerPage);  
    
   // Obtener los usuarios para la página actual  
   $stmt = $conn->prepare("SELECT id, full\_name, email, registration\_date FROM users ORDER BY registration\_date DESC LIMIT ?, ?");  
   if ($stmt === false) {  
   throw new Exception("Error al preparar la consulta de usuarios: " . $conn->error);  
   }  
   $stmt->bind\_param("ii", $offset, $recordsPerPage);  
   $stmt->execute();  
   $result = $stmt->get\_result();  
   $usuarios = $result->fetch\_all(MYSQLI\_ASSOC);  
   $stmt->close();  
    
  } catch (Exception $e) {  
   error\_log("Error al recuperar registros para tabla: " . $e->getMessage());  
   // Opcional: mostrar un mensaje de error al usuario en el frontend  
  }  
    
  // ... (HTML para la tabla y paginación) ...  
  ?>

1. \*\*Edición de Usuarios (Frontend - editar\_registro.php & Backend - editar\_registro.php POST):  
   \*\*
   * Cuando se hace clic en "Editar" en registro.php, se redirige a editar\_registro.php?id={ID\_USUARIO}.
   * El script editar\_registro.php recibe el ID del usuario de la URL ($\_GET).
   * Realiza una consulta a la base de datos para obtener los datos actuales de ese usuario y pre-rellena un formulario de edición.
   * Cuando el usuario envía el formulario de edición (POST), el mismo script editar\_registro.php procesa la actualización:
     + Valida los datos (nombre, email, y si se ingresó, las nuevas contraseñas).
     + Si se proporciona una nueva contraseña, esta se hashea antes de la actualización.
     + Ejecuta una consulta UPDATE en la base de datos para modificar los datos del usuario.
     + Redirige de vuelta a registro.php tras una actualización exitosa.

* **Código Relevante - Lógica de Edición en editar\_registro.php (Resumen):**
* <?php  
  require\_once 'php/db\_config.php';  
    
  $user = null;  
  $message = '';  
  $message\_type = '';  
    
  // Lógica para cargar datos del usuario si se accede con GET (para mostrar el formulario)  
  if (isset($\_GET['id']) && !empty($\_GET['id'])) {  
   $user\_id = $\_GET['id'];  
   $sql = "SELECT id, full\_name, email FROM users WHERE id = ? LIMIT 1";  
   if ($stmt = $conn->prepare($sql)) {  
   $stmt->bind\_param("i", $user\_id);  
   $stmt->execute();  
   $result = $stmt->get\_result();  
   if ($result->num\_rows === 1) {  
   $user = $result->fetch\_assoc();  
   } else {  
   $message = "Usuario no encontrado.";  
   $message\_type = "danger";  
   }  
   $stmt->close();  
   } else {  
   error\_log("Error al preparar consulta de carga de usuario: " . $conn->error);  
   $message = "Error interno al cargar usuario.";  
   $message\_type = "danger";  
   }  
  }  
    
  // Lógica para procesar la actualización si se accede con POST (después de enviar el formulario)  
  if (isset($\_POST['update\_user'])) {  
   $user\_id = htmlspecialchars(trim($\_POST['user\_id']));  
   $full\_name = htmlspecialchars(trim($\_POST['nombre']));  
   $email = htmlspecialchars(trim($\_POST['email']));  
   $password = htmlspecialchars(trim($\_POST['password']));  
    
   // Validaciones (similar a registro, pero con lógica para actualización de contraseña opcional)  
   // ...  
    
   $sql = "UPDATE users SET full\_name = ?, email = ?";  
   $params\_types = "ss";  
   $params\_values = [$full\_name, $email];  
    
   if (!empty($password)) {  
   $password\_hash = password\_hash($password, PASSWORD\_DEFAULT);  
   $sql .= ", password\_hash = ?";  
   $params\_types .= "s";  
   $params\_values[] = $password\_hash;  
   }  
    
   $sql .= " WHERE id = ?";  
   $params\_types .= "i";  
   $params\_values[] = $user\_id;  
    
   if ($stmt = $conn->prepare($sql)) {  
   $stmt->bind\_param($params\_types, ...$params\_values);  
   if ($stmt->execute()) {  
   header("Location: registro.php?status=updated");  
   exit();  
   } else {  
   error\_log("Error al ejecutar UPDATE: " . $stmt->error);  
   $message = "Error al actualizar el registro.";  
   $message\_type = "danger";  
   }  
   $stmt->close();  
   } else {  
   error\_log("Error al preparar UPDATE SQL: " . $conn->error);  
   $message = "Error interno al preparar actualización.";  
   $message\_type = "danger";  
   }  
    
   // Recargar datos del usuario para mostrar el formulario con los datos actualizados  
   // Esto es útil si no hay redirección inmediata por algún error  
   // ... (código similar al de carga de GET para $user)  
  }  
  // ... (HTML del formulario de edición)  
  ?>

1. \*\*Eliminación de Usuarios (Backend - php/eliminar\_registro.php):  
   \*\*
   * Cuando se hace clic en "Eliminar" en registro.php, se redirige a php/eliminar\_registro.php?id={ID\_USUARIO}.
   * Se utiliza un onclick="return confirm(...)" en el enlace para una confirmación previa del lado del cliente.
   * El script php/eliminar\_registro.php recibe el ID del usuario a través de $\_GET.
   * Ejecuta una consulta DELETE en la tabla users para eliminar el registro correspondiente.
   * Redirige de vuelta a registro.php con un parámetro de estado (status=deleted o status=error).

* **Código Relevante - Lógica de Eliminación en php/eliminar\_registro.php (Resumen):**
* <?php  
  require\_once 'db\_config.php';  
    
  if (isset($\_GET['id']) && !empty($\_GET['id'])) {  
   $user\_id = (int)$\_GET['id']; // Asegurar que es un entero  
    
   $sql = "DELETE FROM users WHERE id = ?";  
   if ($stmt = $conn->prepare($sql)) {  
   $stmt->bind\_param("i", $user\_id);  
   if ($stmt->execute()) {  
   header("Location: ../registro.php?status=deleted");  
   exit();  
   } else {  
   error\_log("Error al ejecutar la eliminación: " . $stmt->error);  
   header("Location: ../registro.php?status=error\_delete\_execution");  
   exit();  
   }  
   $stmt->close();  
   } else {  
   error\_log("Error al preparar la consulta de eliminación: " . $conn->error);  
   header("Location: ../registro.php?status=error\_delete\_prepare");  
   exit();  
   }  
  } else {  
   header("Location: ../registro.php?status=id\_not\_provided");  
   exit();  
  }  
  $conn->close();  
  ?>

### 5.3. Gestión del Mini-Carrito de Compras

La funcionalidad del mini-carrito se implementa completamente en el lado del cliente utilizando JavaScript, lo que proporciona una experiencia de usuario fluida sin recargas de página constantes.

1. \*\*Añadir Producto al Carrito (Frontend - productos.html & js/mini-cart.js):  
   \*\*
   * En productos.html, cada botón "Añadir al carrito" (.add-to-cart-btn) tiene atributos data-\* que almacenan la información del producto (ID, nombre, precio, etc.).
   * El script js/mini-cart.js escucha los clics en estos botones. Cuando se detecta un clic, recupera los datos del producto.
   * El producto se añade a un array cart en JavaScript. Si el producto ya está en el carrito, se incrementa su cantidad.
   * El carrito se guarda en localStorage del navegador para persistir entre sesiones.
   * La interfaz del mini-carrito (un div flotante) se actualiza dinámicamente para mostrar el nuevo ítem y el total, y se abre automáticamente.

* **Código Relevante - Función addToCart en js/mini-cart.js (Resumen):**
* // js/mini-cart.js  
    
  let cart = loadCart(); // Carga el carrito desde localStorage o inicializa vacío  
    
  function addToCart(productData) {  
   const existingItem = cart.find(item => item.id === productData.id);  
    
   if (existingItem) {  
   existingItem.quantity++;  
   } else {  
   cart.push({ ...productData, quantity: 1 });  
   }  
   saveCart(); // Guarda el carrito actualizado en localStorage  
   renderCart(); // Vuelve a dibujar el contenido del mini-carrito  
   updateCartTotals(); // Actualiza los totales en la UI  
   openMiniCart(); // Muestra el mini-carrito  
  }  
    
  // Listener para los botones de añadir al carrito en productos.html  
  document.querySelectorAll('.add-to-cart-btn').forEach(button => {  
   button.addEventListener('click', function() {  
   const productId = this.dataset.id;  
   const productName = this.dataset.name;  
   const productPrice = parseFloat(this.dataset.price);  
   const productImage = this.dataset.image; // Path de la imagen (corregido a ruta local)  
    
   const productData = {  
   id: productId,  
   name: productName,  
   price: productPrice,  
   image: productImage  
   };  
   addToCart(productData);  
   });  
  });

1. \*\*Actualización de Cantidad y Eliminación (Frontend - js/mini-cart.js):  
   \*\*
   * Dentro del mini-carrito, los usuarios pueden aumentar/disminuir la cantidad de un producto o eliminarlo por completo.
   * EventListeners en los botones +, - y x (eliminar) detectan estas interacciones.
   * Las funciones correspondientes (updateQuantity, removeItem) modifican el array cart.
   * Después de cada cambio, el carrito se guarda en localStorage y la UI del mini-carrito se re-renderiza para reflejar las modificaciones.

### 5.4. Proceso de Pago Simplificado

El proyecto incluye una simulación de un proceso de pago, donde el total del mini-carrito se transfiere a una página de pago.

1. \*\*Inicio del Proceso de Pago (Frontend - js/mini-cart.js):  
   \*\*
   * Cuando el usuario hace clic en "PROCEDER A PAGAR" en el mini-carrito, se activa un EventListener en js/mini-cart.js.
   * Antes de la redirección, el script calcula el subtotal, ISV (Impuesto sobre Ventas) y el total final del carrito.
   * Estos valores se almacenan en localStorage bajo claves específicas (byteStoreTotalToPay, byteStoreCartISV, byteStoreCartSubtotal). Esto permite que los datos persistan y sean accesibles en la siguiente página.
   * Finalmente, el navegador es redirigido a pago-cuotas.html.

* **Código Relevante - Redirección a Pago en js/mini-cart.js (Resumen):**
* // js/mini-cart.js  
  const checkoutBtn = document.getElementById('checkout-btn');  
    
  checkoutBtn.addEventListener('click', function() {  
   const totals = calculateTotals(); // Función que calcula subtotal, ISV, total  
    
   // Guardar totales en localStorage para la página de pago  
   localStorage.setItem('byteStoreTotalToPay', totals.total.toFixed(2));  
   localStorage.setItem('byteStoreCartISV', totals.isv.toFixed(2));  
   localStorage.setItem('byteStoreCartSubtotal', totals.subtotal.toFixed(2));  
    
   // Redirigir a la página de pago  
   location.href = 'pago-cuotas.html';  
  });

1. \*\*Visualización del Total en la Página de Pago (Frontend - pago-cuotas.html):  
   \*\*
   * Al cargar pago-cuotas.html, un bloque de script JavaScript incrustado (<script>) se ejecuta.
   * Este script recupera los valores de total, isv y subtotal que se guardaron previamente en localStorage.
   * Los valores se muestran en elementos <span> designados en la página y también se pre-rellenan en el campo de entrada "Monto a Pagar" (#monto\_pago).

* **Código Relevante - Recuperación en pago-cuotas.html (Resumen):**
* <!-- pago-cuotas.html -->  
  <p class="text-center mb-4" style="font-size: 1.2rem; font-weight: bold;">  
   Monto Total del Carrito: <span id="cartTotalAmount">L0.00</span>  
  </p>  
    
  <form action="php/submit\_pago\_cuotas.php" method="POST">  
   <!-- ... otros campos del formulario ... -->  
   <div class="form-group">  
   <label for="monto\_pago">Monto a Pagar:</label>  
   <input type="number" class="form-control" id="monto\_pago" name="monto\_pago" step="0.01" required readonly>  
   </div>  
   <!-- ... botón de envío ... -->  
  </form>  
    
  <script>  
   document.addEventListener('DOMContentLoaded', function() {  
   const totalAmount = localStorage.getItem('byteStoreTotalToPay');  
   const cartTotalAmountSpan = document.getElementById('cartTotalAmount');  
   const montoPagoInput = document.getElementById('monto\_pago');  
    
   if (totalAmount) {  
   cartTotalAmountSpan.textContent = `L${parseFloat(totalAmount).toFixed(2)}`;  
   montoPagoInput.value = parseFloat(totalAmount).toFixed(2);  
   } else {  
   cartTotalAmountSpan.textContent = 'L0.00';  
   montoPagoInput.value = '0.00';  
   }  
   });  
  </script>

**(Continúa en la siguiente sección: Consideraciones de Seguridad)**

## 6. Consideraciones de Seguridad

La seguridad es un aspecto fundamental en el desarrollo de cualquier aplicación web. Este proyecto ha implementado algunas prácticas de seguridad básicas, pero es crucial entender que siempre hay espacio para mejoras, especialmente en un entorno de producción. A continuación, se detallan las consideraciones de seguridad actuales y recomendaciones.

### 6.1. Hashing de Contraseñas

Las contraseñas de los usuarios nunca se almacenan en texto plano en la base de datos. En su lugar, se utiliza la función password\_hash() de PHP con el algoritmo PASSWORD\_DEFAULT para generar un hash seguro. Esta función incorpora un salt aleatorio y maneja la complejidad del algoritmo por defecto, lo que protege contra ataques de tablas arcoíris y fuerza bruta.

* **Ubicación:** php/submit\_registration.php y editar\_registro.php.
* **Implementación:**
* // En php/submit\_registration.php o editar\_registro.php (cuando se actualiza la contraseña)  
  $password = $\_POST['password']; // Contraseña ingresada por el usuario  
  $password\_hash = password\_hash($password, PASSWORD\_DEFAULT);  
  // Luego, $password\_hash se almacena en la base de datos
* **Verificación de Contraseña:** Para verificar una contraseña, se utiliza password\_verify():
* // Ejemplo de verificación de contraseña (en un futuro submit\_login.php)  
  $password\_ingresada = $\_POST['password'];  
  $hash\_almacenado = $user['password\_hash']; // Hash obtenido de la base de datos  
    
  if (password\_verify($password\_ingresada, $hash\_almacenado)) {  
   // Contraseña correcta  
  } else {  
   // Contraseña incorrecta  
  }

### 6.2. Prevención de Inyección SQL (Consultas Preparadas)

El proyecto utiliza consultas preparadas (prepared statements) con MySQLi para interactuar con la base de datos. Esto es una defensa esencial contra ataques de inyección SQL. Al usar prepare() y bind\_param(), los datos de entrada del usuario se tratan como datos, no como parte de la consulta SQL, lo que evita que un atacante manipule la consulta.

* **Ubicación:** php/submit\_registration.php, registro.php, editar\_registro.php, php/eliminar\_registro.php, y cualquier otro script PHP que interactúe con la base de datos.
* **Implementación Típica:**
* // Ejemplo de SELECT con consulta preparada  
  $email = $\_POST['email'];  
  $stmt = $conn->prepare("SELECT id, full\_name, password\_hash FROM users WHERE email = ?");  
  if ($stmt === false) {  
   die("Error al preparar la consulta: " . $conn->error);  
  }  
  $stmt->bind\_param("s", $email); // "s" indica que $email es un string  
  $stmt->execute();  
  $result = $stmt->get\_result();  
  // ... procesar resultados ...  
  $stmt->close();
* // Ejemplo de INSERT con consulta preparada  
  $full\_name = $\_POST['nombre'];  
  $email = $\_POST['email'];  
  $password\_hash = $hashed\_password\_from\_above;  
    
  $stmt = $conn->prepare("INSERT INTO users (full\_name, email, password\_hash) VALUES (?, ?, ?)");  
  if ($stmt === false) {  
   die("Error al preparar la consulta: " . $conn->error);  
  }  
  $stmt->bind\_param("sss", $full\_name, $email, $password\_hash); // "sss" indica 3 strings  
  $stmt->execute();  
  // ... verificar éxito ...  
  $stmt->close();

### 6.3. Sanitización de Entrada de Datos

Además de las consultas preparadas, se utiliza htmlspecialchars() y trim() para limpiar y escapar los datos de entrada del usuario. Esto ayuda a prevenir ataques de Cross-Site Scripting (XSS) al asegurar que el contenido ingresado por el usuario se muestre de forma segura en las páginas web, sin ser interpretado como código HTML o JavaScript.

* **Ubicación:** En todos los scripts PHP que reciben datos del usuario ($\_POST, $\_GET).
* **Implementación:**
* // Ejemplo en submit\_registration.php o editar\_registro.php  
  $full\_name = htmlspecialchars(trim($\_POST['nombre']));  
  $email = htmlspecialchars(trim($\_POST['email']));  
  // Aplicado a todas las variables de entrada

### 6.4. Manejo de Errores y Registro de Logs

Durante el desarrollo, se ha habilitado el reporte de errores (error\_reporting(E\_ALL); ini\_set('display\_errors', 1);) para facilitar la depuración. Sin embargo, para un entorno de producción, estos deben ser deshabilitados y, en su lugar, los errores deben ser registrados en archivos de log seguros (error\_log()). Esto evita la exposición de rutas de archivos, consultas SQL o mensajes de error detallados a los usuarios finales, que podrían ser explotados por atacantes.

* **Ubicación:** php/db\_config.php y otros scripts PHP.
* **Recomendación para Producción:**
* // En php/db\_config.php (para producción)  
  error\_reporting(0); // Deshabilitar todos los reportes de errores en el navegador  
  ini\_set('display\_errors', 0); // No mostrar errores en la salida  
  ini\_set('log\_errors', 1); // Habilitar el registro de errores  
  ini\_set('error\_log', '/path/to/your/php\_error.log'); // Especificar la ruta del archivo de log  
    
  // Ejemplo de registro de error en cualquier script PHP  
  try {  
   // ... alguna operación que podría fallar ...  
  } catch (Exception $e) {  
   error\_log("Error crítico en la aplicación: " . $e->getMessage() . " en " . $e->getFile() . " en la línea " . $e->getLine());  
   // Mostrar un mensaje de error genérico al usuario  
   header("Location: ../error\_page.html"); // Redirigir a una página de error amigable  
   exit();  
  }

### 6.5. Gestión de Sesiones (Consideraciones Futuras)

Aunque el proyecto actual no implementa un sistema de autenticación de usuario completo con sesiones de PHP para mantener el estado de login, es una consideración de seguridad crítica para futuras expansiones.

* **Recomendaciones:**
  + Utilizar session\_start() al inicio de cada script que necesite manejar sesiones.
  + Regenerar el ID de sesión (session\_regenerate\_id(true);) después de un login exitoso para prevenir ataques de fijación de sesión.
  + Almacenar solo información mínima y no sensible directamente en la sesión ($\_SESSION).
  + Configurar session.cookie\_httponly = true y session.cookie\_secure = true (para HTTPS) en php.ini para proteger las cookies de sesión.

### 6.6. Autenticación de Acceso a Archivos PHP (Protección de Directorios)

Los archivos PHP en el directorio php/ que procesan datos (ej. submit\_registration.php, eliminar\_registro.php) deben ser accedidos solo a través de POST o GET con validaciones adecuadas. Acceder a ellos directamente en el navegador sin los parámetros correctos podría generar errores o comportamientos inesperados.

* **Práctica:** Siempre verificar $\_SERVER["REQUEST\_METHOD"] y la presencia de parámetros esperados (isset($\_POST['...']), isset($\_GET['...'])).
* **Ejemplo:**
* // En submit\_registration.php  
  if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
   // Procesar formulario  
  } else {  
   // Si se accede directamente por GET, redirigir o mostrar error  
   header("Location: ../registro.php");  
   exit();  
  }

### 6.7. Uso de HTTPS

Para un entorno de producción, es imprescindible utilizar HTTPS para encriptar todas las comunicaciones entre el cliente y el servidor. Esto protege los datos sensibles (credenciales, información personal, detalles de pago) de ser interceptados por atacantes (man-in-the-middle attacks).

* **Implementación:** Requiere un certificado SSL/TLS en el servidor web (Apache).

### 6.8. Actualizaciones y Parches

Mantener PHP, MySQL y el servidor web (Apache) actualizados con las últimas versiones y parches de seguridad es vital. Las vulnerabilidades conocidas son a menudo explotadas en sistemas desactualizados.

**(Continúa en la siguiente sección: Mantenimiento y Depuración)**

## 7. Mantenimiento y Depuración

Esta sección proporciona pautas y técnicas para el mantenimiento regular del proyecto Suministros-Dayana y estrategias para la depuración eficiente de problemas que puedan surgir.

### 7.1. Mantenimiento Regular

El mantenimiento continuo es clave para asegurar el rendimiento, la seguridad y la estabilidad de la aplicación.

* **Actualización de Componentes:**
  + **Servidor Web (Apache), PHP y MySQL:** Mantener estos componentes actualizados es crucial para la seguridad y el rendimiento. Las nuevas versiones a menudo incluyen parches de seguridad para vulnerabilidades descubiertas y mejoras de eficiencia.
  + **Dependencias de Frontend:** Aunque este proyecto utiliza principalmente Bootstrap y FontAwesome de forma estática, en proyectos más grandes con gestores de paquetes (npm/Yarn), sería necesario actualizar periódicamente las librerías JavaScript y CSS.
* **Respaldo de la Base de Datos:** Realizar copias de seguridad regulares de la base de datos bytestore\_db es fundamental para prevenir la pérdida de datos. Puedes usar phpMyAdmin (Exportar) o herramientas de línea de comandos como mysqldump.
  + **Ejemplo mysqldump (en terminal):**
  + mysqldump -u root -p bytestore\_db > bytestore\_db\_backup\_$(date +%Y%m%d).sql
  + (Reemplaza root con tu usuario de MySQL y, si tienes contraseña, se te pedirá después de -p).
* **Monitoreo de Logs:** Revisar regularmente los logs de errores de Apache y PHP (configurados en php.ini o en la sección 6.4 de este manual) para identificar y resolver problemas de forma proactiva.
* **Optimización de la Base de Datos:** Ocasionalmente, puedes ejecutar la operación OPTIMIZE TABLE en tablas de MySQL para desfragmentarlas y mejorar el rendimiento de las consultas, especialmente si hay muchas operaciones de inserción/eliminación.
  + **Ejemplo SQL (en phpMyAdmin o cliente MySQL):**
  + OPTIMIZE TABLE users;

### 7.2. Estrategias de Depuración

Cuando surgen problemas, una depuración sistemática es esencial.

* \*\*Mensajes de Error de PHP (error\_reporting y display\_errors):  
  \*\*
  + **Durante el Desarrollo:** Mantén error\_reporting(E\_ALL); y ini\_set('display\_errors', 1); habilitados en php/db\_config.php (o en tu php.ini) para ver los errores directamente en el navegador. Recuerda desactivarlos en producción.
  + **Registro de Errores:** Utiliza error\_log("Mensaje de depuración o error"); para escribir mensajes en el log de errores de PHP. Esto es útil en entornos de producción donde display\_errors está desactivado.
* \*\*Depuración de Base de Datos (MySQLi Errores):  
  \*\*
  + Cuando una consulta preparada falla (prepare() devuelve false), el objeto $conn de MySQLi tendrá información detallada sobre el error ($conn->error).
  + De manera similar, si execute() o bind\_param() fallan en un stmt, puedes usar $stmt->error para obtener el mensaje de error.
  + **Ejemplo de Verificación de Errores de MySQLi:**
* // En cualquier script PHP que use la base de datos  
  $stmt = $conn->prepare("SELECT \* FROM non\_existent\_table"); // Ejemplo de error  
  if ($stmt === false) {  
   error\_log("Error al preparar consulta: " . $conn->error); // Log el error de conexión/preparación  
   // Opcional: echo "Error: " . $conn->error; en desarrollo  
  } else {  
   $stmt->bind\_param("i", $id);  
   if ($stmt->execute() === false) {  
   error\_log("Error al ejecutar consulta: " . $stmt->error); // Log el error de ejecución  
   // Opcional: echo "Error: " . $stmt->error; en desarrollo  
   }  
   // ...  
  }
* \*\*Uso de var\_dump() y print\_r() (PHP):  
  \*\*
  + Estas funciones son invaluables para inspeccionar el contenido de variables, arrays y objetos en cualquier punto de tu código PHP.
  + **Ejemplo:**
  + // Para ver el contenido de los datos POST  
    echo "<pre>"; var\_dump($\_POST); echo "</pre>";  
      
    // Para ver los usuarios recuperados  
    echo "<pre>"; print\_r($usuarios); echo "</pre>";
  + **Nota:** Siempre encierra la salida de var\_dump() o print\_r() dentro de etiquetas <pre> para una mejor legibilidad en el navegador.
* \*\*Consola del Navegador (JavaScript y Frontend):  
  \*\*
  + La consola de desarrollador de tu navegador (F12 en Chrome/Firefox) es tu mejor amiga para depurar problemas de JavaScript, CSS y HTML.
  + **Errores de JavaScript:** Busca mensajes de error en la pestaña "Consola".
  + **Errores de Red:** La pestaña "Red" (Network) te mostrará todas las solicitudes HTTP realizadas por la página, sus respuestas y cualquier error de estado (404 Not Found, 500 Internal Server Error, etc.). Esto es útil para verificar si los scripts PHP se están cargando correctamente o si hay problemas de redirección.
  + **console.log():** Utiliza console.log() en tu código JavaScript para imprimir valores de variables, objetos y mensajes en la consola del navegador.
* // En js/mini-cart.js o cualquier script JS  
  console.log("Carrito actual:", cart);  
  console.log("Total calculado:", totals.total);
* \*\*Herramientas del Navegador (Inspector de Elementos y Estilos):  
  \*\*
  + La pestaña "Elementos" (Elements) te permite inspeccionar la estructura HTML del DOM y modificar estilos CSS en tiempo real para probar cambios visuales.
  + La pestaña "Estilos" (Styles) te muestra qué reglas CSS se están aplicando a un elemento seleccionado, de dónde provienen y si hay alguna regla que las esté sobrescribiendo.
* **Entornos de Desarrollo Integrados (IDE):**
  + Si utilizas un IDE como VS Code, PhpStorm o Sublime Text, considera la configuración de extensiones para depuración (por ejemplo, Xdebug para PHP) que permiten establecer puntos de interrupción, seguir el flujo de ejecución y examinar el estado de las variables en tiempo real.