Proyecto Final Programado

Nombre del Proyecto: Gestión de Control de Ingresos y Trabajos para un Taller de Motocicletas

Nombre del Estudiante/Desarrollador: Cesar Antonio Ramírez Calderón

Profesor/Curso: Estefanía Boza Villalobos

Fecha: 25/11/2025

Tabla de Contenido

Parte I: Información General del Proyecto 3

Objetivo General 3

Objetivos Específicos 3

Justificación del Proyecto 3

Alcances Esperados 4

Requerimientos del Proyecto 4

Parte II: Diseño del Sistema (Pre-diseño visual) 5

Elementos de Diseño 5

Explicación 6

Parte III: Base de Datos (MySQL) 8

1. Definición 8

2. Elementos 8

Parte IV: Desarrollo del Backend 15

1. Arquitectura del Backend 15

2. Componentes y Estructura del Codigo 15

3. Elementos Clave del Backend 20

4. Conexión con la base de datos y el frontend: 24

5. Capturas o fragmentos de código: 28

Descripción: 29

Descripción: 29

Descripción: 30

Descripción: 30

Descripción: 31

Conclusión 32

Referencias 32

Parte I: Información General del Proyecto

El proyecto consiste en desarrollar una herramienta tecnológica que permita registrar el ingreso de vehículos (motocicletas) a un taller, así como detallar los trabajos realizados a cada vehículo, contando de igual manera con su respectiva facturación una vez completado el mantenimiento.

Objetivo General

Desarrollar un sistema de información web que permita al taller mecánico de motocicletas registrar, gestionar y controlar el ingreso de vehículos, así como almacenar datos relevantes tanto de la motocicleta (marca, cilindraje, kilometraje, etc.) como de la información de contacto de sus propietarios.

Objetivos Específicos

1. Implementar un módulo de registro que permita almacenar la información de las motocicletas ingresadas al taller, incluyendo datos como marca, cilindraje, kilometraje y estado actual.
2. Diseñar un apartado de gestión de propietarios para registrar y mantener actualizados los datos de contacto de los clientes vinculados a cada vehículo.
3. Desarrollar una interfaz web sencilla e intuitiva que facilite al propietario del taller el acceso, consulta y control de la información registrada de manera rápida y organizada.

Justificación del Proyecto

En la actualidad, la tecnología se ha convertido en una herramienta fundamental para optimizar la gestión de información en distintos sectores. Los talleres mecánicos de motocicletas, en muchos casos, aún mantienen sus registros de manera manual, lo que puede generar pérdida de información, errores en el control de los servicios realizados y dificultades para dar seguimiento a los vehículos de los clientes.

El desarrollo de un sistema de información web permitirá al propietario del taller llevar un control más ordenado y eficiente sobre las motocicletas que ingresan, registrando datos relevantes como marca, cilindraje, kilometraje y estado del vehículo, además de contar con la información de contacto de los clientes. Esto contribuirá a mejorar la organización, reducir errores humanos y agilizar los procesos de atención.

De esta manera, el proyecto no solo facilitará la administración interna del taller, sino que también fortalecerá la relación con los clientes al brindar un servicio más ágil, confiable y moderno. Asimismo, el uso de una herramienta tecnológica se alinea con las tendencias actuales de digitalización, fomentando la innovación en pequeñas y medianas empresas del sector automotriz.

Alcances Esperados

- El sistema permitirá registrar, consultar y gestionar información relacionada con las motocicletas y sus propietarios, permitiendo almacenar datos como marca, cilindraje, kilometraje y contactos de clientes.
- Generación de reportes básicos que faciliten el control de ingresos de vehículos al taller y que sirvan como apoyo en la toma de decisiones administrativas.
- Acceso a la información en tiempo real garantizando que el propietario del taller pueda consultar y actualizar los datos de manera inmediata desde la plataforma web.
- Escalabilidad del sistema, dejando la posibilidad de integrar en el futuro nuevos módulos o funcionalidades según las necesidades del taller, como historial de reparaciones, inventario de repuestos o control de facturación

Requerimientos del Proyecto

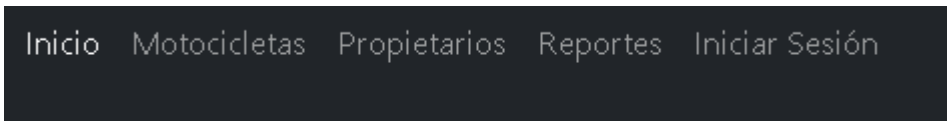
- Requerimientos funcionales: Registro de ingresos, gestión de información, reportes, etc.
- Requerimientos no funcionales: Seguridad, rendimiento, escalabilidad, usabilidad.
- Infraestructura mínima: Servidor con [especificaciones], base de datos SQL Server/MySQL, framework .NET u otro según el desarrollo.

Parte II: Diseño del Sistema (Pre-diseño visual)

El diseño presentado corresponde a un pre-diseño, es decir, una propuesta inicial de cómo se espera que luzca el sistema.

Elementos de Diseño

- Estructura de pantallas: Secciones de login, panel principal, menús de navegación, formularios de registro, reportes.



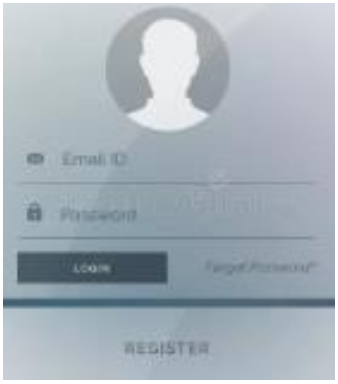
- Colores: Paleta basada en negro, gris, blanco, turquesa claro para transmitir modernidad y simplicidad.
- Tipografía: Arial.
- Imágenes: Iconografía sencilla y representativa para cada módulo.
- Íconos: Uso de librerías estándar (ej. FontAwesome, Lucide-React).
- Slogan (opcional): “**Donde la mecánica se encuentra con la precisión!** “
- Logo:

Explicación

Menú:

- **Inicio:**

La opción **Inicio/ Iniciar Sesion** es donde el propietario o empleado del taller ingresa sus credenciales. Se puede incluir validación básica y un mensaje de error si los datos no coinciden.



- **Motocicletas:**

En el botón de **Motocicletas** se permitirá registrar y visualizar motocicletas ingresadas al taller.

Registro de Motocicletas	
Marca	
Cilindraje	
Kilometraje	
Fecha de Ingreso	
Mantenimiento Requerido	

- **Propietarios**:

La opción de **Propietarios** permitirá registrar, editar o eliminar los datos de contacto de los propietarios.

Gestión de Propietarios	
Nombre Completo	
Teléfono	
Email	
Dirección	

- **Reportes:**

El botón **Reportes**, permitirá consultar reportes de ingresos o servicios realizados,

Ingreso del Admin

Reportes	
Buscar por Fecha	Desde { } Hasta { }
Buscar por N. Placa	
Buscar por Email	

Parte III: Base de Datos (MySQL)

1. Definición

La base de datos es relacional, he decidido utilizar MySQL, SQL Server, la diseñada para garantizar integridad y consistencia. Además, fundamentalmente debe almacenar, organizar, gestionar y proteger la información del sistemay se asegura que el mismo funcione de manera eficiente.

2. Elementos

'CREATE TABLE `propietarios` (

`IdPropietario` int NOT NULL AUTO_INCREMENT,

`Nombre` varchar(100) DEFAULT NULL,

`Apellido` varchar(100) DEFAULT NULL,

`Telefono` varchar(20) DEFAULT NULL,

`Email` varchar(100) DEFAULT NULL,

`Direccion` text,

PRIMARY KEY (`IdPropietario`)

) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'

'CREATE TABLE `facturas` (

`IdFactura` int NOT NULL AUTO_INCREMENT,

`IdIngreso` int DEFAULT NULL,

`FechaFactura` date DEFAULT NULL,

`MontoTotal` decimal(10,2) DEFAULT NULL,

PRIMARY KEY (`IdFactura`),

KEY `IdIngreso` (`IdIngreso`),

CONSTRAINT `facturas_ibfk_1` FOREIGN KEY (`IdIngreso`) REFERENCES `ingresos` (`IdIngreso`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'

'CREATE TABLE `ingresos` (

`IdIngreso` int NOT NULL AUTO_INCREMENT,

`FechaIngreso` date DEFAULT NULL,

`IdMotocicleta` int DEFAULT NULL,

`Observaciones` text,

PRIMARY KEY (`IdIngreso`),

KEY `IdMotocicleta` (`IdMotocicleta`),

CONSTRAINT `ingresos_ibfk_1` FOREIGN KEY (`IdMotocicleta`) REFERENCES `motocicletas` (`IdMotocicleta`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'

'CREATE TABLE `motocicletas` (

`IdMotocicleta` int NOT NULL AUTO_INCREMENT,

`Marca` varchar(50) DEFAULT NULL,

`Cilindraje` varchar(20) DEFAULT NULL,

`Kilometraje` int DEFAULT NULL,

`EstadoActual` text,

`IdPropietario` int DEFAULT NULL,

PRIMARY KEY (`IdMotocicleta`),

KEY `IdPropietario` (`IdPropietario`),

CONSTRAINT `motocicletas_ibfk_1` FOREIGN KEY (`IdPropietario`) REFERENCES `propietarios` (`IdPropietario`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'

'CREATE TABLE `reportes` (

`IdReporte` int NOT NULL AUTO_INCREMENT,

`IdMotocicleta` int DEFAULT NULL,

`Estado` varchar(50) DEFAULT NULL,

`FechaReporte` datetime DEFAULT NULL,

PRIMARY KEY (`IdReporte`),

KEY `IdMotocicleta` (`IdMotocicleta`),

CONSTRAINT `reportes_ibfk_1` FOREIGN KEY (`IdMotocicleta`) REFERENCES `motocicletas` (`IdMotocicleta`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'

'CREATE TABLE `trabajosrealizados` (

`IdTrabajo` int NOT NULL AUTO_INCREMENT,

`IdIngreso` int DEFAULT NULL,

`DescripcionTrabajo` text,

`Costo` decimal(10,2) DEFAULT NULL,

`FechaTrabajo` date DEFAULT NULL,

PRIMARY KEY (`IdTrabajo`),

KEY `IdIngreso` (`IdIngreso`),

CONSTRAINT `trabajosrealizados_ibfk_1` FOREIGN KEY (`IdIngreso`) REFERENCES `ingresos` (`IdIngreso`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'

'CREATE TABLE `usuarios` (

`IdUsuario` int NOT NULL AUTO_INCREMENT,

`NombreUsuario` varchar(50) NOT NULL,

`Contrasena` varchar(20) NOT NULL,

`Rol` enum("Mecanico","Cliente") DEFAULT "Cliente",

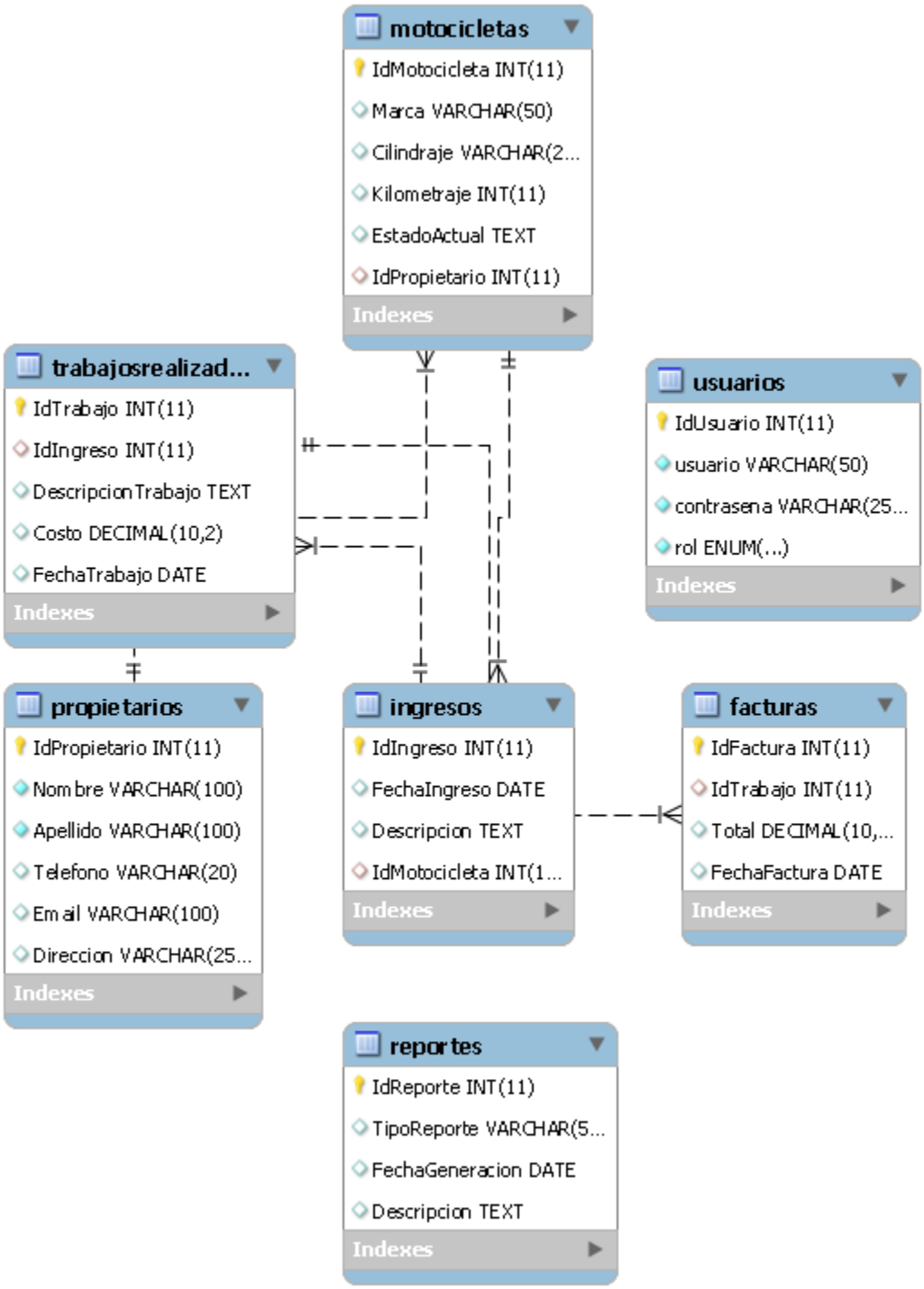
`FechaCreacion` timestamp NULL DEFAULT CURRENT_TIMESTAMP,

PRIMARY KEY (`IdUsuario`),

UNIQUE KEY `NombreUsuario_UNIQUE` (`NombreUsuario`)

) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'

3. Diagrama entidad–relación (ERD)



4. Conexión con la Base de Datos

El proyecto consiste en una aplicación web para la gestión de un taller de motocicletas (MotoGarage), desarrollada con PHP, HTML, CSS y JavaScript, que interactúa con una base de datos MySQL. Para ejecutar la aplicación localmente se utiliza XAMPP Control Panel, donde Apache interactúa con los archivos PHP/HTML y MySQL gestiona la base de datos.

La conexión entre el proyecto y la base de datos se realiza mediante PHP y la extensión MySQLi, lo que permite ejecutar consultas SQL para realizar operaciones de CRUD (crear, leer, actualizar y eliminar) en las tablas principales como `propietarios`, `motos`, `ingresos` y `trabajos`.

```
// Conexión con la BD
$conexion = new mysqli("localhost", "root", "", "proyecto");
```

Evidencia de conexión funcional en guardar_moto.php

```
// Conexión con la BD
$conexion = new mysqli("localhost", "root", "", "proyecto");

if ($conexion->connect_error) {
    echo json_encode(["status" => "error", "mensaje" => "Error de conexión
a la base de datos: " . $conexion->connect_error]);
    exit;
}
```

Evidencia de Funcionalidad:

```
$sqlProp = "SELECT IdPropietario FROM propietarios WHERE CONCAT(Nombre, '
', Apellido) LIKE ?";
$stmtProp = $conexion->prepare($sqlProp);
$busqueda = "%$propietario%";
$stmtProp->bind_param("s", $busqueda);
$stmtProp->execute();
$resultadoProp = $stmtProp->get_result();
```

Inserción o actualización de una motocicleta:

```
 $sql = "INSERT INTO motocicletas (Marca, Cilindraje, Kilometraje,
EstadoActual, IdPropietario)
            VALUES (?, ?, ?, ?, ?)";
    $stmt = $conexion->prepare($sql);
    $stmt->bind_param("ssisi", $marca, $cilindraje, $kilometraje, $estado,
$idPropietario);
    $accion = "registrada";
}

if ($stmt->execute()) {
    echo json_encode(["status" => "success", "mensaje" => " Motocicleta
$accion correctamente."]);
```

Parte IV: Desarrollo del Backend

El backend constituye la **lógica del sistema**, encargado de procesar la información, garantizar la seguridad y coordinar la comunicación entre la base de datos, el frontend y servicios externos.

1. Arquitectura del Backend

Tipo Monolítica: El proyecto está implementado como una aplicación PHP alojada en XAMPP que agrupa la lógica en scripts PHP específicos por operación (por ejemplo `guardar_moto.php`, `listar_motos.php`, `actualizar_moto.php`, `eliminar_moto.php`, `guardar_propietario.php`, `ingresos.php`,`etc`.).

2. Componentes y Estructura del Codigo

- Descripción de las clases, controladores, modelos, servicios y su función dentro del sistema.

  El backend del sistema se construyó utilizando PHP como lenguaje principal, adoptando una estructura modular basada en archivos independientes que encapsulan acciones específicas —registro, actualización, consulta y eliminación— para cada entidad del sistema. Si bien no se utiliza un framework MVC formal, la organización del código sigue una lógica equivalente: algunos archivos actúan como “controladores”, otros como “modelos” simples (operaciones sobre la base de datos), y ciertos archivos funcionan como interfaz de presentación conectada al frontend HTML.

  A continuación, se describe cada tipo de componente presente en el proyecto y su función dentro de la arquitectura general:

  2.1 Controladores (Archivos de acción PHP):

  Estos archivos se encargan de recibir las solicitudes del usuario (normalmente mediante formularios HTML), procesar datos, realizar validaciones básicas y ejecutar las consultas SQL correspondientes. Su función es equivalente a la capa Controller en un modelo MVC tradicional.

  a) Módulo de Propietarios

  `guardar_propietario.php`

  Registra un nuevo propietario en la base de datos. Recibe los datos desde `propietarios.php`, valida campos obligatorios e inserta el registro.

`actualizar_propietario.php`

Procesa la edición de un propietario existente. Actualiza nombre, teléfono u otros atributos.

`eliminar_propietario.php`

Ejecuta la eliminación del registro según su identificador.

`editar_propietario.php`

Obtiene los datos del propietario para mostrarlos en un formulario editable.

`listar_propietarios.php`

Recupera la lista completa de propietarios utilizando una consulta SQL. Funciona como "puente" para que el frontend muestre la tabla.

b) Módulo de Motocicletas

`guardar_moto.php, editar_moto.php, actualizar_moto.php, eliminar_moto.php, listar_motos.php`

Cumplen funciones equivalentes a las del módulo de propietarios, pero aplicadas a las motocicletas registradas en el sistema. Algunas operaciones incluyen relaciones con propietarios mediante `IdPropietario`.

c) Módulo de Ingresos

`guardar_ingreso.php`

Inserta un nuevo ingreso de motocicleta al taller. Valida que existan fecha y motocicleta, y registra la descripción.

`editar_ingreso.php`

Obtiene un ingreso específico, presenta el formulario y permite modificarlo.

`eliminar_ingreso.php`

Elimina un ingreso según su ID.

`listar_ingresos.php`

Genera la lista completa de ingresos para visualizar en `ingresos.php`.

d) Autenticación y Sesiones

`login.php`

Verifica credenciales mediante consulta SQL y crea la sesión de usuario. Utiliza validaciones simples y redirige según el rol (cliente o mecánico/administrador).

`logout.php`

Destruye la sesión activa y devuelve al usuario al inicio.

`registro_usuario.php`

Crea nuevas cuentas de usuario para el sistema.

2.2. Modelos (Consultas y entidades)

En este proyecto no se implementan clases formales de modelo; en su lugar, la "capa de datos" se representa mediante:

Consultas SQL preparadas (prepare, bind_param, execute) y validaciones de entrada.

Manipulación directa de tablas como propietarios, motocicletas, ingresos, usuarios.

Cada archivo controlador actúa como modelo y controlador a la vez, integrando la lógica de negocio con la interacción directa con la base de datos.

2.3. Servicios

Aunque no existen servicios como componentes separados (por ejemplo, en una arquitectura MVC estricta), algunas funcionalidades cumplen un rol equivalente:

`navbar_cliente.php y navbar_mecanico.php`

Funcionan como módulos reutilizables que componen la interfaz del sistema dependiendo del rol.

`reportes.php`

Centraliza la generación y consulta de reportes, actuando como servicio de consulta avanzada.

`factura.php`

Compone y entrega la factura basada en datos recuperados de diferentes tablas.

2.4. Estructura general del backend

La interacción se puede resumir de la siguiente manera:

Usuario → Formulario HTML → Archivo controlador PHP → Consulta SQL → Respuesta (HTML o redirección)

Este flujo permite que el sistema funcione de manera estable sin necesidad de un framework complejo, manteniendo un enfoque modular y fácil de mantener.

- Diagrama general de flujo o interacción entre capas (MVC, API REST, etc.).

  El backend del sistema funciona bajo un modelo basado en scripts PHP organizados por funciones (registro, edición, eliminación, listado), que en conjunto siguen el patrón Cliente → Controlador PHP → Base de Datos MySQL → Respuesta al Cliente. Aunque no existe una separación estricta en capas como en un framework MVC, el flujo sí estructura las operaciones en pasos claramente definidos.

A continuación, se detlla el flujo general del sistema:

1. Interacción del Usuario (Frontend)

El proceso inicia cuando el usuario interactúa con alguna de las vistas HTML, tales como:

`propietarios.php`

`motocicletas.php`

`ingresos.php`

`login.php`

`registro.php`

`factura.php`

Cada vista contiene formularios HTML que envían datos mediante métodos POST o GET a los scripts del backend.

Ejemplos:

El botón Guardar Propietario envía datos a guardar_propietario.php

El botón Editar Moto envía datos a editar_moto.php

2. Recepción y Validación de Datos (Controladores PHP)

Cada módulo está compuesto por scripts que cumplen la función de controladores, ya que reciben la solicitud y ejecutan la lógica correspondiente.

Ejemplos:

`guardar_propietario.php`

`guardar_moto.php`

`guardar_ingreso.php`

`editar_propietario.php`

`editar_moto.php`

`editar_ingreso.php`

`login.php`

`registro_usuario.php`

Cada script realiza las siguientes tareas:

Recibe datos vía $_POST o $_GET.

Valida campos obligatorios (campos vacíos, formatos incorrectos, etc.).

Establece conexión a la base de datos mediante new mysqli(...).

Prepara consultas SQL seguras usando prepare() y bind_param() para evitar inyección SQL.

Ejecuta la operación (INSERT, UPDATE, DELETE o SELECT).

Redirige con estado mediante parámetros msg=ok, msg=error_bd, etc.

3. Conexión y Procesamiento en la Base de Datos MySQL

Cada operación del backend se conecta a la base de datos del proyecto y accede a tablas como:

```
propietarios
motocicletas
ingresos
usuarios
```

Cada script ejecuta operaciones SQL específicas:

INSERT → para crear registros

UPDATE → para editar

DELETE → en las funciones de eliminación

SELECT → para listar o cargar información

Ejemplo real:

```
$sql = "INSERT INTO ingresos (FechaIngreso, IdMotocicleta,
Descripcion)
    VALUES (?, ?, ?)";
```

4. Generación de Respuestas y Retroalimentación al Usuario

Los controladores devuelven las respuestas al usuario de dos maneras:

a) Redirección con mensajes (flujo tradicional)

Muy utilizado en ingresos, propietarios y motocicletas:

header("Location: ingresos.php?msg=ok");

Esto permite que la interfaz muestre:

✓ Registro exitoso

❌ Error al guardar

❌ Campos incompletos

b) Respuestas JSON (flujo AJAX)

Algunas funciones emplean salida JSON para integrarse con JavaScript:

Ejemplo: `echo json_encode(["status" => "success", "mensaje" => "Ingreso registrado correctamente."]);`

Esta acción permite que el frontend actualice contenido sin recargar la página.

5. Actualización Dinámica del Frontend

La interfaz obtiene los datos mediante:

Listados (tabla HTML generada en PHP)

`listar_propietarios.php`

`listar_motos.php`

`listar_ingresos.php`

Ejemplo de fila generada dinámicamente:

```
<td><?= $fila['Nombre'] ?></td>
<td><a href="editar_propietario.php?id=<?=
$fila['IdPropietario'] ?>">Editar</a></td>
```

Actualización por AJAX

Para algunas operaciones, el mensaje se muestra sin recargar la página, como en: `mostrarMensaje("success", "Ingreso registrado correctamente.");`

3. Elementos Clave del Backend

- Seguridad: Manejo de sesiones, autenticación y autorización.

  Aunque el proyecto aún no implementa un módulo complejo de autenticación, sí incorpora medidas básicas de control y seguridad compatibles con aplicaciones PHP tradicionales:

  - Manejo de Sesiones

  El sistema utiliza sesiones de PHP (session_start()) para conservar información del usuario en módulos donde se requiere restringir el acceso.

  Esto permite mantener el estado del usuario mientras navega entre módulos y evitar la manipulación directa de URLs para acceder a funciones que no corresponden.

  - Validación de Formularios

  Cada formulario del sistema —tanto para propietarios, motocicletas, ingresos, ediciones, etc.— valida que ningún campo obligatorio se envíe vacío; además, que los datos tengan el formato adecuado y que los valores numéricos (como cilindraje o teléfono) sean válidos.

  Si ocurre una omisión o error, se redirige al usuario mediante mensajes como:

  `?msg=error_campos`

  `?msg=error_bd`

  - Protección básica contra inyecciones SQL

Todos los módulos de inserción y edición utilizan sentencias preparadas ($stmt->bind_param()), lo cual protege contra ataques de inyección SQL.

Ejemplo real del sistema:

```
$sql = "INSERT INTO ingresos (FechaIngreso, IdMotocicleta,
Descripcion)
        VALUES (?, ?, ?)";
$stmt = $conexion->prepare($sql);
$stmt->bind_param("sis", $fecha, $id_moto, $obs);
```

- Gestión de Errores: Mecanismos de logging y manejo de excepciones.

  El backend incorpora varios mecanismos que permiten identificar fallos y prevenir caídas generales.

  - Manejo de errores con redirecciones.

  Cuando ocurre un error en la base de datos o en la validación, el backend devuelve indicadores a través de parámetros GET:

  ```
  ?msg=error_bd
  ?msg=notfound
  ?msg=edited
  ```

  Estos mensajes se muestran al usuario mediante alertas Bootstrap bien definidas.

  - Validación exhaustiva en cada operación

  Cada módulo del backend verifica:

  Existencia de IDs antes de editar o eliminar.

  Que los registros existan antes de operar sobre ellos.

  Que los valores enviados por POST no estén vacíos.

  Ejemplo real:

  ```
  if (!isset($_GET['id'])) {
      header("Location: ingresos.php?msg=error_id");
      exit;}
  ```

- Logging implícito mediante MySQL y mensajes de depuración

Aunque el sistema no utiliza un archivo dedicado de logs, MySQL registra:

Consultas fallidas, errores de integridad y violaciones de llaves foráneas; lo cual facilita la depuración durante el desarrollo.

- Servicios Externos: El sistema actual no requiere conectarse a APIs externas para su funcionamiento.
  Sin embargo, su arquitectura permite integrar servicios adicionales en versiones futuras, por ejemplo:
    - APIs de WhatsApp o correo para notificar al propietario.
    - APIs de facturación electrónica
    - Servicios de validación de VIN o datos de motocicletas.

El diseño actual basado en módulos PHP independientes facilita agregar este tipo de integraciones sin afectar el core del sistema.

- Pruebas: Aunque el proyecto no utiliza frameworks de testing automatizado como PHPUnit, sí implementa pruebas manuales documentadas durante el desarrollo:
    - Pruebas Unitarias Manuales
      Se evaluaron individualmente las funciones CRUD:
        - Insertar propietario / motocicleta / ingreso
        - Actualizar registros
        - Borrar registros
        - Validar que los datos se almacenen correctamente
    - Pruebas de Integración
      Se verificó la comunicación entre:
        - Formularios HTML
        - Scripts PHP
        - Base de datos MySQL
    - Pruebas Funcionales
      Se probaron escenarios completos del sistema, por ejemplo:
        - Registrar propietario → Registrar motocicleta → Registrar ingreso.

        - Editar un ingreso y verificar actualización en la BD.
        - Eliminar un registro y asegurar que desaparece de la lista.

Estas pruebas permiten garantizar que el backend responde correctamente a las necesidades del taller.

- Escalabilidad: El sistema ha sido diseñado siguiendo prácticas que permiten su ampliación futura:
    - Uso de arquitectura modular

      Cada módulo (propietarios, motocicletas, ingresos) está contenido en archivos independientes. Esto permite:
        - Agregar nuevos componentes sin afectar los existentes.
        - Mantener un código ordenado y fácil de extender.
    - Uso de sentencias preparadas

      Además de la seguridad, estas consultas optimizan el rendimiento en cargas más altas.
    - Base de datos estructurada con relaciones

      Las tablas poseen:
    - Llaves foráneas
    - Integridad referencial
    - Índices en campos clave

Esto asegura eficiencia incluso cuando los registros crezcan.

    - Facilidad de migrar a MVC

      La separación entre HTML, PHP y SQL permite que el sistema pueda reestructurarse en un futuro hacia:
    - MVC (Model–View–Controller)
    - API REST con endpoints reales
    - Frameworks modernos (Laravel, Spring Boot, Node.js)

4. Conexión con la base de datos y el frontend:

- Código o evidencia funcional de la conexión con la base de datos definida en la Parte III.

  El sistema MotoGarage utiliza MySQL como motor de base de datos, administrado mediante XAMPP, donde Apache ejecuta el backend en PHP y MySQL procesa las consultas.

  La comunicación entre el backend y la base de datos se implementa mediante la extensión MySQLi orientada a objetos, lo que permite mantener un control seguro y estructurado de las operaciones CRUD.

  La forma estándar y centralizada de conexión utilizada en todos los módulos se basa en la siguiente instrucción:

```
$conexion = new mysqli("localhost", "root", "",
"proyecto");
```

  Cada archivo del backend valida inmediatamente si la conexión fue exitosa:

```
if ($conexion->connect_error) {
    die("Error de conexión con la BD");
}
```

  Esta estructura garantiza que cualquier falla de acceso a MySQL detiene la ejecución y previene errores encadenados.

- Evidencia funcional: `guardar_moto.php`

  Este archivo muestra un ejemplo claro de cómo el sistema se conecta correctamente y ejecuta consultas SQL preparadas:

```
$conexion = new mysqli("localhost", "root", "", proyecto");
if ($conexion->connect_error) {
    echo json_encode([
        "status" => "error",
```

```
        "mensaje" => "Error de conexión a la base de datos:
" . $conexion->connect_error ]);

    exit;

}
```

Se incluye además el uso de prepared statements, una buena práctica que previene inyección SQL, mejora la seguridad del sistema y garantiza integridad en los datos enviados.

Búsqueda y validación previa:

```
$sqlProp = "SELECT IdPropietario

            FROM propietarios

            WHERE CONCAT(Nombre, ' ', Apellido) LIKE ?";

$stmtProp = $conexion->prepare($sqlProp);

$busqueda = "%$propietario%";

$stmtProp->bind_param("s", $busqueda);

$stmtProp->execute();

$resultadoProp = $stmtProp->get_result();
```

Esto demuestra cómo el backend verifica la existencia del propietario antes de permitir registrar la motocicleta.

- Inserción de motocicletas:
  Cuando la validación se completa, el sistema procede a registrar la moto:

```
$sql = "INSERT INTO motocicletas
        (Marca, Cilindraje, Kilometraje, EstadoActual,
IdPropietario)
        VALUES (?, ?, ?, ?, ?)";

$stmt = $conexion->prepare($sql);
```

```
$stmt->bind_param("ssisi", $marca, $cilindraje,
$kilometraje, $estado, $idPropietario);
```

Cuando la validación se completa, el sistema procede a registrar la moto:

- . Endpoints o rutas disponibles para la comunicación con el frontend: aunque el sistema utiliza PHP clásico (no un framework MVC), cada archivo PHP funciona conceptualmente como un endpoint que ejecuta una acción específica dentro del backend. A continuación, se presenta una clasificación organizada por módulos.

  a) Módulo de Propietarios

Archivo/Endpoint	Método	Función
propietarios.php	GET	Muestra la lista de propietarios.
guardar_propietario.php	POST	Registra nuevos propietarios vía formulario.
editar_propietario.php	GET/POST	Carga datos del propietario y actualiza la información.
eliminar_propietario.php	GET	Elimina un propietario por su ID.

  b) Módulo de Motocicletas

Archivo/Endpoint	Método	Función
motocicletas.php	GET	Lista de motocicletas registradas.
guardar_moto.php	POST	Registra o actualiza una motocicleta mediante JSON.
eliminar_moto.php	GET	Borra la moto seleccionada.

c) Módulo de Ingresos al Taller (implementa correctamente la relación entre Motocicletas e ingresos,utilizando claves foráneas.

Archivo/Endpoint	Método	Función
ingresos.php	GET	Formulario para registrar ingresos + tabla de historial.
guardar_ingreso.php	POST	Inserta un nuevo ingreso al taller.
editar_ingreso.php	GET/POST	Modifica información del ingreso.
eliminar_ingreso.php	GET	Elimina un ingreso específico.

d) Módulo Facturación

Archivo/Endpoint	Método	Función
facturas.php	GET	Muestra facturas generadas.
generar_factura.php	POST	Crea una factura asociada a un ingreso.

e) Módulo de Trabajos Realizados

Archivo/Endpoint	Método	Función
trabajos.php	GET	Lista trabajos realizados
generar_trabajo.php	POST	Registra un trabajo vinculado a un ingreso.

f) Autenticación y Usuarios

Archivo	Método	Función
login.php	POST	Verifica credenciales.
logout.php	GET	Cierra sesión destruyendo las variables.
usuarios.php	GET	Vista para administración de usuarios.

El backend envía información al frontend mediante tres mecanismos:

- Renderizado directo (clásico PHP + HTML) para páginas completas como propietarios, ingresos, motos.
- AJAX con JSON, usado especialmente en el registro y edición de motocicletas.

- Redirecciones con parámetros GET, por ejemplo: ingresos.php?msg=ok

El frontend interpreta estos mensajes para mostrar alertas Bootstrap dinámicas.

5. Capturas o fragmentos de código:

- Mostrar ejemplos del código implementado (controladores, rutas, consultas SQL, validaciones, etc.).
- Describir brevemente el funcionamiento de cada módulo clave.

a. Conexión a la Base de Datos

Este fragmento aparece en diversos archivos del sistema (por ejemplo: guardar_moto.php, guardar_propietario.php, guardar_ingreso.php). Define la conexión general al motor MySQL utilizando la extensión MySQLi.

```
// Conexión con la BD

$conexion = new mysqli("localhost", "root", "", "proyecto");

if ($conexion->connect_error) {

    die("Error de conexión con la BD");

}
```

Descripción:

Establece el enlace entre el backend PHP y la base de datos MySQL.; además, valida si ocurrió un error durante la conexión y ess un punto crítico para garantizar la disponibilidad del sistema.

b. Ejemplo de Controlador: Registro de una Motocicleta

```
// Consulta para buscar propietario
$sqlProp = "SELECT IdPropietario FROM propietarios
            WHERE CONCAT(Nombre, ' ', Apellido) LIKE ?";
$stmtProp = $conexion->prepare($sqlProp);

$busqueda = "%$propietario%";
$stmtProp->bind_param("s", $busqueda);
$stmtProp->execute();
$resultadoProp = $stmtProp->get_result();
```

Descripción:

- Se busca un propietario utilizando coincidencia parcial (LIKE); además, se emplean consultas preparadas para evitar inyección SQL.
- Es parte de la lógica de negocio que garantiza que una moto se registre siempre con un propietario válido.

c. Ejemplo de Módulo CRUD: Registrar un Ingreso:

```
$fecha = $_POST['fecha_ingreso'] ?? null;

$id_moto = $_POST['id_motocicleta'] ?? null;

$obs = trim($_POST['observaciones'] ?? "");

// Validación

if (!$fecha || !$id_moto) {

    header("Location: ingresos.php?msg=error_campos");

    exit;

}
```

Descripción:

- Valida campos obligatorios antes de registrar un ingreso.
- Previene envíos incompletos y fortalece la integridad del sistema.

Inserción del ingreso

```
$sql = "INSERT INTO ingresos (FechaIngreso, IdMotocicleta,
Observaciones)

        VALUES (?, ?, ?)";

$stmt = $conexion->prepare($sql);
```

```
$stmt->bind_param("sis", $fecha, $id_moto, $obs);

if ($stmt->execute()) {

    header("Location: ingresos.php?msg=ok");}
```

Descripción:

- Registra un ingreso al taller.
- Utiliza parámetros tipados (s string, i integer).
- Redirige según el resultado de la operación.

d. Ejemplo de Módulo de Actualización: Editar Ingreso

Consulta del ingreso a modificar:

```
$sql = "SELECT * FROM ingresos WHERE IdIngreso = ?";

$stmt = $conexion->prepare($sql);

$stmt->bind_param("i", $id);

$stmt->execute();

$ingreso = $stmt->get_result()->fetch_assoc();
```

Descripción:

- Recupera los valores actuales del registro para precargar el formulario.
- Evita acceder a registros inexistentes mediante validaciones.

Actualización del registro:

```
$sqlUpdate = "UPDATE ingresos

              SET FechaIngreso=?, IdMotocicleta=?,
Observaciones=?

              WHERE IdIngreso=?";
```

```
$stmt2 = $conexion->prepare($sqlUpdate);

$stmt2->bind_param("sisi", $fecha, $moto, $obs, $id);

$stmt2->execute();
```

Descripción:

- Actualiza la información del ingreso seleccionado.
- Implementa lógica de validación y retorno de mensajes para el usuario.

Conclusión

El proyecto representa un avance significativo en la automatización de procesos y la gestión eficiente de información. A través de la definición de objetivos claros, un pre-diseño visual, una base de datos sólida y una lógica backend bien estructurada, se garantiza un sistema adaptable, seguro y escalable.

Se concluye que el enfoque utilizado permite no solo cumplir los objetivos planteados, sino también dejar las bases para futuras mejoras y nuevas funcionalidades que fortalezcan la solución tecnológica.

Referencias

https://www.youtube.com/watch?v=q6RKq91FKC4

https://www.youtube.com/watch?v=wZniZEbPAzk

https://www.youtube.com/watch?v=MJkdaVFHrto

https://www.youtube.com/watch?v=STNnG5K3jdM

https://www.php.net/manual/es/tutorial.firstpage.php

https://www.freecodecamp.org/espanol/news/los-mejores-ejemplos-de-php/

https://www.youtube.com/watch?v=NrE13WsPihs

https://www.w3schools.com/php/php_mysql_create.asp