**DOCUMENTO DE MANUAL TÉCNICO**

Curso: Construcción de Software SOF-S-MA-6-3

GRUPO C

INTEGRANTES

Soto Monar Jeremy David.

Delgado Pilay Justin Gabriel.

Montalvo Sandoval Sabina Isabel.

Mota Magallanes Ronald Steven.

Quinde Aspiazu Nicolás Ismael.

**Tabla de contenido**

[1 Introducción 3](#_Toc201853606)

[1.1 Propósito del Documento 3](#_Toc201853607)

[1.2 Público Objetivo 3](#_Toc201853608)

[2 Tecnologías Utilizadas 3](#_Toc201853609)

[3 Presentación del Producto 4](#_Toc201853610)

[3.1 Objetivo 4](#_Toc201853611)

[3.1.1 Objetivos Específicos: 4](#_Toc201853612)

[3.2 Alcance 5](#_Toc201853613)

[3.3 El Sistema no contempla 5](#_Toc201853614)

[4 Arquitectura del Sistema 6](#_Toc201853615)

[4.1 Diagrama de Paquetes. 6](#_Toc201853616)

[4.2 Diagrama de Despliegue 9](#_Toc201853617)

[5 Estructura de Carpetas del Sistema 10](#_Toc201853618)

[6 Base de Datos 11](#_Toc201853619)

[6.1 Diagrama Entidad Relación 11](#_Toc201853620)

[6.2 Tablas Principales 12](#_Toc201853621)

[7 Configuración del Entorno 13](#_Toc201853622)

[8 Explicación del Código 13](#_Toc201853623)

[8.1 ¿Qué hace cada controlador? 13](#_Toc201853624)

[8.1.1 Controlador: libros\_controlador.php 13](#_Toc201853625)

[8.1.2 Controlador: existencias\_controlador.php 17](#_Toc201853626)

[8.1.3 Controlador: prestamos\_controlador.php 21](#_Toc201853627)

[8.1.4 Controlador: lector\_controlador.php 23](#_Toc201853628)

[8.1.5 Controlador: bibliotecario\_controlador.php 27](#_Toc201853629)

[8.1.6 Controlador: cubiculo\_controlador.php 30](#_Toc201853630)

[8.1.7 Controlador: control\_controlador.php 33](#_Toc201853631)

[8.2 ¿Qué funciones hay en cada modelo? 36](#_Toc201853632)

[8.2.1 Modelo: libros\_modelo.php 36](#_Toc201853633)

[8.2.2 Modelo: existencias\_modelo.php 38](#_Toc201853634)

[8.2.3 Modelo: préstamo\_modelo.php 40](#_Toc201853635)

[8.2.4 Modelo: lector\_modelo.php 42](#_Toc201853636)

[8.2.5 Modelo: bibliotecario\_modelo.php 44](#_Toc201853637)

[8.2.6 Modelo: cubículo\_modelo.php 46](#_Toc201853638)

[8.2.7 Modelo: control\_modelo.php 48](#_Toc201853639)

[9 Errores Comunes y Soluciones 50](#_Toc201853640)

[10 Anexos 52](#_Toc201853641)

# Introducción

El Sistema de Gestión de Biblioteca es una solución informática diseñada para automatizar los procesos administrativos de una biblioteca. Este sistema permite registrar y administrar información relacionada con libros, lectores, cubículos, préstamos, devoluciones, bibliotecarios y multas. Su implementación representa una mejora significativa en la eficiencia y organización de tareas que suelen ser repetitivas y tediosas.

## Propósito del Documento

El propósito de este documento es proporcionar una visión general de la construcción del sistema, detallando su arquitectura, funcionalidades principales y diseño de base de datos. Además, tiene como objetivo documentar el diseño del sistema para facilitar la comprensión de sus componentes, así como servir de referencia ante posibles errores o futuras implementaciones de nuevos módulos.

## Público Objetivo

Este documento está dirigido a desarrolladores, administradores del sistema, administradores de bases de datos y cualquier persona interesada en comprender el diseño e implementación del sistema.

# Tecnologías Utilizadas

Para el desarrollo del Sistema de Gestión de Biblioteca se emplearon las siguientes tecnologías:

*Lenguajes de Programación*

* **HP:** Utilizado para el desarrollo del backend del sistema. Se optó por trabajar con PHP puro, sin el uso de frameworks.
* **JavaScript:** Usado en el desarrollo del frontend, así como en la conexión e interacción con el backend.

*Diseño Frontend*

* **HTML5:** Estructura principal de las interfaces del sistema.
* **CSS3:** Encargado del diseño visual y estilos del HTML.
* **JavaScript:** Implementado para agregar dinamismo, animaciones y funcionalidades interactivas a los botones y ventanas.

*Base de Datos*

* **MySQL 8:** Sistema de gestión de base de datos utilizado para almacenar y organizar la información del sistema.

*Servidor Web*

* **Apache** (a través de XAMPP v3.3.0): Utilizado como entorno local de desarrollo y servidor web para ejecutar el sistema.

*Entorno de Desarrollo*

* **Visual Studio Code (VS Code):** Editor de código empleado durante el desarrollo del sistema.

# Presentación del Producto

## Objetivo

Diseñar e implementar un sistema informático de automatización de procesos para la gestión de la biblioteca, que permita registrar y administrar libros, lectores, cubículos, préstamos, devoluciones, bibliotecarios y multas, mejorando la productividad y eficacia mediante la automatización de tareas y consultas.

### Objetivos Específicos:

Automatizar la gestión de préstamos y devoluciones mediante registros automatizados de transacciones para mejorar el flujo de trabajo

Optimizar la Productividad del personal, reduciendo los tiempos y el esfuerzo en la realización de tareas repetitivas, mediante la automatización de procesos clave.

Brindar un sistema que centralice el almacenamiento de los diferentes procesos

## Alcance

Nuestra solución permitirá la implementación de 6 módulos:

1. ***Inicio de Sesión:*** Modulo encargado en la autenticación de los usuarios que utilizan el sistema, también se encarga de la recuperación de las contraseñas.
2. ***Gestión de Libros:*** Modulo encargado para la gestión de todos los libros que se encuentran en la biblioteca, además de gestionar las existencias de los ejemplares de cada libro.
3. ***Gestión de Usuarios:*** Modulo encargado para la creación, eliminación y modificación de usuarios bibliotecarios y lectores.
4. ***Gestión de Prestamos:*** Modulo encargado de la realización de préstamos de libros.
5. ***Gestión de Cubículos:*** Modulo encargado de la visualización, creación, modificación y eliminación de los cubículos que se encuentran en la biblioteca.
6. ***Gestión y Control:*** Modulo encargado de llevar el control de las multas realizadas a los usuarios lectores. Además, es el encargado de revocar o colocar permisos a los usuarios lectores tales permisos de bloqueo o desbloqueo de lector.

## El Sistema no contempla

El sistema de la Biblioteca no implementará:

Gestión de empleados: Funcionalidades para el control de los bibliotecarios: control de horas trabajadas, programación de empleados, horarios etc.

Integración con otros sistemas: No incluirá la integración con sistemas externos.

Gestión de Idiomas: Funcionalidades para la gestión del Idioma.

Registro de usuarios: No incluirá modulo para registro de usuarios de manera autónoma.

Gestión de digitalización: No incluirá módulos para la digitalización de un libro físico en uno digital.

Análisis de Datos: No implementará análisis estadísticos de la biblioteca.

Gestión de libros electrónicos: No será compatible para el préstamo de libros electrónicos.

# Arquitectura del Sistema

## Diagrama de Paquetes.

El diseño del sistema desarrollado, respeta el modelo arquitectónico de MVC (Modelo, Vista, Controlador). Al utilizar este tipo de arquitectura de capas nosotros conseguimos en el proyecto una separación clara y concisa de las responsabilidades además nos permite crear un proyecto que sea escalable y fácil de mantener.

***Modelo***

El modelo para nuestro proyecto es la que se encarga del almacenamiento de todos los datos del sistema, es decir, guarda toda la información relacionada con: Usuarios, Libros, multas, reservas, devoluciones, existencias y otros. Para esta capa, nosotros utilizaremos el sistema de gestión de Base de Datos MySql como nuestro gestor de bases de datos, esto es debido a su gran escalabilidad, robustez y el soporte de las diferentes consultas que se realizan para nuestro sistema.

*Responsabilidades:*

* Ejecución de y transacción de diferentes Procesos almacenados.
* Gestión para la conexión a la Base de Datos.
* Almacenamiento y recuperación de Datos.
* Mantenimiento de los diferentes Datos.

*Controlador*

El controlador, como su nombre lo indica, es el encargado de servir como controlador entre la vista y el modelo. Además, se encarga de manejar las operaciones relacionadas con los datos recibidos de las VISTAS o los datos que se envían al Modelo

*Responsabilidades:*

* Controlar la lógica y las reglas del negocio
* Controlar la navegación del sistema.
* Intermediar entre el flujo de la capa de datos y capa de presentación.

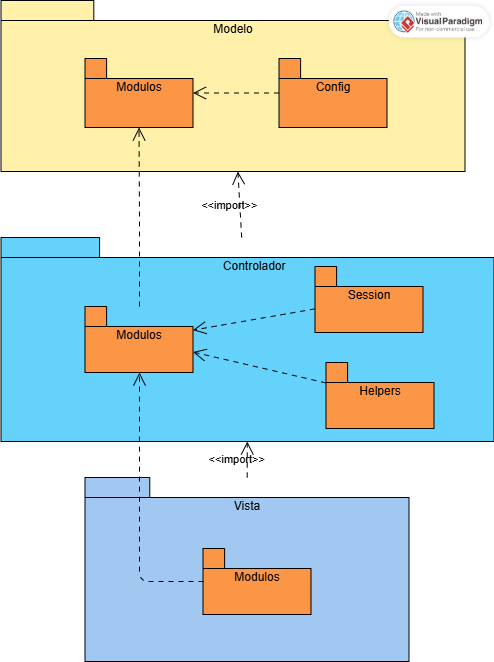
*Vista*

La vista es la encargada de contener, visualizar la interfaz del sistema, con la cual todos los usuarios del sistema interactuaran.

Responsabilidades:

* Visualizar Interfaces para el usuario del sistema.
* Permitir que los usuarios realicen operaciones en el sistema.
* Mostrar todos los datos de manera intuitiva.

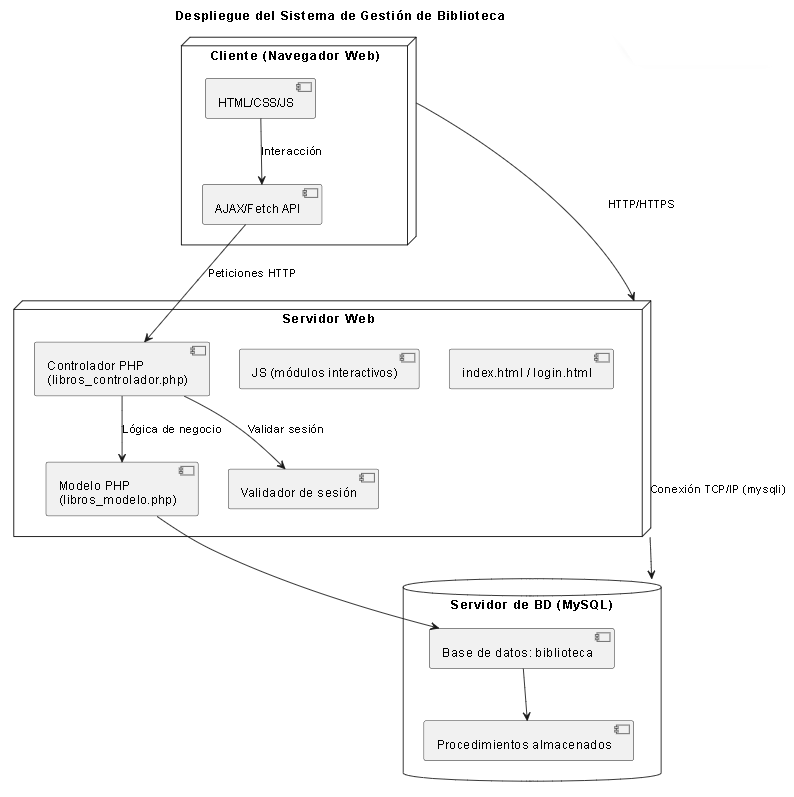
La ventaja de este tipo de arquitecturas es que las aplicaciones presentan modularidad, separación de responsabilidades y reutilización de datos. Asegura la escalabilidad y alta disponibilidad.



*Diagrama de Paquetes del Sistema de Biblioteca*

## Diagrama de Despliegue

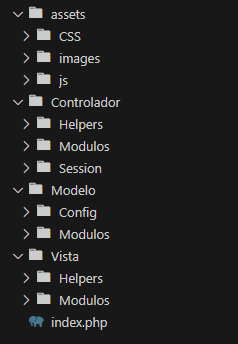
Se describe el siguiente diagrama de despliegue del Sistema de Gestión de Biblioteca, el cual al realizar un despliegue tiene la estructura Cliente-Servidor.



*Diagrama de Despliegue del Sistema de Biblioteca*

# Estructura de Carpetas del Sistema

El Sistema desarrollado en VS CODE, tiene la siguiente estructura:



***Donde:***

***Assets:*** Esta carpeta es la encargada de contener todas las imágenes, css y js del sistema. Por ejemplo: estilos.css, script.css y banner.png

***Modelo:*** Encargado de contener la conexión a la Base de Datos del Sistema. Además, contiene todas las clases de los módulos del sistema que hacen inserción a la base de datos. Por ejemplo: Conexión.php y Libros\_modelo.php

***Vista:*** Encargado de contener todas las ventanas del sistema. Ejemplo: Libros.php

***Controlador:*** Encargado de contender todas las clases que sirven de conexión entre la vista y el modelo, además contiene los helpers que son el manejo de excepciones del sistema y por último contiene la Sesión del sistema. Libros\_controlador.php, libroexepcion.php, LibroValidatorServices.php y Session.php.

# Base de Datos

## Diagrama Entidad Relación

## Tablas Principales

* ***Libro. -*** Encargado de contener toda la información de un libro con: título, autor, año de publicación, cantidad de páginas y el género.
* ***Existencias. -*** Encargado de almacenar todas las existencias asociadas a un libro registrado en la tabla de Libro. Contiene información como: ubicación, disponibilidad, estado y libro que pertenece.
* ***Lector. –*** Encargado de almacenar toda la información de los lectores de la biblioteca con: nombres, apellidos, cédula, teléfono, dirección y correo electrónico.
* ***Bibliotecario. –*** Encargado de almacenar toda la información de un bibliotecario, es importante ya que aquí se almacenan las contraseñas del bibliotecario y del administrador del sistema. La información a guardar es: nombres, apellidos, cédula, email, nombre de usuario y contraseña.
* ***Cubículo. -*** Encargado de almacenar toda la información que tiene un cubículo como: nombre del cubículo, equipamiento que posee y la capacidad de personas que puede albergar.
* ***Préstamo. -*** Encargado de almacenar toda la información de un préstamo de libro, cuenta con la siguiente información: fecha del préstamo, fecha de la devolución.
* ***Multa. –*** Encargado de Almacenar toda la información de las multas realizadas por los bibliotecarios que los lectores tienen, llevan la información de: fecha de emisión de la multa, el motivo, el valor a cancelar, el estado de las multas y la existencia que dañaron.

# Configuración del Entorno

Pasos para Ejecutar el Sistema:

1.- Instalar XAMPP

2.- Clonar el Proyecto en htdcos.

3.- Crear la Base de Datos utilizando el Archivo “gestionbibliote.sql”.

5.- Acceder mediante navegador a “localhost/Gestion-Biblioteca”

Para más información revisar los Anexos de este Documento el apartado de Manual para abrir el Proyecto.

# Explicación del Código

## ¿Qué hace cada controlador?

Los controladores en el sistema cumplen la función de servir de conectores entre la vista y el modelo para hacer el registro de algún dato o la consulta de datos. Veamos que hace cada controlador a continuación:

### Controlador: libros\_controlador.php

Este controlador gestiona todas las operaciones relacionadas con los libros en el sistema, incluyendo creación, modificación, eliminación, cambio de estado y búsqueda de libros. También maneja la recuperación de datos para mostrar en la vista correspondiente.

**Funcionalidades Principales**

1. Obtención de datos de libros
2. Procesamiento de formularios para operaciones CRUD
3. Manejo de excepciones específicas

***Funciones Principales***

obtenerDatosLibros()

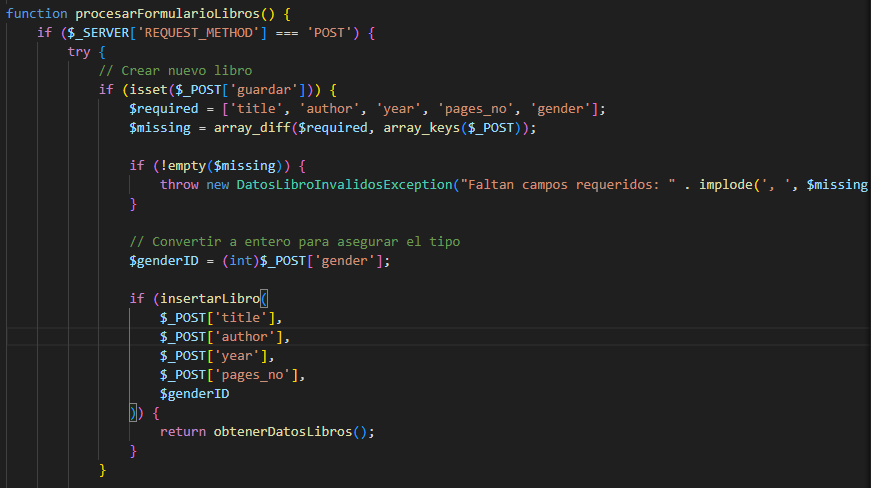


Propósito: Recupera todos los datos necesarios para la vista de libros

Retorna:

* Array con tres listas:
  + libros: Todos los libros disponibles
  + librosDeshabilitados: Libros marcados como deshabilitados
  + librosHabilitados: Libros marcados como habilitados
* Manejo de errores:
  + Si no encuentra libros, retorna arrays vacíos
  + Propaga otras excepciones

procesarFormularioLibros()



Propósito: Procesa todas las acciones enviadas por formulario POST

* Acciones manejadas:

1. Crear nuevo libro (cuando se envía 'guardar')
   * Campos requeridos: title, author, year, pages\_no, gender
   * Valida que todos los campos estén presentes
2. Modificar libro existente (cuando se envía 'modificar')
3. Eliminar libro (cuando se envía 'eliminarID')
4. Cambiar estado (cuando se envía 'estadoID' y 'nuevoEstado')
5. Buscar libros (cuando se envía 'buscar')

* Manejo de errores:

Captura excepciones específicas y redirige con mensajes de error apropiados

Tipos de excepciones manejadas:

* + LibroDuplicadoException
  + DatosLibroInvalidosException
  + LibroPrestadoException
  + Exception genérica

Flujo del Controlador

1. Obtiene datos iniciales llamando a obtenerDatosLibros()
2. Intenta procesar cualquier formulario enviado llamando a procesarFormularioLibros()
3. Si el formulario se procesó correctamente, actualiza los datos
4. Extrae las variables para la vista
5. Incluye el archivo de vista correspondiente (libros\_vista.php)

Dependencias

1. Modelos: libros\_modelo.php (para operaciones con la base de datos)
2. Seguridad: validar\_sesion.php (para autenticación)
3. Validación: LibroValidatorService.php (para validación de datos)
4. Excepciones personalizadas: excepciones\_libro.php

Mensajes de Error

El controlador puede generar los siguientes mensajes de error:

1. Faltan campos requeridos: Cuando no se completan todos los campos obligatorios
2. Libro duplicado: Cuando se intenta crear un libro que ya existe
3. Datos inválidos: Cuando los datos proporcionados no pasan la validación
4. Libro prestado: Cuando se intenta modificar/eliminar un libro que está prestado

Uso Típico

1. El usuario accede a la vista de libros
2. El sistema carga los datos iniciales mediante obtenerDatosLibros()
3. El usuario realiza una acción (crear, modificar, eliminar, etc.)
4. El sistema procesa la acción mediante procesarFormularioLibros()
5. Según el resultado:
   * Muestra mensaje de éxito y actualiza la vista
   * Muestra mensaje de error si falla la operación

*Notas Importantes*

* Todas las operaciones requieren sesión válida
* Los errores se redirigen a la vista con códigos y mensajes descriptivos
* El controlador sigue el patrón MVC (Modelo-Vista-Controlador)

### Controlador: existencias\_controlador.php

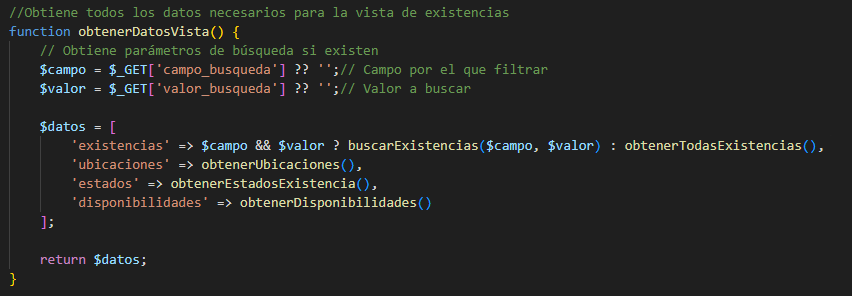
Este controlador gestiona todas las operaciones relacionadas con las existencias de libros en el sistema, incluyendo la creación, modificación y búsqueda de existencias. También proporciona los datos necesarios para poblar los selectores de la vista (ubicaciones, estados y disponibilidades).

***Funcionalidades Principales***

* Obtención de datos para la vista
* Creación de nuevas existencias
* Modificación de existencias existentes
* Búsqueda de existencias

*Funciones Principales*

obtenerDatosVista()



Propósito: Obtiene todos los datos necesarios para mostrar en la vista de existencias

* Parámetros de búsqueda opcionales (vía GET):
  + campo\_busqueda: Campo por el que filtrar
  + valor\_busqueda: Valor a buscar

*Retorna:*

1. Array con:
   * existencias: Lista de existencias (filtradas si hay búsqueda)
   * ubicaciones: Lista de ubicaciones disponibles
   * estados: Lista de estados posibles para existencias
   * disponibilidades: Lista de tipos de disponibilidad
2. Lógica:
   * Si hay parámetros de búsqueda, filtra las existencias
   * Si no hay búsqueda, devuelve todas las existencias

*Procesamiento de Formularios*

***Creación de Nueva Existencia***

Activador: Formulario POST con guardar\_existencia

1. Campos requeridos:
   * libroID: ID del libro
   * ubicacionID: ID de la ubicación
   * estadoExistenciaID: ID del estado
   * disponibilidadExistenciaID: ID de la disponibilidad

Acción: Llama a insertarExistencia() con los parámetros

1. Resultados:
   * Éxito: Redirecciona con parámetro exito=1
   * Error: Muestra mensaje de error

Modificación de Existencia



Activador: Formulario POST con accion=modificar

1. Campos requeridos:
   * existenciaID: ID de la existencia a modificar
   * libroID: Nuevo ID del libro
   * ubicacionID: Nueva ubicación
   * estadoExistenciaID: Nuevo estado
   * disponibilidadExistenciaID: Nueva disponibilidad

Acción: Llama a modificarExistencia() con los parámetros

1. Resultados:
   * Éxito: Redirecciona con parámetro modificado=1
   * Error: Muestra mensaje de error

***Dependencias***

Modelo: existencias\_modelo.php (contiene todas las funciones de acceso a datos)

***Mensajes y Redirecciones***

1. Creación exitosa: Redirección a existencias.php?exito=1
2. Modificación exitosa: Redirección a existencias.php?modificado=1
3. Errores: Muestra mensaje directo en pantalla (en producción se recomienda redireccionar con mensajes de error)

**Flujo del Controlador**

1. Carga inicial:
   * Obtiene datos para la vista mediante obtenerDatosVista()
   * Proporciona datos para selectores (ubicaciones, estados, disponibilidades)
2. Procesamiento de formularios:
   * Detecta si hay envío POST para creación o modificación
   * Valida y procesa los datos
   * Redirecciona con feedback apropiado

***Uso Típico***

Visualización inicial:

* + Usuario accede a la vista
  + Sistema carga todos los datos necesarios

Búsqueda:

* + Usuario introduce criterios de búsqueda (GET)
  + Sistema filtra las existencias mostradas

Creación/Modificación:

* + Usuario envía formulario
  + Sistema procesa los cambios
  + Redirecciona con confirmación o muestra error

***Notas Importantes***

1. No incluye manejo explícito de excepciones (se recomienda añadirlo)
2. Los mensajes de error se muestran directamente (en producción usar redirecciones)
3. La búsqueda se realiza mediante parámetros GET
4. Las acciones de creación/modificación usan POST

### Controlador: prestamos\_controlador.php

Este controlador gestiona las operaciones relacionadas con la búsqueda de préstamos en el sistema. Se encarga de validar los términos de búsqueda, consultar los préstamos según los criterios proporcionados y manejar los errores correspondientes.

***Funcionalidades Principales***

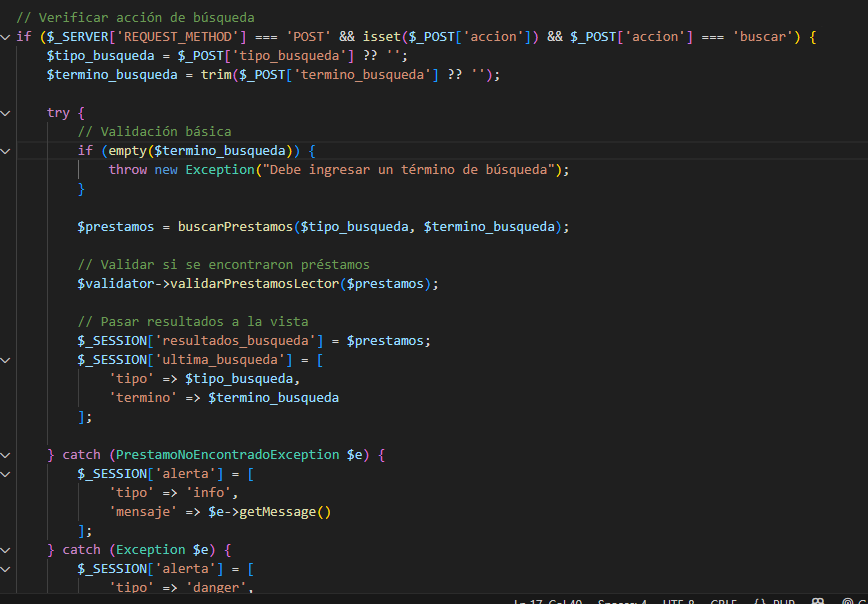
* Búsqueda de préstamos (mediante filtros).
* Validación de datos de búsqueda.
* Manejo de excepciones y alertas (éxito, error, información).
* Redirección a la vista con resultados o mensajes de error.

***Dependencias***

* Modelo: prestamos\_modelo.php (para operaciones con la base de datos).
* Seguridad: validar\_sesion.php (verifica que el usuario tenga sesión activa).
* Validación: PrestamoValidatorService.php (valida los préstamos encontrados).

***Flujo del Controlador***

1. Búsqueda de Préstamos



*Activador:*

* Método POST con accion=buscar.
* Campos requeridos:
  + tipo\_busqueda (criterio de búsqueda: "lector", "libro", "fecha", etc.).
  + termino\_busqueda (valor a buscar).

*Validaciones:*

* Verifica que el término de búsqueda no esté vacío.
* Llama a buscarPrestamos() con los parámetros proporcionados.
* Valida los resultados con PrestamoValidatorService.

*Resultados:*

* Éxito:
  + Almacena los resultados en $\_SESSION['resultados\_busqueda'].
  + Guarda el historial de búsqueda en $\_SESSION['ultima\_busqueda'].
* Error:
  + Si no se encuentran préstamos (PrestamoNoEncontradoException), muestra alerta de tipo info.
  + Si hay otro error (Exception), muestra alerta de tipo danger.
* Redirección:
  + Siempre redirige a prestamos\_vista.php después de procesar la búsqueda.

***Notas Importantes***

1. Sesión requerida: El usuario debe estar autenticado (validar\_sesion.php).
2. Solo maneja búsquedas: No gestiona creación, modificación o eliminación de préstamos.
3. Mensajes en sesión: Los errores y resultados se pasan mediante $\_SESSION.

### Controlador: lector\_controlador.php

Este controlador gestiona todas las operaciones relacionadas con los lectores en el sistema, incluyendo:

* Registro de nuevos lectores
* Modificación de datos de lectores existentes
* Búsqueda de lectores
* Activación/Desactivación de cuentas

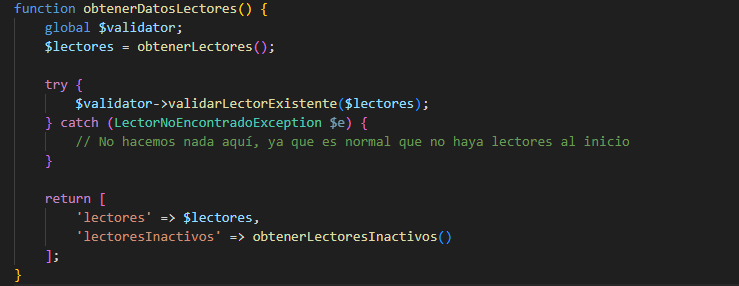
Utiliza un servicio de validación (LectorValidatorService) para asegurar la integridad de los datos y evitar duplicados.

*Dependencias*

* Modelo: lector\_modelo.php (operaciones con la base de datos).
* Seguridad: validar\_sesion.php (autenticación del usuario).
* Validación: lectorValidatorService.php (valida datos y evita duplicados).

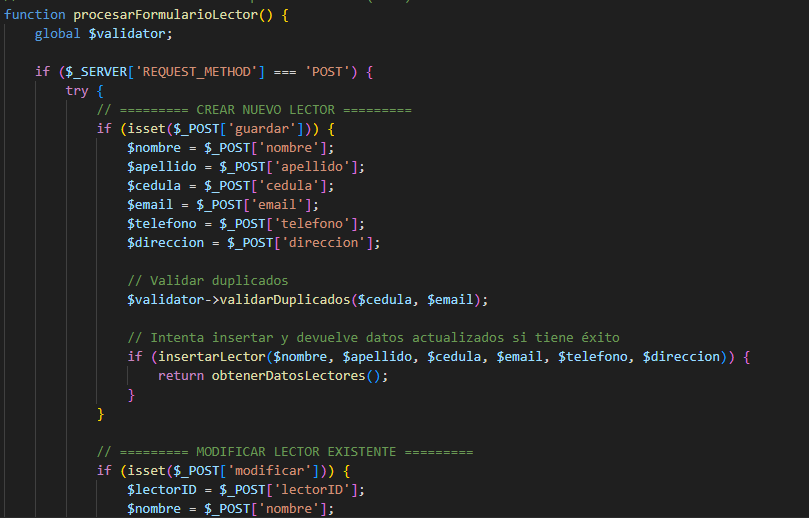
***Funciones Principales***

1. obtenerDatosLectores()



* Propósito: Obtiene los datos iniciales para la vista.
* Retorna:
  + lectores: Lista de lectores activos.
  + lectoresInactivos: Lista de lectores inactivos.
* Manejo de errores:
* Si no hay lectores, no genera error (solo retorna arrays vacíos).

2. procesarFormularioLector()



Procesa las acciones enviadas por POST y maneja:

***A. Creación de un nuevo lector (guardar)***

* *Campos requeridos:*
* nombre, apellido, cedula, email, telefono, direccion
* *Validaciones:*
* Verifica duplicados en cedula y email (LectorValidatorService).
* *Acción:*
* Llama a insertarLector() y actualiza los datos si es exitoso.

***B. Modificación de lector existente (modificar)***

* *Campos requeridos:*
  + lectorID, nombre, apellido, cedula, email, telefono, direccion
* *Validaciones:*
  + Verifica duplicados excluyendo el lector actual.
* *Acción:*
  + Llama a modificarLector() y actualiza los datos si es exitoso.

***C. Búsqueda de lectores (buscar)***

* *Campos requeridos:*
  + campo\_busqueda (ej: "nombre", "cedula"), valor\_busqueda
* *Validaciones:*
  + Verifica que existan resultados (validarLectorExistente()).
* *Retorna:*
  + Lista filtrada de lectores.

***D. Desactivar lector (desactivarID)***

* *Acción:*
  + Llama a desactivarLector() y actualiza los datos.
* *Nota:*
  + Falta implementar validación de préstamos pendientes antes de desactivar.

***E. Reactivar lector (reactivarID)***

* *Acción:*
  + Llama a reactivarLector() y actualiza los datos.

**Manejo de Errores**

* *Errores esperados:*
  + LectorDuplicadoException (cédula/email ya registrados).
  + LectorNoEncontradoException (búsqueda sin resultados).
  + LectorBloqueadoException (intento de modificar un lector inactivo).
* *Almacenamiento de errores:*
  + Los mensajes de error se guardan en $\_SESSION['error\_message'].

**Flujo del Controlador**

1. Carga inicial:
   * Obtiene datos con obtenerDatosLectores().
2. Procesamiento de formulario:
   * Si hay una acción POST, ejecuta procesarFormularioLector().
3. Actualización de datos:
   * Si el formulario se procesó correctamente, actualiza $datos.
4. Renderizado de la vista:
   * Extrae variables y carga lector\_vista.php.

### Controlador: bibliotecario\_controlador.php

Este controlador gestiona todas las operaciones relacionadas con los bibliotecarios en el sistema, incluyendo:

* Registro de nuevos bibliotecarios
* Modificación de datos existentes
* Eliminación de cuentas
* Búsqueda de bibliotecarios

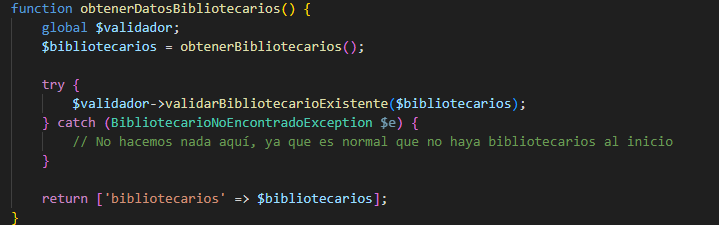
Utiliza un servicio de validación (BibliotecarioValidatorService) para asegurar la integridad de los datos y evitar duplicados.

*Dependencias*

* Modelo: bibliotecario\_modelo.php (operaciones con la base de datos).
* Seguridad: validar\_sesion.php (autenticación del usuario).
* Validación: BibliotecarioValidatorService.php (valida datos y evita duplicados).

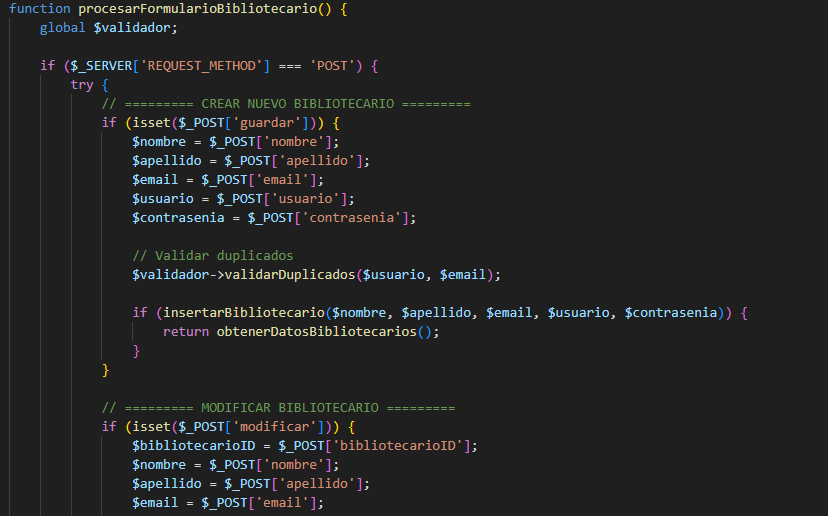
***Funciones Principales***

1. obtenerDatosBibliotecarios()



* *Propósito:* Obtiene los datos iniciales para la vista.
* *Retorna:*
  + bibliotecarios: Lista de bibliotecarios activos.
* *Manejo de errores:*
  + Si no hay bibliotecarios, no genera error (solo retorna un array vacío).

1. procesarFormularioBibliotecario()



Procesa las acciones enviadas por POST y maneja:

***A. Creación de un nuevo bibliotecario (guardar)***

* *Campos requeridos:*
  + nombre, apellido, email, usuario, contrasenia
* Validaciones:
  + Verifica duplicados en usuario y email (BibliotecarioValidatorService).
* Acción:
  + Llama a insertarBibliotecario() y actualiza los datos si es exitoso.

***B. Modificación de bibliotecario existente (modificar)***

* Campos requeridos:
  + bibliotecarioID, nombre, apellido, email, usuario, contrasenia (opcional)
* *Validaciones:*
  + Verifica duplicados excluyendo el bibliotecario actual.
* *Acción:*
  + Llama a modificarBibliotecario() y actualiza los datos si es exitoso.

***C. Eliminación de bibliotecario (eliminarID)***

* *Acción:*
  + Llama a eliminarBibliotecario() y actualiza los datos.

***D. Búsqueda de bibliotecarios (buscar)***

* *Campos requeridos:*
  + campo\_busqueda (ej: "nombre", "email"), valor\_busqueda
* *Validaciones:*
  + Verifica que existan resultados (validarBibliotecarioExistente()).
  + Si hay un solo resultado, valida su estado (validarEstadoBibliotecario()).
* *Retorna:*
  + Lista filtrada de bibliotecarios.

***Manejo de Errores***

* *Errores esperados:*

1. BibliotecarioDuplicadoException (usuario/email ya registrados).
2. BibliotecarioNoEncontradoException (búsqueda sin resultados).
3. BibliotecarioBloqueadoException (intento de modificar un bibliotecario bloqueado).

* Almacenamiento de errores:

1. Los mensajes de error se guardan en $\_SESSION['error\_message'].

***Flujo del Controlador***

* *Carga inicial:*
  + Obtiene datos con obtenerDatosBibliotecarios().
* *Procesamiento de formulario:*
  + Si hay una acción POST, ejecuta procesarFormularioBibliotecario().
* *Actualización de datos:*
  + Si el formulario se procesó correctamente, actualiza $datos.
* *Renderizado de la vista:*
  + Extrae variables y carga bibliotecario\_vista.php.

### Controlador: cubiculo\_controlador.php

Este controlador gestiona todas las operaciones relacionadas con la administración de cubículos en el sistema, incluyendo:

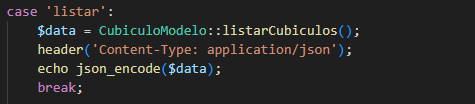
* Listado de cubículos disponibles
* Registro de nuevos cubículos
* Proceso de alquiler/devolución
* Validación de lectores para alquiler

Utiliza un servicio de validación (CubiculoValidatorService) para garantizar la integridad de las operaciones.

*Dependencias*

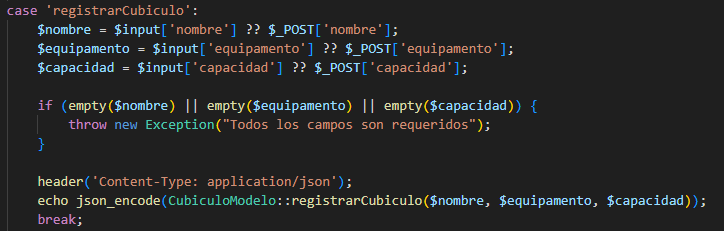
* Modelo: cubiculo\_modelo.php (operaciones con la base de datos)
* Validación: CubiculoValidatorService.php (valida disponibilidad y estados)
* Formato de datos: Manejo de JSON para API RESTful

1. ***Listar Cubículos (listar)***



* *Método:* GET/POST
* *Propósito:* Obtiene el listado completo de cubículos registrados

1. *Registrar Cubículo (registrarCubiculo)*



* *Método:* POST
* Parámetros requeridos:
  + nombre (string)
  + equipamento (string)
  + capacidad (integer)

1. ***Buscar Lector (buscarLector)***

* *Método:* GET
* *Parámetro requerido:*
* cedula (string)
* Validaciones:
* Existencia del lector
* Estado habilitado del lector

***Alquilar Cubículo (alquilarCubiculo)***

* *Método:* POST
* *Parámetros requeridos:*
* lectorID (integer)
* cubiculoID (integer)
* *Validaciones:*
* Disponibilidad del cubículo
* Integridad de datos

***Devolver Cubículo (devolverCubiculo)***

* *Método:* POST
* *Parámetro requerido:*
* cubiculoID (integer)

**Excepciones manejadas:**

* LectorNoEncontradoException (código 404)
* LectorBloqueadoException (código 403)
* CubiculoNoDisponibleException (código 409)
* Exception genérica (código 500)

**Flujo del Controlador**

1. Inicialización:

* Detecta el formato de entrada (JSON/POST/GET)
* Identifica la acción solicitada

1. Ejecución:

* Valida parámetros requeridos
* Ejecuta la operación correspondiente
* Aplica validaciones adicionales

1. Respuesta:

* Devuelve datos en formato JSON
* Incluye cabeceras HTTP apropiadas

### Controlador: control\_controlador.php

Este controlador sirve para gestionar las operaciones relacionadas con multas y bloqueo/desbloqueo de lectores en el sistema.

***Estructura General***

1. Inicialización

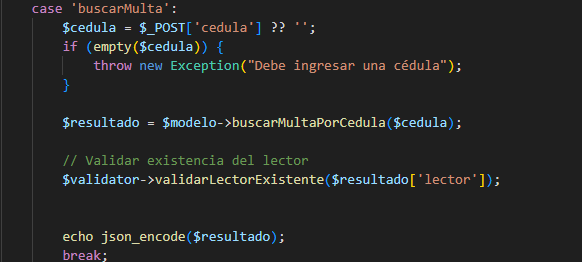
* Inicia sesiones con session\_start()
* Establece el tipo de contenido como JSON
* Incluye los archivos del modelo y servicio de validación

1. Flujo Principal

* Crea instancias del validador y modelo
* Obtiene la acción a realizar desde POST
* Ejecuta un switch-case para manejar las diferentes acciones

***Acciones Disponibles***

*1. Buscar Multa (buscarMulta)*



Propósito: Buscar multas asociadas a una cédula

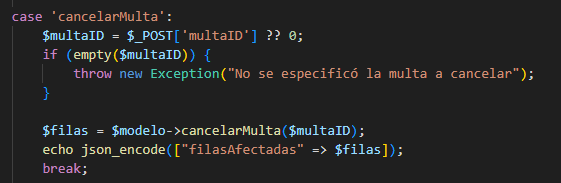
Validaciones:

* Verifica que se haya proporcionado una cédula
* Valida la existencia del lector usando el servicio ControlValidatorService

Respuesta:

Retorna un JSON con el resultado de la búsqueda

*2. Cancelar Multa (cancelarMulta)*



Propósito: Marcar una multa como cancelada

Validaciones:

* Verifica que se haya proporcionado un ID de multa

Respuesta:

Retorna un JSON con el número de filas afectadas

*3. Bloquear Lector (bloquearLector)*

Propósito: Bloquear la cuenta de un lector

*Validaciones:*

* Verifica que se haya proporcionado un ID de lector

*Respuesta:*

* Retorna un mensaje de confirmación

*4. Desbloquear Lector (desbloquearLector)*

Propósito: Desbloquear la cuenta de un lector

Validaciones:

* Verifica que se haya proporcionado un ID de lector

Respuesta:

* Retorna un mensaje de confirmación

***Manejo de Errores***

El controlador captura y maneja tres tipos de excepciones:

1. LectorNoEncontradoException: Cuando no se encuentra un lector
2. MultasNoEncontradasException: Cuando no se encuentran multas
3. Exception: Para errores genéricos

Cada error se devuelve como un objeto JSON con el mensaje correspondiente.

## ¿Qué funciones hay en cada modelo?

El modelo es el encargado de enviar los datos a la base de datos y de la conexión a esta misma. Aquí tenemos:

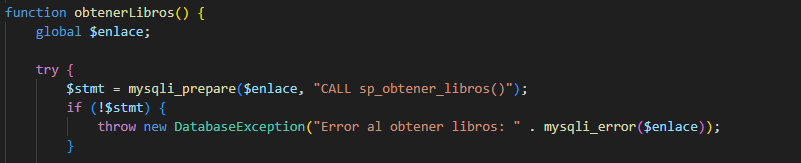
### Modelo: libros\_modelo.php

***Funciones Principales***

insertarLibro($title, $author, $year, $pages\_no, $genderID)

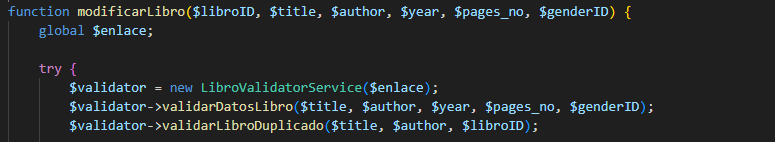
* Propósito: Inserta un nuevo libro en la base de datos
* Validaciones:
  + Valida los datos del libro (título, autor, año, páginas, género)
  + Verifica que no exista un libro duplicado (mismo título y autor)
* Proceso: Ejecuta el procedimiento almacenado sp\_insertar\_libro
* Excepciones: Lanza DatabaseException en caso de error

***obtenerLibros()***



* Propósito: Obtiene todos los libros registrados en el sistema
* Proceso: Ejecuta sp\_obtener\_libros y devuelve un array con los resultados
* Excepciones:
* DatabaseException para errores de conexión
* LibroNoEncontradoException si no hay libros registrados

***modificarLibro($libroID, $title, $author, $year, $pages\_no, $genderID)***



* Propósito: Actualiza la información de un libro existente
* Validaciones: Igual que insertarLibro pero excluyendo el libro actual
* Proceso: Ejecuta sp\_modificar\_libro con los nuevos datos

**Funciones de Estado**

*obtenerLibrosHabilitados()*

* Propósito: Devuelve solo los libros con estado activo
* Proceso: Ejecuta sp\_obtener\_libros\_habilitados

*obtenerLibrosDeshabilitados()*

* Propósito: Devuelve solo los libros desactivados
* Proceso: Ejecuta sp\_obtener\_libros\_deshabilitados

*cambiarEstadoLibro($libroID, $estado)*

* Propósito: Activa/desactiva un libro (1=activo, 0=inactivo)
* Proceso: Ejecuta sp\_habilitar\_libro

**Búsqueda**

*buscarLibros($campo\_busqueda, $valor\_busqueda)*

* Propósito: Busca libros por un campo específico (ej: título, autor)
* Proceso: Ejecuta sp\_buscar\_libros con los parámetros de búsqueda

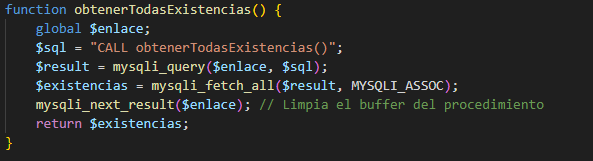
**Características Comunes**

* Todas las funciones utilizan procedimientos almacenados
* Manejo de excepciones para errores de base de datos
* Conexión a BD mediante variable global $enlace
* Validación centralizada en LibroValidatorService

### Modelo: existencias\_modelo.php

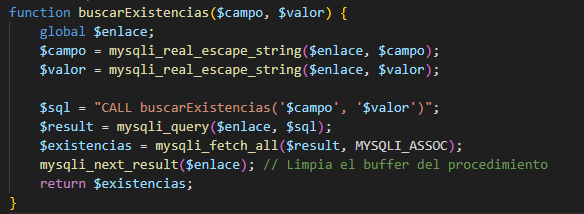
**Funciones Principales**

*obtenerTodasExistencias()*



* Propósito: Obtiene todos los registros de existencias con información relacionada
* Proceso: Ejecuta obtenerTodasExistencias() y devuelve array asociativo
* Nota: Limpia el buffer de resultados con mysqli\_next\_result()

*buscarExistencias($campo, $valor)*



* Propósito: Busca existencias por campo específico (ID o título)
* Seguridad: Aplica mysqli\_real\_escape\_string a los parámetros
* Proceso: Ejecuta buscarExistencias() con los parámetros filtrados

**Funciones de Catálogo**

*obtenerUbicaciones()*

* Propósito: Obtiene lista de ubicaciones disponibles (para combos/selects)
* Proceso: Ejecuta obtenerUbicaciones()

*obtenerEstadosExistencia()*

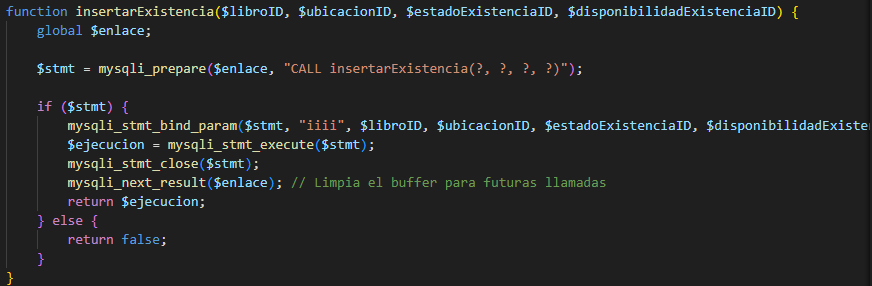
* Propósito: Obtiene posibles estados de existencia (ej: "Disponible", "Prestado")
* Proceso: Ejecuta obtenerEstadosExistencia()

*obtenerDisponibilidades()*

* Propósito: Obtiene tipos de disponibilidad (ej: "Física", "Digital")
* Proceso: Ejecuta obtenerDisponibilidades()

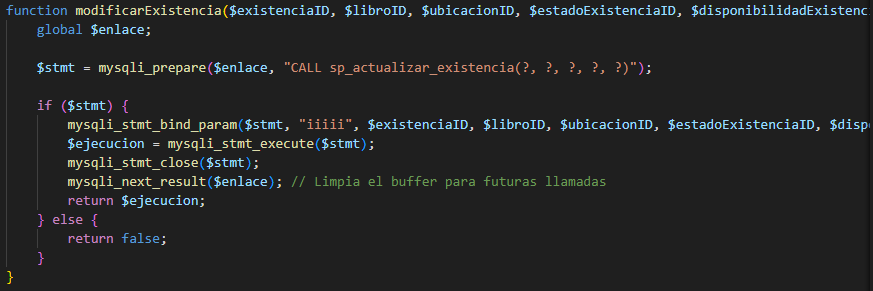
**Operaciones CRUD**

*insertarExistencia($libroID, $ubicacionID, $estadoExistenciaID, $disponibilidadExistenciaID)*



* Propósito: Crea un nuevo registro de existencia
* Proceso: Ejecuta insertarExistencia() con parámetros preparados
* Retorno: Booleano indicando éxito/fracaso

*modificarExistencia($existenciaID, $libroID, $ubicacionID, $estadoExistenciaID, $disponibilidadExistenciaID)*

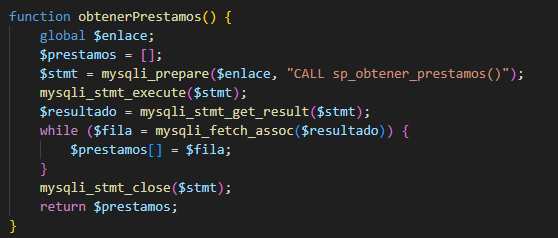


* Propósito: Actualiza un registro de existencia existente
* Proceso: Ejecuta sp\_actualizar\_existencia con parámetros preparados

### Modelo: préstamo\_modelo.php

**Funciones Principales**

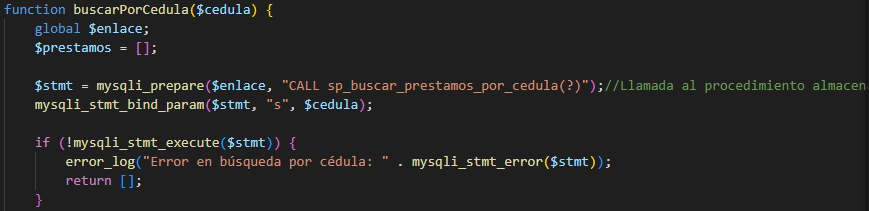
*obtenerPrestamos()*



* Propósito: Obtiene todos los préstamos registrados
* Proceso: Ejecuta sp\_obtener\_prestamos()

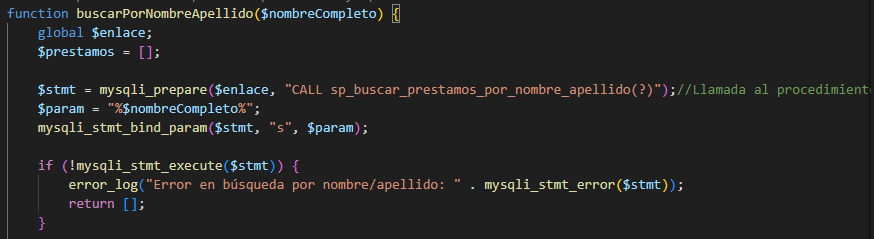
**Funciones de Búsqueda**

*buscarPorCedula($cedula)*



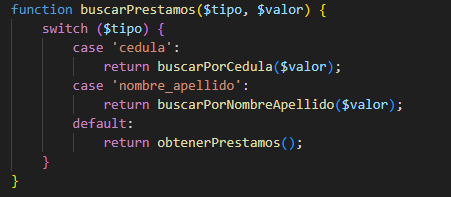
* Propósito: Busca préstamos por número de cédula
* Proceso: Ejecuta sp\_buscar\_prestamos\_por\_cedula()
* Log: Registra errores en el log

*buscarPorNombreApellido($nombreCompleto)*



* Propósito: Busca préstamos por coincidencia parcial de nombre/apellido
* Proceso: Ejecuta sp\_buscar\_prestamos\_por\_nombre\_apellido() con búsqueda LIKE

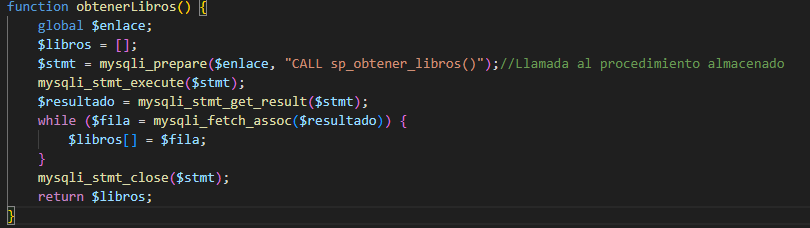
*buscarPrestamos($tipo, $valor)*



* Propósito: Función unificada para búsqueda (redirige según tipo)
* Tipos soportados: 'cedula' o 'nombre\_apellido'

**Función Auxiliar**

*obtenerLibros()*

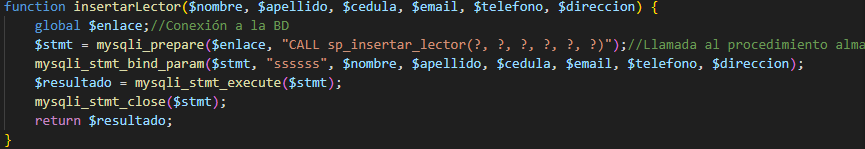


* Propósito: Obtiene lista completa de libros (probablemente para combos/selects)
* Proceso: Ejecuta sp\_obtener\_libros()

### Modelo: lector\_modelo.php

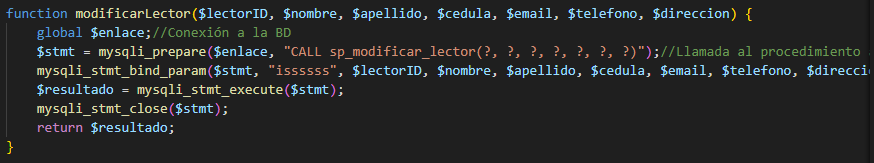
**Funciones CRUD**

*insertarLector($nombre, $apellido, $cedula, $email, $telefono, $direccion)*



* Propósito: Registra un nuevo lector en el sistema
* Proceso: Ejecuta sp\_insertar\_lector con datos personales
* Retorno: Booleano (éxito/fracaso)

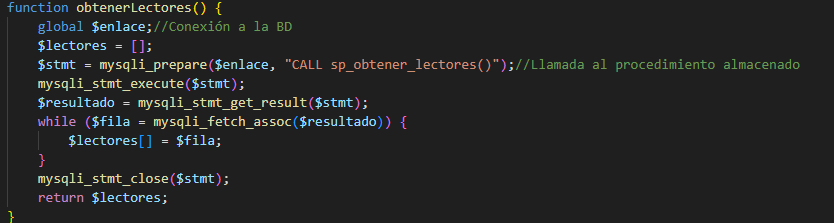
*modificarLector($lectorID, $nombre, $apellido, $cedula, $email, $telefono, $direccion)*



* Propósito: Actualiza datos de un lector existente
* Proceso: Ejecuta sp\_modificar\_lector con ID y nuevos datos

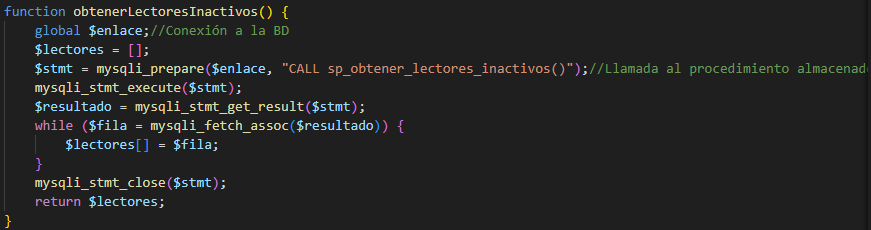
**Funciones de Consulta**

*obtenerLectores()*



* Propósito: Lista todos los lectores activos
* Proceso: Ejecuta sp\_obtener\_lectores

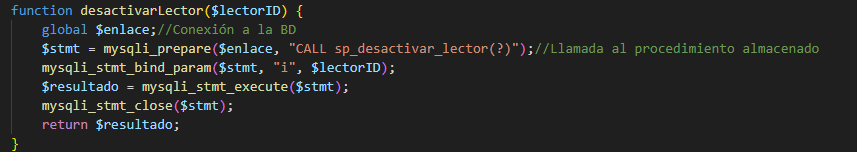
*obtenerLectoresInactivos()*



* Propósito: Lista lectores desactivados
* Proceso: Ejecuta sp\_obtener\_lectores\_inactivos

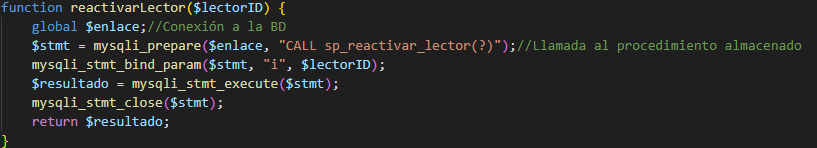
**Gestión de Estado**

*desactivarLector($lectorID)*



* Propósito: Desactiva un lector (borrado lógico)
* Proceso: Ejecuta sp\_desactivar\_lector

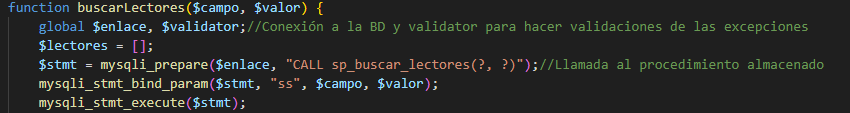
*reactivarLector($lectorID)*



* Propósito: Reactiva un lector previamente desactivado
* Proceso: Ejecuta sp\_reactivar\_lector

**Búsqueda**

*buscarLectores($campo, $valor)*

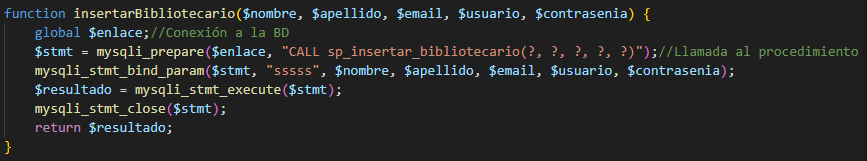


* Propósito: Búsqueda flexible por diferentes campos
* Validación: Verifica estado del lector si hay un único resultado
* Proceso: Ejecuta sp\_buscar\_lectores

### Modelo: bibliotecario\_modelo.php

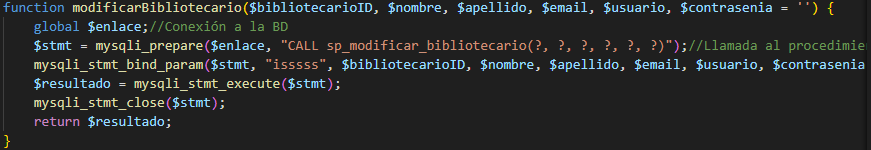
**Funciones CRUD**

*insertarBibliotecario($nombre, $apellido, $email, $usuario, $contrasenia)*



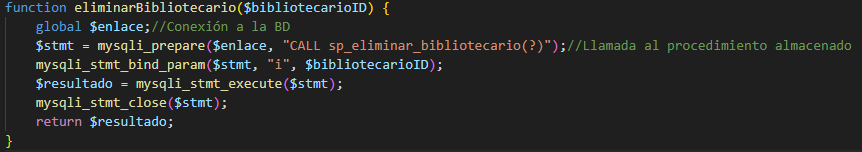
* Propósito: Registra nuevo personal bibliotecario
* Seguridad: Contraseña almacenada con hash (debe implementarse)
* Proceso: Ejecuta sp\_insertar\_bibliotecario

*modificarBibliotecario($bibliotecarioID, $nombre, $apellido, $email, $usuario, $contrasenia = '')*



* Propósito: Actualiza datos del bibliotecario
* Característica: Contraseña opcional para actualizaciones parciales
* Proceso: Ejecuta sp\_modificar\_bibliotecario

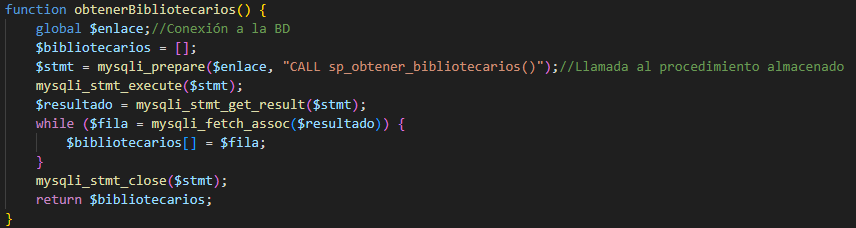
*eliminarBibliotecario($bibliotecarioID)*



* Propósito: Eliminación permanente de bibliotecarios
* Proceso: Ejecuta sp\_eliminar\_bibliotecario

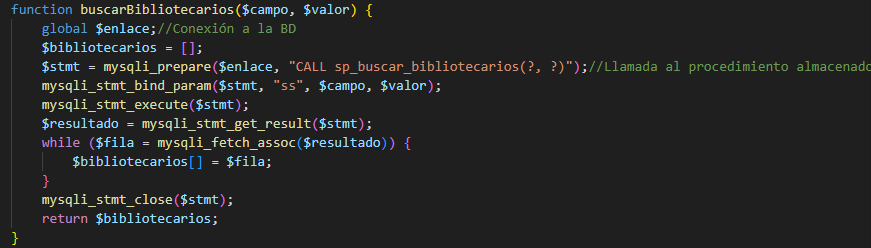
**Funciones de Consulta**

*obtenerBibliotecarios()*



* Propósito: Lista todo el personal bibliotecario
* Proceso: Ejecuta sp\_obtener\_bibliotecarios

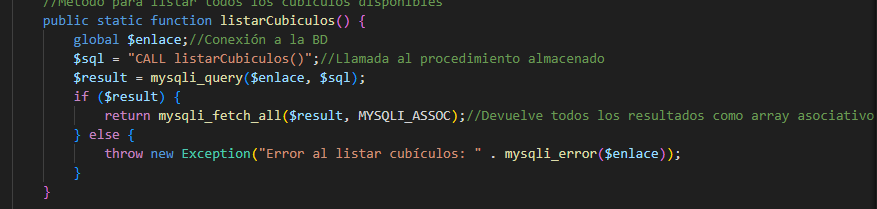
*buscarBibliotecarios($campo, $valor)*



* *Propósito:* Búsqueda por diferentes criterios
* *Proceso:* Ejecuta sp\_buscar\_bibliotecarios

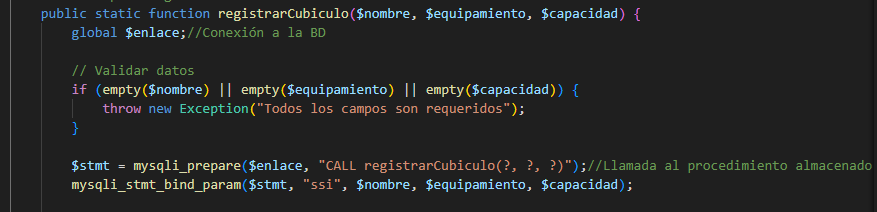
### Modelo: cubículo\_modelo.php

*listarCubiculos()*



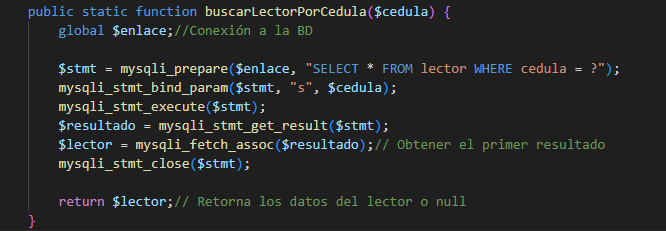
* Propósito: Obtiene todos los cubículos registrados
* Proceso: Ejecuta listarCubiculos()
* Retorno: Array asociativo con datos de cubículos
* Excepción: Lanza error si falla la consulta

*registrarCubiculo($nombre, $equipamiento, $capacidad)*



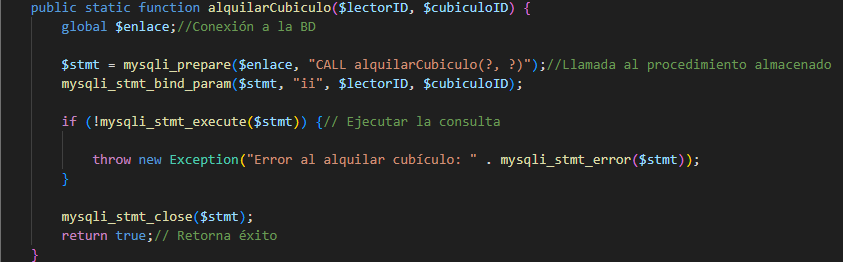
* Propósito: Crea un nuevo cubículo
* Validaciones:
  + Campos obligatorios (nombre, equipamiento, capacidad)
* Proceso: Ejecuta registrarCubiculo con parámetros preparados
* Retorno: Array con indicador de éxito

*buscarLectorPorCedula($cedula)*



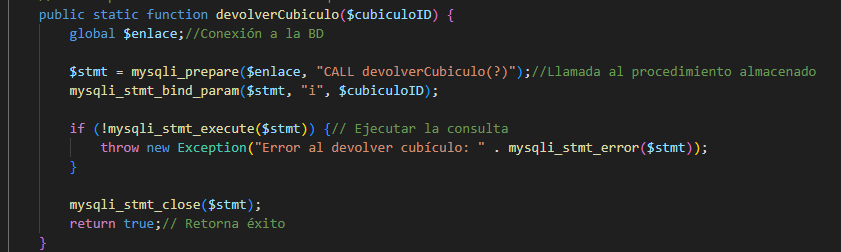
* Propósito: Busca lectores para asignar a cubículos
* Proceso: Consulta directa a tabla lector
* Retorno: Datos del lector o null

*alquilarCubiculo($lectorID, $cubiculoID)*



* Propósito: Asigna cubículo a lector
* Proceso: Ejecuta alquilarCubiculo
* Excepción: Lanza error si falla el alquiler

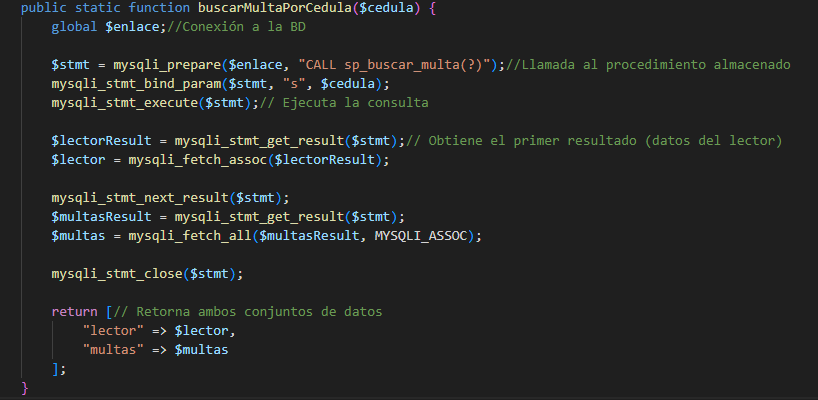
*devolverCubiculo($cubiculoID)*



* Propósito: Libera un cubículo ocupado
* Proceso: Ejecuta devolverCubiculo
* Excepción: Lanza error si falla la devolución

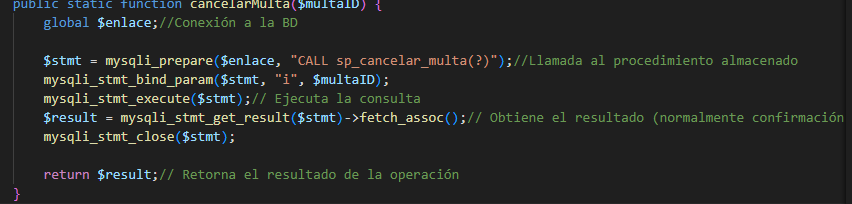
### Modelo: control\_modelo.php

*buscarMultaPorCedula($cedula)*



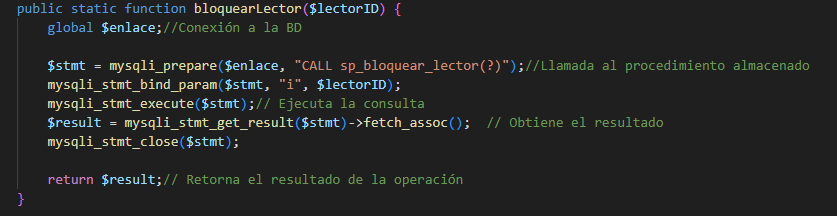
* Propósito: Consulta multas por cédula de lector
* Proceso:
  + Ejecuta sp\_buscar\_multa
  + Maneja múltiples resultados (datos lector + multas)
* Retorno: Array con:
  + lector: Datos del lector
  + multas: Listado de multas asociadas

*cancelarMulta($multaID)*



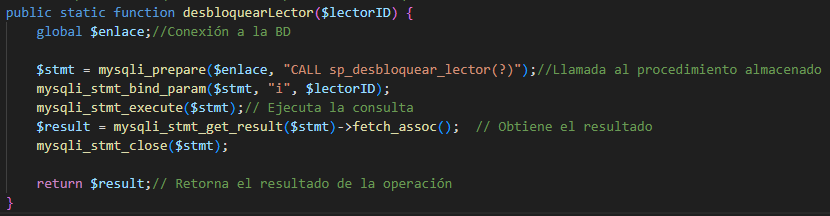
* Propósito: Marca multa como cancelada
* Proceso: Ejecuta sp\_cancelar\_multa
* Retorno: Resultado de la operación

*bloquearLector($lectorID)*



* Propósito: Restringe acceso del lector
* Proceso: Ejecuta sp\_bloquear\_lector
* Retorno: Confirmación de bloqueo

desbloquearLector($lectorID)



* Propósito: Reactiva acceso de lector bloqueado
* Proceso: Ejecuta sp\_desbloquear\_lector
* Retorno: Confirmación de desbloqueo

# Errores Comunes y Soluciones

***No funciona la conexión a la Base de Datos***

Solución:

1. Credenciales incorrectas

* *Verificar:*
  + Usuario (root)
  + Contraseña (vacía en este caso)
  + Nombre de la base de datos (gestionbiblioteca)
* Acción: Confirmar que coincidan con la configuración del servidor MySQL

1. Servidor MySQL no disponible

* *Verificar:*

sudo service mysql status

1. Base de datos no existe

* *Verificar:*

SHOW DATABASES;

* Solución: Crear la base de datos si no existe

1. Problemas de permisos

* *Verificar:*

SELECT User, Host FROM mysql.user;

* Solución:

GRANT ALL PRIVILEGES ON gestionbiblioteca.\* TO 'root'@'localhost';

FLUSH PRIVILEGES;

1. Solución Alternativa:

Probar conexión alternativa (PDO):

try {

$pdo = new PDO('mysql:host=localhost;dbname=gestionbiblioteca', 'root', '');

echo "Conexión PDO exitosa";

} catch (PDOException $e) {

echo "Error PDO: " . $e->getMessage();

}

***Formulario no Guarda Cambios.***

Solución:

1. Revisar la conexión con el punto anterior.
2. En caso de que algún formulario no guarde cambio se debe verificar que el procedimiento almacenado a ese formulario exista.

1.- Ir a Modelos y seleccionar el modelo que presenta error.

2.- Buscar la parte que diga: “CALL SP\_INSERTAR\_LECTOR”.

3.- Con el nombre dirigirse a la BD y buscar en la lista de procedimientos y verificar que exista en caso de que no exista update a la BD con el archivo “gestionbiblioteca.sql”.

1. En caso de que el procedimiento almacenado exista revisar los parámetros del procedimiento almacenado que espera y los parámetros que son mandados desde php con el controlador.

# Anexos

Para conocer el funcionamiento del sistema visualizar el Manual para Abrir el proyecto en el siguiente link:

<https://drive.google.com/file/d/1T8luU-mSxsYdkC1aYwHNnROx2z7AL60t/view?usp=sharing>

Para más información acerca de los requerimientos, casos de uso y demás diagramas visualizar el Documento Detallado de Software en el siguiente link:

<https://drive.google.com/file/d/1OhK1KLgPS1ULwvI7uD_mw50bw0q-BJY1/view?usp=sharing>