Sistema de Gestión de Libros Electrónicos

Objetivo General

Diseñar y desarrollar un sistema web funcional y modular en Go que permita gestionar libros electrónicos mediante la incorporación de técnicas de programación funcional, estructuras de datos organizadas y un enfoque escalable, reutilizable y mantenible.

Objetivos Específicos

Permitir el registro de nuevos libros electrónicos

El sistema debe brindar una interfaz para que el usuario ingrese información como título, autor, categoría, año y un identificador único.

Visualizar y listar todos los libros registrados

Se debe renderizar una vista dinámica que muestre todos los libros almacenados, de forma ordenada y clara.

Implementar búsqueda y filtrado funcional

Los usuarios deben poder buscar libros por criterios como autor o título, utilizando funciones puras para procesar los datos.

Actualizar la información de libros existentes

El sistema debe permitir modificar los datos de un libro de forma controlada y validada.

Eliminar libros del sistema de forma segura

Se debe ofrecer una opción para eliminar registros, con verificación para evitar eliminaciones accidentales.

Separar claramente la lógica funcional de la lógica de entrada/salida

Todas las funciones de procesamiento (filtrar, validar, calcular) deben estar desacopladas de la capa que maneja las peticiones HTTP o la base de datos.

Utilizar programación funcional como base de desarrollo lógico

El sistema debe estructurarse para aprovechar funciones puras, evitar el uso de estado global y fomentar código inmutable y reutilizable.

Incorporar paquetes estándar y externos documentados

Se deben utilizar paquetes como html/template, net/http, y otros externos como gorilla/mux, documentando su propósito y aplicación en el sistema.

Construir una interfaz web básica con vistas HTML renderizadas dinámicamente

Se deben generar páginas dinámicas que muestren los datos del sistema utilizando el motor de plantillas de Go.

Módulos a desarrollar

Módulo	Descripción
bd/	Contendrá la lógica para conectarse con la base de datos (ej. SQLite) y ejecutar operaciones CRUD.
controllers/	Manejadores HTTP (por ejemplo, CrearLibro, ListarLibros) que invocan funciones lógicas y devuelven respuestas al navegador.
funciones/	Conjunto de funciones reutilizables como validadores, filtros o transformaciones (no dependen del estado del programa).
models/	Definiciones de estructuras (struct) como Libro, que representarán los datos que manipula el sistema.
static/	Archivos estáticos para la interfaz de usuario (CSS, imágenes, JS).
vistas/	Plantillas HTML renderizadas con html/template para mostrar los datos en el navegador.
inicio.go	Punto de entrada del sistema (servidor web, rutas, inicialización de módulos).

Estructura de Gestión del Sistema

```
proyecto/
│
├── bd/
│   └── conexion.go            → conexión y funciones DB (crearTabla, insertarLibro, etc.)
│
├── controllers/
│   └── libroController.go     → controladores de rutas (manejan entrada y salida)
│
├── funciones/
│   └── operaciones.go         → funciones puras como filtrarPorAutor, validarEntrada
│
├── models/
│   └── libro.go               → definición del tipo `Libro`
│
├── static/
│   ├── css/estilos.css        → estilos de interfaz
│   ├── img/                   → portadas y recursos gráficos
│   └── js/app.js              → funcionalidades básicas de UI
│
├── vistas/
│   └── index.html             → interfaz principal
│
└── inicio.go                  → entry point, servidor y rutas
```

Enfoque de arquitectura

Se adopta una arquitectura modular **MVC simplificada**, distribuida de la siguiente manera:

Modelo (models/): define los tipos de datos (por ejemplo, Libro, Usuario si se amplía).

Controlador (controllers/): conecta el modelo con las vistas, ejecuta acciones según las rutas.

Vista (vistas/): renderiza HTML dinámicamente con los datos recibidos del controlador.

Flujo típico de una operación (ej. listar libros)

El usuario accede a localhost:8080/libros.

El router (mux) invoca ListarLibros en el paquete controllers.

El controlador llama a bd.ObtenerLibros() para obtener la lista desde la base de datos.

La lista se pasa a la plantilla vistas/libros.html para su renderizado, es decir el proceso de generar y mostrar contenido visual (HTML) a partir de datos.

Paquetes Incorporados

Paquetes estándar de Go:

net/http:	servidor y manejadores de rutas HTTP.
html/template:	motor de plantillas HTML.
fmt, log, os:	utilidades para impresión, logging y manejo de archivos.
database/sql:	interfaz para conectarse con la base de datos.
encoding/json:	para responder en formato JSON (si se hace una API).

Paquetes de terceros (documentados)

Paquete	**Propósito**
github.com/gorilla/mux Router	Enrutador HTTP, fácil de usar y escalable
github.com/mattn/go-sqlite3	Driver para usar SQLite en Go
github.com/joho/godotenv	Permite cargar variables de entorno desde archivos .env.
github.com/go-playground/validator/v10	Validación de structs de entrada (funcional sin estado)

Documentación de terceros

Cada paquete debe ir acompañado de comentarios y/o README en el proyecto indicando:

Dónde se usa

Por qué se eligió

Qué licencia tiene (si aplica)

Programación Funcional

La programación funcional se basa en construir software usando funciones puras, es decir, funciones que no dependen ni modifican el estado externo, sino que reciben datos y devuelven resultados sin efectos secundarios. En Go se aplica evitando variables globales, usando funciones reutilizables y separando la lógica del sistema de la entrada/salida. Esto hace el código más modular, legible y fácil de mantener.