Manual Técnico SoftLibray

Integrantes: Melissa C Palma Sosa,
Edson J Mijangos Cú, Cristopher Marquez Valenzuela,Saleth Huehuet.
SoftLibrary inc.
al068939@uacam.mx

MANUAL TÉCNICO DEL PROYECTO SOFTLIBRARY

Introducción

SoftLibrary es una aplicación desarrollada en Laravel diseñada para gestionar bibliotecas, incluyendo la administración de libros, materiales didácticos, aulas y sus respectivos procesos de renta y devolución.
Este manual técnico tiene como objetivo proporcionar información detallada sobre la arquitectura del sistema, las tecnologías empleadas, la estructura del proyecto, y la configuración necesaria para el despliegue y mantenimiento.

Requisitos del Sistema

- Servidor Web: Apache
- PHP: Versión 8.1 o superior
- Base de Datos: MySQL 5.7+ o MariaDB
- Composer: Herramienta de gestión de dependencias de PHP
- Laravel: Para manejar assets front-end y compilaciones

Configuración Inicial

1. Clonar el repositorio del proyecto desde el control de versiones.
2. Configurar el archivo `.env` con las credenciales de la base de datos y otros detalles.
3. Ejecutar `composer install` para instalar las dependencias PHP.
4. Ejecutar `npm install && npm run dev` para compilar los assets front-end.
5. Ejecutar `php artisan migrate --seed` para crear y poblar las tablas de la base de datos.

Estructura del Proyecto

1. App. La carpeta app contiene el núcleo de la lógica del negocio y es el lugar donde Laravel organiza los componentes principales que interactúan con la aplicación.
    a. Controladores (app/Http/Controllers):
        i. Contienen la lógica que procesa las solicitudes HTTP.
        ii. Ejemplo en el sistema: LibroController maneja acciones como agregar libros, mostrar la lista, etc.
        iii. Posible problema: Sobrecarga de lógica dentro de los controladores. Se recomienda mover la lógica compleja a servicios o modelos.
    b. Modelos (app/Models):
        i. Representan las tablas de la base de datos y proporcionan una forma de interactuar con los datos utilizando Eloquent.
        ii. Ejemplo: El modelo Empleado se utiliza para gestionar los datos de los usuarios administrativos.
        iii. Posible problema: Relaciones mal definidas entre modelos. Es importante verificar los nombres de claves foráneas y métodos como hasMany o belongsTo.
    c. Middleware (app/Http/Middleware):
        i. Actúan como filtros para las solicitudes HTTP. Controlan el acceso, la seguridad y otros procesos previos al envío de una solicitud al controlador.
        ii. Ejemplo: EmpleadoMiddleware verifica que solo los empleados autenticados puedan acceder a rutas administrativas.
2. Config. La carpeta config contiene archivos de configuración que definen cómo funciona el sistema.
    a. Archivos relevantes:
        i. config/auth.php: Define la configuración de autenticación, como los guards para empleado y usuarios regulares.
        ii. config/session.php: Gestiona las sesiones, incluida la duración y el almacenamiento.
        iii. config/database.php: Configura la conexión a la base de datos.
    b. Importancia:
        i. Centraliza la configuración del sistema, facilitando ajustes sin necesidad de modificar el código fuente directamente.
        ii. Ejemplo: Puedes cambiar el tiempo de expiración de las sesiones para garantizar la seguridad.
    c. Posibles problemas:
        i. Configuración incorrecta en el archivo auth.php puede romper el sistema de autenticación.
        ii. Falta de valores predeterminados en los entornos de producción o pruebas.
3. Database. La carpeta database contiene todos los elementos relacionados con la base de datos.
    a. Migraciones (database/migrations):
        i. Son archivos que definen la estructura de las tablas de la base de datos y permiten versionar cambios.
        ii. Ejemplo: La migración que crea la tabla libros incluye campos como nombre, autor, isbn, etc.
    b. Seeders (database/seeders):
        i. Son clases que se utilizan para poblar la base de datos con datos iniciales o de prueba.
        ii. Ejemplo: LibroSeeder podría cargar libros predefinidos en la base de datos.
    c. Importancia:

        i. Facilita la configuración inicial del sistema en nuevos entornos.
        ii. Permite mantener un historial de cambios en la base de datos.
    d. Posibles problemas:
        i. Si no se sincronizan correctamente las migraciones y los modelos, pueden surgir errores de incompatibilidad.
        ii. Migraciones conflictivas en equipos colaborativos.
4. Public. La carpeta public contiene recursos accesibles públicamente.
    a. Archivos relevantes:
        i. index.php: Es el punto de entrada principal de la aplicación.
        ii. Recursos estáticos como imágenes, hojas de estilo CSS y scripts JavaScript.
    b. Importancia:
        i. Los archivos de esta carpeta son directamente accesibles desde el navegador.
        ii. Ejemplo en el sistema: Las imágenes de portada de libros se almacenan en public/images.
    c. Posibles problemas:
        i. Subir archivos sensibles o ejecutables a esta carpeta puede representar un riesgo de seguridad.
        ii. Desorganización de recursos puede dificultar su mantenimiento.
5. Resources. La carpeta resources contiene vistas y recursos sin compilar.
    a. Vistas (resources/views):
        i. Plantillas Blade utilizadas para generar HTML dinámico.
        ii. Ejemplo: rentas.blade.php genera la lista de rentas en la interfaz.
    b. Assets (resources/css y resources/js):
        i. Contienen archivos CSS y JavaScript que se compilan para optimizar el rendimiento en producción.
    c. Importancia:
        i. Organiza la presentación del sistema y separa la lógica de negocio de la interfaz.
        ii. Facilita la reutilización y modularidad de componentes visuales.
    d. Posibles problemas:
        i. Si las vistas no siguen una estructura coherente, puede ser difícil localizar plantillas específicas.
        ii. Cargar demasiados assets no optimizados puede afectar el rendimiento.
6. Routes. La carpeta routes contiene archivos que definen las rutas de la aplicación.
    a. Archivos relevantes:
        i. routes/web.php: Define las rutas para solicitudes HTTP.
        ii. routes/api.php: Contiene las rutas para API RESTful.
    b. Importancia:
        i. Centraliza la configuración de las URL, asociándolas con controladores y acciones.
        ii. Ejemplo: La ruta GET /admin/rentas dirige al método index de RentaController.
    c. Posibles problemas:
        i. Rutas duplicadas pueden causar errores de redirección.
        ii. No aplicar middleware adecuado puede exponer rutas sensibles.
7. Storage. La carpeta storage contiene archivos generados por el sistema.
    a. Logs (storage/logs):
        i. Registro de errores y actividades.
        ii. Ejemplo: Si ocurre un error en el controlador, se guarda en laravel.log.
    b. Caché (storage/framework):

        i. Almacena datos en caché para optimizar el rendimiento.
    c. Importancia:
        i. Almacena datos temporales y persistentes necesarios para el funcionamiento interno.
        ii. Los registros son útiles para la depuración.
    d. Posibles problemas:
        i. Si los permisos de la carpeta no están configurados correctamente, puede impedir que Laravel escriba logs.
        ii. Archivos grandes no gestionados pueden llenar el almacenamiento.
8. Tests. La carpeta tests contiene pruebas automatizadas.
    a. Archivos relevantes:
        i. tests/Feature: Pruebas que simulan interacciones de usuario con las rutas.
        ii. tests/Unit: Pruebas para métodos individuales de la lógica del negocio.
    b. Importancia:
        i. Garantiza que el sistema funcione correctamente después de realizar cambios.
        ii. Ejemplo: Una prueba podría verificar que el guard empleado protege las rutas administrativas.
    c. Posibles problemas:
        i. Falta de cobertura de pruebas puede permitir errores no detectados.
        ii. Pruebas mal diseñadas pueden dar falsos positivos.

Seguridad

- Encriptación: Contraseñas de usuarios encriptadas usando el algoritmo bcrypt.
- Middleware: Uso de middlewares personalizados para garantizar el acceso basado en roles, específicamente en el lado administrador.
- Protección CSRF: Implementada automáticamente por Laravel para formularios.

Despliegue

1. Subir los archivos del proyecto al servidor.
2. Configurar el archivo. env en el entorno de producción.
3. Ejecutar composer install --optimize-autoloade y php artisan config:cache.
4. Ejecutar php artisan serve para levantar el servidor

Explicación de los módulos

1. Módulo de administrador.

El módulo de administrador es la sección del sistema destinada a los usuarios con privilegios administrativos. Este módulo les permite gestionar los recursos del sistema y supervisar su funcionamiento. Sus características principales son:

- Gestión de recursos:
    - Libros: Registrar nuevos libros con información como título, autor, género, y unidades disponibles. Consultar el inventario de libros y actualizar detalles o eliminar registros. Generar reportes sobre libros más rentados o mejor calificados.

    - Material didáctico: Registrar, editar y eliminar materiales como proyectores, manuales, y marcadores. Controlar el inventario para conocer las unidades disponibles.
    - Aulas: Registrar nuevas aulas con detalles como nombre, capacidad y ubicación. Administrar la disponibilidad de las aulas según su uso o mantenimiento.
    - Gestión de rentas y devoluciones: Supervisar las rentas realizadas por los clientes, tanto de libros como de materiales y aulas. Registrar devoluciones y calcular penalizaciones, si aplica. Visualizar el historial de rentas y devoluciones de un cliente o un recurso.
    - Reportes y análisis: Generar reportes de actividad del sistema, como estadísticas de uso y disponibilidad de recursos. Analizar patrones de uso para optimizar la gestión de recursos.

2.Módulo de cliente.

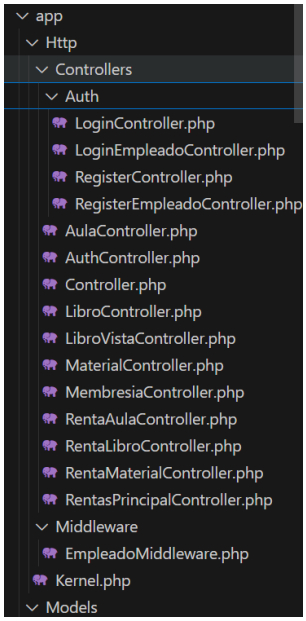
El módulo de cliente está diseñado para usuarios finales que desean interactuar con el sistema para consultar, rentar y devolver recursos. Sus principales funciones son:

- Consulta de recursos:
    - Navegar por un catálogo de libros, materiales y aulas disponibles para renta.
    - Aplicar filtros por categoría, género o disponibilidad.
    - Ver detalles de cada recurso, como descripción, calificaciones y unidades disponibles.
- Gestión de rentas:
    - Solicitar la renta de recursos disponibles:
        - Libros: Selección del libro deseado y número de unidades a rentar.
        - Material didáctico: Elección del tipo de material y la cantidad requerida.
        - Aulas: Reservar un aula especificando la fecha y duración de uso.
    - Revisar el historial personal de rentas activas y pasadas.
- Devoluciones:
    - Registrar la devolución de un recurso prestado, indicando la fecha en que se devuelve.
    - Ver si se aplican penalizaciones por retraso en la devolución.
- Gestión de cuenta:
    - Registro y autenticación para acceder al sistema.
    - Modificación de información personal, como nombre, correo electrónico o contraseña.
    - Revisión de membresías activas y sus beneficios.
- Calificación y comentarios:
    - Permitir a los clientes calificar los recursos rentados.
    - Dejar comentarios para mejorar la experiencia de otros usuarios y facilitar decisiones informadas.

Documentación del código

El proyecto SoftLibrary se creó utilizando el framework Laravel, lo que facilita la implementación de la arquitectura Modelo-Vista-Controlador (MVC). Esta separación asegura un código organizado, escalable y fácil de mantener, dividiendo claramente la lógica del negocio, las interfaces de usuario y la interacción con la base de datos.

Los Modelos, los Controladores y los Middleware se encuentran en la carpeta ”App/HTTP”.



Los modelos manejan la estructura básica de los datos y las relaciones entre las tablas de la base de datos, definiendo cómo se interactúa con la información almacenada. Por ejemplo, el modelo Libro en SoftLibrary define los atributos que un libro debe tener (como nombre, autor, genero, isbn, etc.) y cómo estos se relacionan con otras entidades, como usuarios o rentas. Los modelos utilizan Eloquent ORM para simplificar estas interacciones.

Los controladores contienen la lógica del negocio y sirven como intermediarios entre los modelos y las vistas. Por ejemplo, el LibroController gestiona las operaciones relacionadas con los libros, como listar todos los disponibles, agregar uno nuevo o editar su información. Cada acción del usuario que llega a través de una ruta se maneja en un método del controlador, lo que asegura que el flujo de datos sea claro y estructurado.

Las Vistas, por otro lado, se encuentran en la carpeta "resources/views", y contienen los archivos Blade encargados de renderizar el contenido para el usuario final. Estas vistas se encargan de presentar la información de manera amigable, utilizando los datos que los controladores les envían.

Además, las Rutas, definidas en "routes/web.php", actúan como la puerta de enlace entre las solicitudes del usuario y los controladores. Cada ruta está asociada a un controlador y un método específico, lo que hace que el flujo de la aplicación sea intuitivo y sencillo de seguir.

Conclusión

El proyecto SoftLibrary se consolida como una solución robusta y escalable para la gestión integral de bibliotecas y centros educativos. Al aprovechar las ventajas del framework Laravel y su arquitectura Modelo-Vista-Controlador (MVC), el sistema garantiza una separación clara de responsabilidades, facilitando el desarrollo, mantenimiento y futura expansión de la aplicación.

Este manual técnico ha sido diseñado para servir como una referencia detallada para desarrolladores y administradores del sistema. A través de sus secciones, se ha proporcionado una visión completa de los componentes clave del proyecto, incluyendo la estructura del código, la configuración del entorno, los módulos funcionales y las medidas de seguridad implementadas. SoftLibrary no solo es un sistema para la gestión de bibliotecas, sino también un ejemplo de cómo una solución tecnológica bien diseñada puede transformar procesos, optimizar recursos y brindar una experiencia de usuario de calidad.