TRABAJO INTEGRADOR

PROGRAMACIÓN ORIENTADA A OBJETOS 1

Docentes de la catedra:

Esp. Lic. Claudio Omar Biale
Melissa Kolb

Estudiantes:

Küster Joaquín
Bialy Liam
Antón Belen

Universidad:

UNaM - Apostoles, Misiones

2024

Índice

1. Introducción

Para comenzar con el desarrollo de nuestro sistema de gestión de una biblioteca, nos guiamos por el ejemplo brindado por la cátedra, el cual sirvió como base para entender los requerimientos y funcionalidades básicas que debe tener un sistema de este tipo. Este ejemplo nos proporcionó un marco inicial sobre el cual desarrollar, permitiéndonos adaptar y expandir las características del sistema para satisfacer las necesidades específicas de una biblioteca.
El sistema de gestión de biblioteca ha sido diseñado para facilitar y optimizar la administración de las funciones principales de una biblioteca. Este sistema se ha desarrollado con el objetivo de proporcionar una solución integral para la gestión de libros, copias, miembros de la biblioteca y préstamos, garantizando una experiencia eficiente tanto para los usuarios como para los bibliotecarios.
El sistema tiene como propósito manejar todas las operaciones esenciales relacionadas con la administración de una biblioteca, incluyendo:

- Validación de Miembros: Los usuarios y bibliotecarios deben ingresar al sistema utilizando un DNI y una clave de acceso, lo que asegura que sólo miembros registrados puedan acceder a las funcionalidades de la biblioteca.

- Búsqueda de Libros: Los miembros podrán buscar libros en la biblioteca utilizando criterios como título, autor o categoría temática, facilitando el acceso a la información relevante.

- Gestión de Libros y Copias: Cada libro tiene un ISBN de identificación único y se asocia con detalles específicos como editorial, autores, categoría temática e idioma. Además, cada libro puede tener múltiples copias, cada una con su propia ubicación en un rack específico.

- Control de Préstamos: Los miembros pueden tomar prestadas copias de libros, pero el sistema impondrá un límite máximo de 5 copias por miembro y un periodo máximo de préstamo de 10 días. Las copias pueden estar disponibles, prestadas o marcadas como pérdida, y algunas copias pueden estar clasificadas como de referencia, lo que impide su préstamo.

- Registro de Actividades: El sistema permitirá recuperar información sobre los libros prestados y quién los ha tomado, así como los libros que un miembro ha sacado.

- Gestión de Multas: En caso de retraso en la devolución de libros, el sistema calculará automáticamente las multas, basadas en el número de días de retraso y el precio estimado del libro. La gestión de multas es parte de otro sistema.

Entre las tecnologías utilizadas se encuentran Java para la lógica de programación, JavaFX para la interfaz gráfica de usuario, JPA y EclipseLink para la persistencia de datos en una base de datos PostgreSQL, y Maven para la gestión de dependencias y la organización del proyecto.

A lo largo de este documento, describiremos en detalle cada componente del sistema, desde la estructura del proyecto y las tecnologías empleadas hasta la lógica de negocios y el funcionamiento del sistema, proporcionando una visión completa de nuestro desarrollo.

---

2. Desarrollo y Versionado

Adoptamos una metodología iterativa e incremental para el desarrollo del proyecto. A medida que avanzábamos, identificamos errores o posibles mejoras, lo que nos permitía retroceder, hacer los ajustes necesarios y continuar progresando de manera más sólida.
Para el desarrollo colaborativo, utilizamos un entorno compartido en Visual Studio Code, aprovechando la extensión Live Share. Esta herramienta nos permitió trabajar de manera remota en el mismo proyecto de forma simultánea, facilitando la colaboración en tiempo real entre todos los miembros del equipo. Es importante destacar que, aunque cada integrante trabajaba desde su propia computadora, uno de los miembros compartía su entorno de desarrollo, permitiendo a los demás acceder a su máquina independientemente del sistema operativo que estuvieran utilizando.
Para el control de versiones, utilizamos un repositorio en GitHub. Cada cambio realizado en el proyecto fue cuidadosamente documentado mediante commits descriptivos, lo que garantiza la trazabilidad y el seguimiento detallado del progreso. Una vez que se implementan mejoras o se solucionaban problemas en el entorno compartido, los cambios se guardaban en la máquina local del miembro que había compartido el entorno y se subían al repositorio remoto para mantener sincronizado el proyecto.
Para evitar problemas de compatibilidad y configuración, definimos con antelación todas las dependencias necesarias, las cuales detallamos en la sección anterior. Estas dependencias y sus versiones específicas se fijaron en el archivo pom.xml gestionado por Maven, lo que nos permitió asegurar un entorno de desarrollo consistente para todos los integrantes del equipo.

---

3. Estructura del Proyecto

El proyecto se organiza y gestiona mediante Maven, lo que facilita la administración de dependencias, la compilación del código y la estructura del proyecto. Maven sigue una convención que divide el proyecto en carpetas claras y específicas, lo que asegura un entorno de desarrollo limpio y organizado.

3.1. Uso de Maven

Maven se utiliza para manejar las dependencias del proyecto, permitiendo que todas las bibliotecas necesarias sean gestionadas y actualizadas de manera centralizada. Las dependencias principales incluyen:

- JavaFX: Para la construcción de la interfaz gráfica de usuario (GUI).

- EclipseLink: Para el mapeo objeto-relacional (ORM) y la interacción con la base de datos.
- PostgreSQL: El driver para la conexión con la base de datos PostgreSQL.
- SLF4J: Para la gestión de los logs en la aplicación.
- ControlsFX: Librería que añade controles adicionales y funcionalidades a JavaFX.
- Spring Security Crypto: Para la codificación y manejo seguro de contraseñas.

3.2. Organización de Carpetas y Archivos

Maven organiza el código fuente y los recursos del proyecto en dos directorios principales:

- src/main/java: Contiene todo el código fuente de la aplicación.
- src/main/resources: Incluye todos los recursos que se utilizan en la aplicación, como archivos FXML, imágenes, archivos CSS, y configuraciones.

Dentro de la carpeta src/main/java, el proyecto está estructurado en varios paquetes, cada uno con una función específica:

Carpeta		Descripción
Config	Clase	
	AppConfig.java	Declara y configura una instancia del repositorio estático, utilizado de manera global y compartida por todos los controladores de la aplicación.
	StageManager.java	Se encarga de la gestión de la pantalla principal de la aplicación y de los modales que se utilizan para formularios y selectores de búsqueda.
Controller		Contiene los controladores que gestionan la lógica de las

Carpeta		Descripción
		diferentes vistas FXML y las operaciones de los ABM (Altas, Bajas, Modificaciones).
Helper	Clase	
	Alerta.java	Contiene métodos estáticos para mostrar alertas de éxito o error a los usuarios.
	ControlUI.java	Maneja configuraciones específicas de algunos elementos visuales (como Spinner, DatePicker, etc.).
	Fecha.java	Proporciona métodos para formatear fechas del tipo LocalDate.
	Selector.java	Facilita el uso de selectores de búsquedas, específicamente en ComboBox con muchos elementos.
	Validacion.java	Contiene métodos estáticos para validar diferentes campos utilizando expresiones regulares (por ejemplo, DNI, ISBN, contraseñas, nombres).
Model		Define las clases de modelo que mapean las entidades de la base de datos en clases Java. Estas clases representan la estructura de los datos que se manejan en la aplicación.
Repository		Contiene un repositorio genérico que interactúa con la base de datos,

Carpeta		Descripción
		proporcionando métodos CRUD básicos para las entidades.
Security	Clase	
	SesionManager.java	Gestiona la sesión activa del usuario, permitiendo abrir, cerrar sesiones y acceder a los datos personales del miembro de la biblioteca que ha iniciado sesión.
	Contraseña.java	Proporciona métodos estáticos para la codificación y decodificación segura de contraseñas.
Service		Contiene los diferentes servicios que heredan de CrudServicio, una clase base que implementa las operaciones CRUD comunes (buscar, insertar, modificar, eliminar). Cada servicio específico añade las validaciones y la lógica de negocio correspondiente.
View		Define un Enum que incluye los títulos y las rutas de los archivos FXML para cada vista de la aplicación.

3.3. Recursos

Dentro de src/main/resources, se organizan los siguientes recursos:

- Archivos FXML: Representan la estructura y diseño de cada vista y ABM del sistema.
- Imágenes: Incluyen el logotipo del sistema e íconos utilizados en la interfaz, en formato PNG.

- Estilos CSS: Archivos CSS que definen los estilos visuales aplicados a las vistas FXML.

- META-INF: Contiene el archivo persistence.xml, que se utiliza para configurar la unidad de persistencia y definir las conexiones a la base de datos.

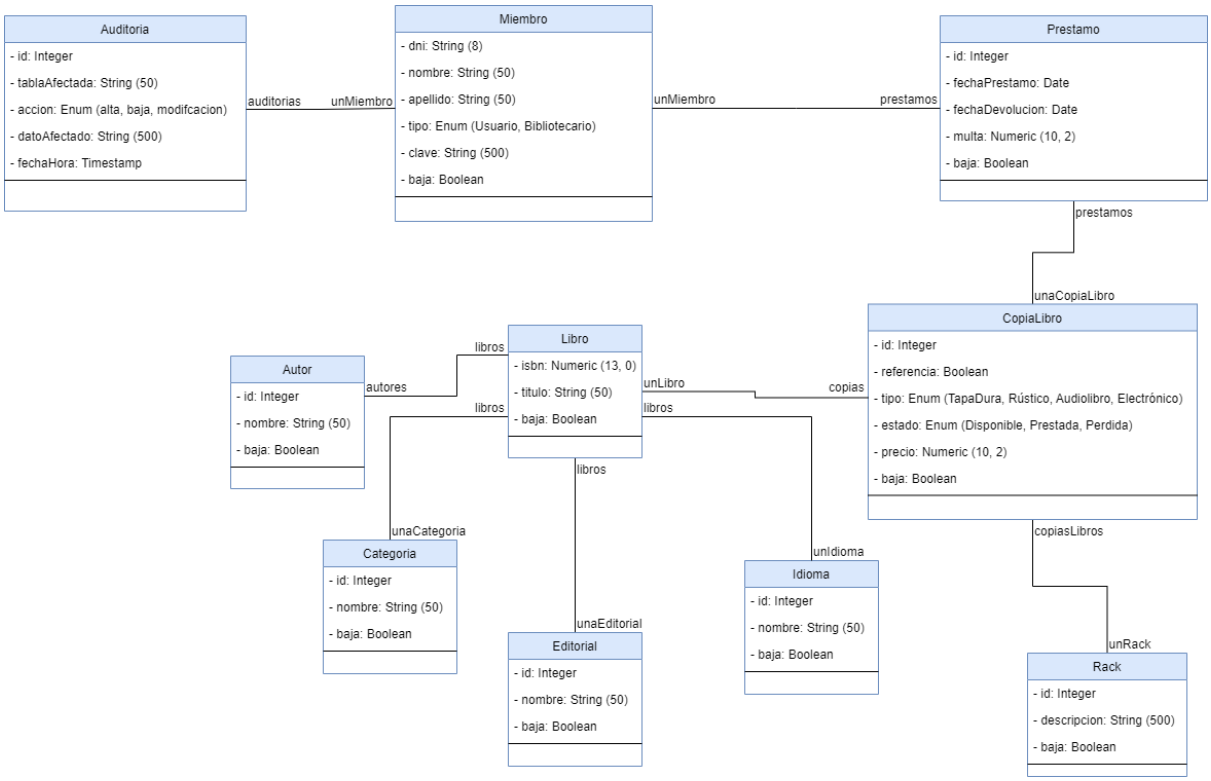
Aclaración: La carpeta target se crea y se utiliza después de compilar el proyecto. Durante el proceso de construcción de Maven, target es el directorio donde se colocan los archivos resultantes de las fases de compilación, empaquetado y otras tareas de construcción. Por ejemplo, después de compilar el código fuente, los archivos .class se colocan en target/classes. Luego, cuando se empaqueta el proyecto, el archivo .jar o .war se genera en target.

---

4. Base de Datos y Diagrama de Clases

4.1. Diagrama de Clases

A continuación se presenta el diagrama de clases de nuestro sistema, el cual representa las entidades principales y sus relaciones. Este diagrama sirve como una abstracción del modelo lógico que se utilizó para implementar las funcionalidades en el sistema:



Descripción de las relaciones:

1. Relación entre Auditoría y Miembro:

    - Tipo de Relación: Uno a Muchos.

    - Descripción: Un miembro puede tener muchas auditorías y una auditoría puede pertenecer a un miembro

2. Relación entre Miembro y Préstamo:

    - Tipo de relación: Uno a muchos.

    - Descripción: Un Miembro puede realizar múltiples Préstamos, pero cada Préstamo está asociado a un único Miembro. Esta relación es crucial porque permite rastrear qué libros ha tomado prestado cada miembro de la biblioteca.

    - Implementación: La tabla Préstamo tiene una clave foránea (unMiembro) que se refiere al dni del Miembro.

3. Relación entre Libro y CopiaLibro:

    - Tipo de relación: Uno a muchos.

    - Descripción: Un Libro puede tener varias CopiaLibro, que representan las distintas copias físicas o digitales de ese libro en la biblioteca. Cada CopiaLibro pertenece a un solo Libro.

    - Implementación: La tabla CopiaLibro tiene una clave foránea (unLibro) que se refiere al ISBN del Libro.

4. Relación entre CopiaLibro y Préstamo:

    - Tipo de relación: Uno a muchos.

    - Descripción: Cada préstamo se refiere a una única CopiaLibro que ha sido prestada. Sin embargo, una CopiaLibro puede estar asociada a múltiples préstamos si ha sido prestada en diferentes momentos.

    - Implementación: La tabla préstamo tiene una clave foránea (unaCopiaLibro) que se refiere al id de CopiaLibro.

5. Relación entre Libro y Autor:

    - Tipo de relación: Muchos a muchos.

- Descripción: Un libro puede tener varios autores y un autor puede haber escrito varios libros. Esta relación es gestionada a través de una tabla intermedia (implícita en el diagrama), que conecta múltiples Autores con múltiples Libros.

- Implementación: Aunque no está explícitamente mostrada en el diagrama, se suele crear una tabla intermedia (LibrosAutores) con dos claves foráneas: una referida al id del Autor y otra al isbn del Libro.

6. Relación entre Libro y Categoría:

    - Tipo de relación: Uno a muchos.

    - Descripción: Un Libro pertenece a una única Categoría, pero una Categoría puede tener muchos Libros. Esto ayuda a clasificar los libros en diferentes géneros o tipos.

    - Implementación: La tabla Libro tiene una clave foránea (unaCategoria) que se refiere al id de Categoría.

7. Relación entre Libro y Editorial:

    - Tipo de relación: Uno a muchos.

    - Descripción: Un Libro es publicado por una única Editorial, pero una Editorial puede publicar muchos Libros. Esta relación es importante para saber quién ha editado un libro.

    - Implementación: La tabla Libro tiene una clave foránea (unaEditorial) que se refiere al id de Editorial.

8. Relación entre CopiaLibro y Rack:

    - Tipo de relación: Uno a muchos.

    - Descripción: Cada CopiaLibro se encuentra en un único Rack (estante), pero un Rack puede contener muchas copias de libros. Esta relación permite localizar físicamente las copias de los libros en la biblioteca.

    - Implementación: La tabla CopiaLibro tiene una clave foránea (unRack) que se refiere al id de Rack.

9. Relación entre Libro y Idioma:

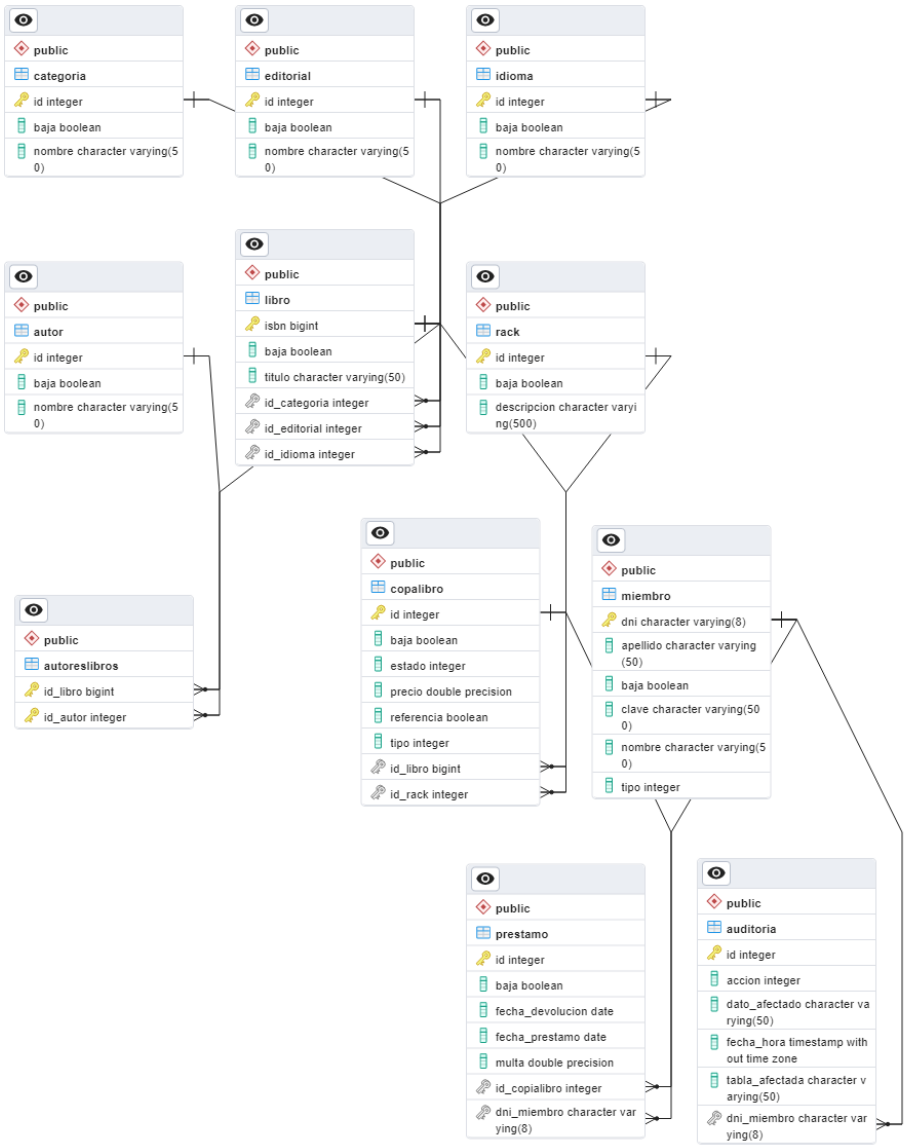
- Tipo de relación: Uno a muchos.

- Descripción: Un libro está en un único idioma, pero un idioma puede estar asociado a muchos libros. Esto es útil para gestionar la disponibilidad de libros en diferentes idiomas.

- Implementación: La tabla Libro tiene una clave foránea (unIdioma) que se refiere al id de Idioma

4.2. Diagrama Entidad-Relación (ER)

El diagrama ER ilustra cómo las diferentes tablas están relacionadas entre sí en la base de datos. A continuación se presenta el diagrama, donde se pueden observar las entidades principales como Miembro, Libro, Autor, CopiaLibro, entre otras, y sus respectivas relaciones.



4.3. Explicación de las Tablas

Las tablas principales de la base de datos son: Auditoría, Autor, AutoresLibros, Categoría, CopiaLibro, Editorial, Idioma, Libro, Miembro, Préstamo, y Rack. Cada tabla cumple un rol específico dentro del sistema, como se detalla a continuación.

- Auditoría: Almacena los registros de acciones realizadas en el sistema.
- Autor: Contiene la información de los autores que han publicado los libros disponibles en la biblioteca.
- AutoresLibros: Tabla intermedia que relaciona Autores con Libros gestionando la cardinalidad de muchos a muchos que se encuentra en estas dos entidades.
- Categoría: Se encuentran las categorías que pueden tener los libros disponibles en la biblioteca.
- CopiaLibro: Almacena la información de las copias disponibles que pueden tener los libros registrados en la biblioteca, como el tipo de copia, el precio y si es de referencia.
- Editorial: Almacena la información de las editoriales de los libros que se encuentran en el sistema.
- Idioma: Se registran los idiomas disponibles de los libros del sistema.
- Libro: En esta tabla están todas las publicaciones que tiene la biblioteca.
- Miembro: Contiene la información de los miembros que pertenecen a la biblioteca.
- Préstamo: Almacena los registros de los préstamos realizados en la biblioteca a cada miembro que haya adquirido un libro.
- Rack: Se registran los racks donde se almacenan los libros físicamente en la biblioteca.

4.4. Tecnologías Utilizadas

La base de datos fue implementada utilizando PostgreSQL, un sistema de gestión de bases de datos relacional de código abierto. La persistencia de los datos se maneja a través de EclipseLink, una implementación de la Java Persistence API (JPA), que facilita el mapeo entre las clases Java y las tablas de la base de datos.

4.5. Descripción Acerca de la Tabla Intermedia AutoresLibros

En el modelo de datos, las entidades Autor y libro tienen una relación de muchos a muchos, lo que significa que un autor puede haber escrito varios libros y un libro puede haber sido escrito por varios autores. Para modelar esta relación, se utilizó la anotación @ManyToMany en ambas entidades.
En la entidad Libro, la relación se define utilizando la anotación @ManyToMany junto con @JoinTable, especificando explícitamente el nombre de la tabla intermedia (autoreslibros). Esta tabla intermedia no aparece directamente como una clase o entidad en el código fuente, sino que es generada automáticamente por Hibernate cuando el sistema se inicializa
La tabla intermedia autoreslibros contiene dos columnas que actúan como claves foráneas:

- id_libro: Referencia al ISBN del libro en la tabla libro.

- id_autor: Referencia al ID del autor en la tabla autor.

Este mapeo se define mediante las anotaciones @JoinColumn, que indican las columnas que actuarán como las claves foráneas en la tabla intermedia. Al utilizar esta configuración, Hibernate se encarga de generar la tabla autoreslibros de manera automática y gestionar las inserciones, actualizaciones y eliminaciones de los registros en esta tabla cuando se manipulan las relaciones entre autores y libros en el código de la aplicación.

4.6. Enumeraciones (Enum) Utilizadas

Además de las relaciones entre entidades, se utilizaron varias enumeraciones (enum) para definir de manera clara y concisa ciertos tipos de datos constantes en el sistema. Las enumeraciones proporcionan un conjunto fijo de valores posibles, lo que facilita la validación y optimiza la base de datos. A continuación, se describen los enum utilizados y su propósito:

EstadoCopiaLibro:

- **Valores:** Disponible, Prestada, Perdida.

- **Descripción:** Este enum se utiliza para representar el estado de las copias físicas o electrónicas de los libros en la biblioteca. Por ejemplo, cuando una copia de un libro está disponible para ser prestada, su estado será Disponible. Si la copia ha sido prestada a un usuario, su estado cambiará a Prestada. En el caso de que la copia haya sido reportada por el bibliotecario como extraviada, su estado será Perdida.

TipoAccion:

- **Valores:** Alta, Baja, Modificación.
- **Descripción:** Este enum se utiliza en el sistema de auditoría para registrar las acciones realizadas sobre las entidades de la base de datos. Por ejemplo, cuando se agrega un nuevo libro al sistema, se registra una acción de tipo Alta. Si se elimina un autor, se registra una acción de tipo Baja. Y si se modifica la información de una copia de un libro, se registra una acción de tipo Modificación.

- Aclaración: El borrado lógico también se registra como baja.

TipoCopiaLibro

- **Valores:** TapaDura, Rústico, AudioLibro, Electrónico.
- **Descripción:** Este enum clasifica las diferentes variantes de una copia de un libro disponible en la biblioteca. TapaDura y Rústico se refieren a copias físicas, mientras que AudioLibro y Electrónico hacen referencia a formatos digitales. Esta enumeración permite a los usuarios y al sistema identificar y gestionar correctamente las diferentes versiones de una copia de un libro.

TipoMiembro

- **Valores:** Usuario, Bibliotecario.
- **Descripción:** Este enum clasifica los diferentes tipos de usuarios que interactúan con el sistema. Los Usuarios son los miembros regulares de la biblioteca que pueden pedir préstamos, mientras que los bibliotecarios son los administradores del sistema que tienen permisos adicionales para gestionar el los libros, copias, usuarios, y otras operaciones críticas.

En lugar de almacenar cadenas de texto que representan estos valores en las tablas, las enumeraciones permiten almacenar de manera más eficiente enteros o cadenas predefinidas que corresponden a los valores permitidos. Hibernate maneja automáticamente la conversión de estos valores entre su representación en la base de datos y las variables de tipo **enum** en el código Java.

5. Arquitectura MVC

El sistema de gestión de biblioteca está diseñado siguiendo el patrón de arquitectura MVC (Modelo-Vista-Controlador), que facilita la organización del código, promueve la separación de responsabilidades y mejora la mantenibilidad y escalabilidad del sistema. A continuación, se describe cómo se implementa cada componente de esta arquitectura en el sistema.

5.1. Modelo

El Modelo representa la capa de datos y las reglas de negocio del sistema. En este contexto, el modelo está compuesto por una serie de clases que encapsulan las entidades de la biblioteca, como Auditoría, Autor, Categoría, CopiaLibro, Editorial, Idioma, Libro, Miembro, Préstamo, Rack, entre otras. También incluye enumeraciones como EstadoCopiaLibro, TipoCopiaLibro, y TipoMiembro.

5.1.1. Validaciones en el Modelo

El Modelo se encarga de realizar validaciones que están directamente relacionadas con las propiedades de cada entidad. Estas validaciones aseguran que los objetos se encuentren en un estado válido antes de ser utilizados o almacenados en la base de datos, dentro de la lógica de negocios explicamos qué tipo de validaciones se realizan.
Este enfoque garantiza que las entidades del sistema siempre mantengan su integridad interna antes de interactuar con otras partes del sistema, como la base de datos.

¿Por qué realizar validaciones dentro del modelo? Las validaciones en el modelo aseguran que los objetos del modelo se crean con datos válidos desde el principio, evitando instancias inválidas o inconsistentes. Esto garantiza que cada objeto cumpla con las reglas definidas, independientemente de cómo se cree la instancia (a través de un constructor o un setter).

5.1.2. Mapeo de Datos y Persistencia con Jakarta y EclipseLink

En el modelo de datos, utilizamos Jakarta y EclipseLink para mapear la base de datos a clases Java mediante anotaciones JPA. Estas anotaciones permiten definir cómo se debe almacenar y recuperar cada entidad en la base de datos.

5.1.2.1. Anotaciones de JPA en el Modelo de Datos

Anotaciones Principales:

- @Entity: Indica que una clase es una entidad JPA y se corresponde con una tabla en la base de datos.

- @Table(name = "auditoria"): Especifica el nombre de la tabla en la base de datos a la que se mapea la entidad.

- @Id: Marca el campo que actúa como la clave primaria de la entidad.

- @Column(name = "nombre", length = 50, nullable = false): Define las características de una columna en la base de datos, como su nombre, longitud máxima y si puede ser nula.

- @GeneratedValue(strategy = GenerationType.IDENTITY): Indica que el valor del campo se generará automáticamente como un ID autoincrementable.

Relaciones entre Entidades:

- @ManyToOne y @OneToMany: Estas anotaciones definen relaciones entre entidades. Por ejemplo, en la relación entre Miembro y Préstamo:

    - En la entidad Préstamo, usamos @ManyToOne para definir que cada préstamo está asociado a un único miembro: @JoinColumn(name = "dni_miembro", nullable = false) especifica el nombre de la columna de la clave foránea en la tabla préstamo y que este campo no puede ser nulo.

    - En la entidad Miembro, usamos @OneToMany para definir que un miembro puede tener múltiples préstamos: @OneToMany(mappedBy = "miembro", cascade = CascadeType.ALL, orphanRemoval = true), donde mappedBy indica que la relación es gestionada por el campo miembro en la entidad Préstamo.

cascade = CascadeType.ALL permite que las operaciones en la entidad Miembro se propaguen a sus préstamos asociados, y orphanRemoval = true elimina automáticamente los préstamos que ya no están asociados a ningún miembro.

- @ManyToMany: Define relaciones muchos a muchos, como entre Autor y Libro:
    - En la entidad Libro: @JoinTable especifica la tabla intermedia autoreslibros que almacena las relaciones entre libros y autores, y joinColumns e inverseJoinColumns definen las columnas en esa tabla que se corresponden con las claves primarias de Libro y Autor, respectivamente.
    - En la entidad Autor: @ManyToMany(mappedBy = "autores"), donde mappedBy indica que esta entidad está mapeada por el campo autores en la entidad Libro.

5.1.3. Constructores y Métodos

- Constructor vacío: ¿Por qué? EclipseLink utiliza la reflexión para crear instancias de entidades desde la base de datos. La reflexión en Java permite inspeccionar y manipular clases y objetos en tiempo de ejecución. Sin un constructor sin argumentos, EclipseLink no podría crear instancias de las entidades, lo que generaría errores.

- Constructores con argumentos, setters y getters: Permiten crear y modificar objetos de manera controlada, incluyendo las validaciones necesarias para garantizar la integridad de los datos.

- Método toString(): Proporciona una representación en texto del objeto, mostrando información relevante en lugar del nombre de la clase y el código hash por defecto.

5.2. Vista

La Vista es responsable de la interfaz de usuario y la presentación de los datos. En el sistema de gestión de biblioteca, la vista incluye las pantallas con las que interactúan los usuarios, como formularios para agregar o modificar libros, tablas que muestran los registros de préstamos y devoluciones, y filtros para buscar y visualizar información específica.
La vista no contiene lógica de negocio; su única responsabilidad es capturar la interacción del usuario y mostrar los datos proporcionados por el controlador. Los componentes visuales, como ComboBox, TableView, y DatePicker, están diseñados para facilitar la interacción del usuario con el sistema. Por ejemplo, cuando un usuario selecciona un filtro en un ComboBox o ajusta una fecha en un DatePicker, se dispara un evento que es manejado por el controlador. En la sección donde se habla del funcionamiento del sistema, se explica detalladamente cómo se manejan las vistas.

5.3. Controlador

El Controlador actúa como un intermediario entre la vista y el modelo. Se encarga de procesar las entradas del usuario, interactuar con los servicios que manejan la lógica de negocio y actualizar la vista en consecuencia.
Contamos con un controlador por cada vista FXML, para mejorar la modularidad y facilitar el mantenimiento y la escalabilidad de la aplicación. Cada controlador se enfoca en una vista específica, manejando únicamente la lógica y los eventos relacionados con esa vista, lo que contribuye a un diseño más limpio y organizado.

5.3.1. Responsabilidad del Controlador

En el sistema de gestión de biblioteca, el controlador tiene varias responsabilidades clave:

- Inicialización de controles visuales: El controlador configura y llena componentes visuales como tablas y ComboBox con los datos correspondientes. Por ejemplo, al cargar una vista de libros, el controlador se asegura de que la tabla esté cargada con los registros actuales de libros.

- Gestión de eventos: El controlador captura eventos de usuario, como la selección de un filtro en un ComboBox o la escritura en un campo de texto. Para los ComboBox y DatePicker, los eventos se manejan con onAction, mientras que para los campos de texto, se utiliza onKeyReleased para filtrar los datos en tiempo real.

- Interacción con formularios modales: Al presionar un botón para agregar o modificar un registro, el controlador abre un formulario en una ventana emergente (modal) y, en el caso de modificar, pasa el objeto seleccionado al controlador del formulario para su edición.

- Ejecución de acciones: El controlador valida que un objeto esté seleccionado en una tabla antes de proceder a acciones como eliminación o modificación, delegando la lógica específica de estas operaciones a los servicios correspondientes.

Nota: Optamos por realizar las validaciones en el modelo y en los servicios en lugar de en el controlador, con el objetivo de asegurar una clara separación de responsabilidades, manteniendo el controlador enfocado en manejar el flujo de la aplicación y delegando la lógica de negocio y las validaciones al modelo y servicio.

5.4. Servicios

Los Servicios son una capa adicional que encapsula la lógica de negocio y las interacciones con la base de datos, asegurando que el Modelo no tenga que interactuar directamente con la base de datos y manteniendo la separación de responsabilidades.

5.4.1. Función de los Servicios

¿Por qué se realizan validaciones dentro de los servicios? Porque hay validaciones que requieren acceso a la base de datos, como verificar si un registro ya existe o si un miembro está inactivo, y esto no puede ir dentro del modelo, ya que el modelo no debería interactuar con la base de datos. Esto permite que el Modelo se mantenga enfocado en validaciones locales sin necesidad de interactuar con la base de datos directamente.
Además de las validaciones, justamente, los Servicios son responsables de ejecutar consultas y comandos en la base de datos, como insertar nuevos registros, modificar existentes, o eliminar registros.
Después de realizar las operaciones necesarias, los Servicios devuelven los resultados al Controlador, quien se encargará de procesar estos datos y actualizar la Vista según sea necesario.

5.5. Flujo de Datos

El flujo de responsabilidades en la arquitectura MVC de este sistema se puede resumir de la siguiente manera:

1. Vista -> Controlador: La Vista envía datos o eventos al Controlador basados en la interacción del usuario.

2. Controlador -> Servicio: El Controlador delega la lógica de negocio al Servicio correspondiente, enviando los datos necesarios para procesar la solicitud.

3. Servicio -> Modelo: El Servicio crea o manipula objetos del Modelo y realiza las validaciones relacionadas con el estado de los datos.

4. Servicio -> Repositorio/Base de Datos: El Servicio ejecuta operaciones en la base de datos como inserciones, actualizaciones o consultas, manteniendo la integridad de los datos.

5. Servicio -> Controlador: El Servicio devuelve los resultados o el estado de la operación al Controlador.

6. Controlador -> Vista: El Controlador actualiza la Vista con los datos o resultados obtenidos, cerrando el ciclo de interacción.

---

6. Capa de Negocio

La capa de negocio, también conocida como capa de servicio o lógica de negocio, sirve para encapsular las reglas y la lógica del negocio que determinan el funcionamiento del sistema. Esta capa actúa como un intermediario entre la capa de presentación (interfaz de usuario) y la

capa de datos (acceso a base de datos), asegurando que las operaciones y procesos se realicen de manera consistente y coherente.
En nuestro sistema de gestión de biblioteca, la capa de negocio incluye:

- Servicios CRUD: Maneja la creación, lectura, actualización y eliminación de entidades clave como libros, miembros de la biblioteca y préstamos. Estos servicios garantizan que las operaciones se realicen conforme a las reglas de negocio definidas.

- Reglas de Validación: Implementa validaciones específicas para asegurar la integridad y precisión de los datos, como verificar fechas de devolución, el formato de un DNI o ISBN específico.

- Procesos de Negocio: Secuencias de actividades y reglas que determinan cómo se gestionan los diferentes aspectos de la biblioteca, como la gestión de los préstamos y las copias de los libros.

6.1. Gestion de Prestamos

En el sistema de gestión de préstamos, el proceso comienza al registrar un nuevo préstamo. Se solicita al usuario que ingrese la fecha del préstamo, la cual no debe ser anterior a la fecha actual. Durante este proceso, se seleccionan el miembro que realizará el préstamo y la copia del libro que se llevará.
Antes de registrar el préstamo, se realizan una serie de validaciones dentro del servicio, incluyendo:

- Verificación de que tanto el usuario como la copia estén activos y registrados correctamente.

- Comprobación de que el miembro no tenga más de 5 préstamos en curso y que cualquier préstamo previo haya sido devuelto a tiempo.

- Validación de que la copia no sea de referencia, no esté actualmente prestada, y no se haya reportado como pérdida.

Si todas las validaciones son satisfactorias, el préstamo se registra en el sistema. Posteriormente, el préstamo puede ser cerrado al confirmar la devolución mediante un botón en el sistema o mediante el acceso a un formulario para modificar el préstamo creado. En el formulario, solo se puede modificar la fecha de devolución. Esta fecha puede ser nula hasta que la copia sea devuelta, pero si se ingresa, se valida que no sea posterior a la fecha actual ni anterior a la fecha del préstamo (aunque puede coincidir con el día del préstamo).
Al ingresar una fecha de devolución, se calcula automáticamente la multa correspondiente si la devolución se realizó después del plazo de 10 días desde el préstamo. La multa se calcula multiplicando los días de retraso por el precio específico de la copia, que varía según el tipo del libro (tapa dura, tapa blanda o rústico, electrónico o audiolibro).

Durante el período del préstamo, la copia cambia su estado de "disponible" a "prestada". Al registrar la fecha de devolución, el estado cambia de nuevo a "disponible".

6.2. Clasificación de Copias de Libros: Físicas vs. Virtuales

Es importante distinguir entre copias de libros físicas y virtuales cuando hablamos de copias de referencia.

- Copias Físicas de Referencia: Las copias físicas de referencia son aquellas que no pueden ser retiradas del lugar, ya que están destinadas a ser usadas dentro de la biblioteca. Su principal función es servir como recurso para consultas o impresiones adicionales. Estas copias permanecen en el establecimiento para evitar su pérdida o deterioro y garantizar que estén siempre disponibles para su consulta.

- Copias Virtuales y Licencias: En el caso de los libros virtuales, como eBooks y audiolibros, el concepto de "referencia" se relaciona con las licencias adquiridas por la biblioteca. La biblioteca puede obtener estas licencias a través de plataformas como Scribd, Libby, Google Play Libros, OverDrive o CloudLibrary. Cuando se presta un libro virtual, se proporciona un enlace a través de la plataforma para acceder al contenido.

Aspectos a Tener en Cuenta:

- Cuentas en la Plataforma: Tanto la biblioteca como el miembro deben tener cuentas registradas en la plataforma para acceder al contenido del libro virtual.

- Términos de la Licencia: La capacidad de compartir un libro virtual con múltiples personas o dispositivos depende de los términos específicos de la licencia adquirida. Algunas licencias permiten que el contenido sea accedido por varios usuarios simultáneamente, mientras que otras no.

Licencias de Referencia en el Contexto Virtual:

- eBooks: Una licencia de referencia para un libro electrónico podría permitir la distribución física del contenido digital, es decir, la creación de copias físicas del libro a partir del formato digital. Sin embargo, en la mayoría de los casos, esto no está permitido debido a las restricciones impuestas por los términos de la licencia. Por lo tanto, la posibilidad de realizar copias físicas a partir del contenido digital depende completamente de los términos específicos de la licencia adquirida.

- Audiolibros: Para los audiolibros, una licencia de referencia suele estar destinada a usuarios con discapacidades, asegurando que el contenido sea de uso exclusivo para este grupo y no pueda ser prestado o retirado. Esto garantiza que los recursos estén disponibles para quienes más lo necesitan.

6.1. Definición de Servicios CRUD con Eliminación Lógica

En la capa de negocio de la aplicación, se emplea el eliminado lógico para gestionar registros, marcándolos como inactivos mediante un atributo booleano llamado baja. Este enfoque permite conservar la integridad referencial y evita la eliminación física de datos, lo cual es útil para mantener el historial y las relaciones entre entidades.
Dentro del paquete service, se encuentra la clase abstracta CrudServicio.java, que define un conjunto de operaciones comunes para la gestión de entidades. Esta clase abstracta incluye métodos como buscarTodosActivos, buscarPorId, insertar, modificar, y borrar. Estos métodos permiten realizar operaciones básicas de CRUD, adaptadas para manejar el estado de los registros.
Además, CrudServicio define un método existe, que recibe una entidad y su ID. Este método tiene el propósito de verificar en la base de datos si la entidad con el ID proporcionado sigue existiendo y está activa. Esta validación es crucial para mantener la estabilidad del sistema, especialmente cuando se insertan o asocian elementos con otros registros, asegurando que las entidades relacionadas aún estén vigentes y no hayan sido eliminadas.
La clase CrudServicio incluye métodos abstractos como validarEInsertar, validarYModificar, validarYEliminar, marcarComoInactivo, y esInactivo. Estos métodos son abstractos porque las operaciones específicas, como la comprobación del estado de inactividad, requieren acceso a propiedades particulares de cada entidad. Al trabajar con un objeto genérico T, no se conoce de antemano la implementación específica de estos atributos, por lo que se define que cada subclase debe proporcionar su propia implementación de estos métodos.
Al definir métodos abstractos en CrudServicio<T>, se asegura que todas las clases derivadas implementan la lógica específica de la entidad que manejan, incluyendo validaciones y reglas de negocio particulares. Esta estructura permite que la clase base maneje las operaciones comunes, mientras que las subclases definen el comportamiento específico, lo que proporciona flexibilidad y extensibilidad en el diseño. La utilización de parámetros Object... datos permite una mayor flexibilidad en la implementación, dado que diferentes entidades pueden requerir distintos conjuntos de datos para sus operaciones.
En resumen, CrudServicio<T> facilita la reutilización del código común y permite la personalización del comportamiento para cada entidad, reduciendo la duplicación y simplificando la extensión del código en el sistema.

¿Por qué usar una clase abstracta en lugar de una interfaz? Elegimos una clase abstracta en lugar de una interfaz para definir la base común de nuestros servicios CRUD principalmente por la necesidad de constructores.
Las clases abstractas pueden incluir tanto métodos abstractos (sin implementación) como métodos con implementación, lo que facilita la definición de funcionalidades compartidas. Aunque las interfaces también pueden tener métodos con implementación desde Java 8 (usando default y static), no pueden tener constructores.
La capacidad de las clases abstractas para tener constructores es crucial para inicializar el repositorio y la clase de entidad en nuestros servicios CRUD. Esto permite una configuración adecuada de los recursos necesarios para que los servicios funcionen correctamente. En

contraste, las interfaces no pueden proporcionar constructores, lo que limita su capacidad para gestionar la inicialización de recursos.
Y aunque las clases en Java solo pueden heredar de una clase abstracta, mientras que pueden implementar múltiples interfaces, para proporcionar una base común con inicialización y funcionalidad compartida, una clase abstracta es más adecuada.

6.2. Implementación de Servicios CRUD con Validaciones

En la capa de negocio de la aplicación, el método abstracto validarEInsertar en la clase CrudServicio se encarga de validar los parámetros al insertar un nuevo elemento. Cada servicio derivado debe implementar este método para realizar validaciones específicas sobre el estado de la instancia del modelo. Las validaciones pueden incluir:

1. No Nulidad: Asegurarse de que los valores no sean nulos.

2. Longitud: Verificar que las cadenas de texto, como nombres, no superen una longitud máxima (por ejemplo, 50 caracteres).

3. Formato Específico: Validar formatos específicos, como el formato del DNI o el ISBN de un libro. Para ello se usan expresiones regulares definidas en la clase Validacion.java.

4. Fechas: Confirmar que las fechas sean adecuadas, como asegurar que la fecha de préstamo no sea anterior a la fecha actual, y que la fecha de devolución no sea posterior a la fecha actual ni anterior a la fecha de préstamo.

Estas validaciones se realizan en el constructor de la entidad. Si algún parámetro no cumple con las reglas establecidas, se lanza una excepción de tipo IllegalArgumentException con un mensaje descriptivo del error y la imposibilidad de realizar la operación.
Los errores encontrados durante el proceso de validación se almacenan en una lista (ArrayList) y se concatenan en un solo mensaje utilizando String.join, separando cada error con un salto de línea para una mejor legibilidad.

En el método validarYModificar, se realiza la modificación de un elemento existente. Los datos proporcionados, junto con el objeto a modificar, se utilizan para establecer los nuevos valores mediante setters. Estos setters también contienen validaciones similares a las del método validarEInsertar. En caso de errores durante estas validaciones, se lanzan excepciones con mensajes descriptivos.
Se utiliza una variable auxiliar para realizar la modificación. ¿Por qué? Esto debido a que el objeto se pasa por referencia y está vinculado al mismo objeto que se muestra en la tabla al usuario. Si se modifica directamente el objeto referenciado por la tabla, se podrían realizar cambios parciales en los datos sin pasar todas las validaciones, lo que podría llevar a inconsistencias. Por ejemplo, si un parámetro es válido y se establece con el setter, se modificaría el objeto de la tabla sin haber validado completamente todos los parámetros. Al

usar una variable auxiliar, se asegura que todos los datos sean validados correctamente antes de actualizar el objeto original. Una vez que los datos han sido validados en la variable auxiliar, se actualiza el objeto original, garantizando que todas las validaciones se hayan cumplido y que las modificaciones se reflejen correctamente en la vista, siempre y cuando se actualice la tabla después de la modificación.

En el método validarYBorrar, se verifica que el objeto a eliminar no esté asociado a ningún registro activo en el sistema. Si el objeto está asociado a otros registros activos, no se permite su eliminación. En casos específicos, como las copias de libros, se ofrece la opción de marcar la copia como "pérdida" en lugar de eliminarla. Esta opción es viable debido a las relaciones bidireccionales entre los objetos; por ejemplo, una copia de un libro puede conocer los préstamos asociados y viceversa.

6.3. Métodos Específicos de Servicios

Dentro de los servicios, además de las operaciones básicas, hay métodos específicos para cada entidad que gestionan funcionalidades particulares:

Servicio de Copias de Libros:

- *verificarReferencias*: Este método recibe una lista de libros y verifica si alguno de ellos tiene copias de referencia asociadas. Si no se encuentran copias de referencia para algún libro, el sistema informa al usuario a través de una alerta emergente, detallando los libros sin copias de referencia, y además los libros que directamente no tengan copias registradas.

- *modificarEstado*: Este método recibe una copia y un nuevo estado para esa copia. Tiene una validación especial: si el estado actual de la copia es "prestada" y se intenta cambiar a "pérdida", el sistema no lo permite. Primero, el préstamo asociado a la copia debe ser cerrado antes de que se pueda marcar la copia como pérdida.

Servicio de Libros:

- *verificarCopias*: Este método recibe un libro y muestra en una alerta emergente todas las copias asociadas a ese libro. Esto elimina la necesidad de acceder al módulo de gestión de copias para consultar las copias filtradas por el libro en cuestión.

Servicio de Miembros:

- *agregarPrestamo*: Recibe un miembro y un préstamo, y agrega el préstamo a la lista de préstamos asociados al miembro. Este método se utiliza para actualizar la lista de préstamos en tiempo real y validar si se puede registrar un nuevo préstamo, dado que un miembro no puede tener más de cinco préstamos en curso. Si se alcanza este límite, se lanza un error.

- *cambiarClave*: Este método permite al usuario cambiar su contraseña. Recibe la contraseña actual, la nueva contraseña y una confirmación de la nueva contraseña. Se valida que la contraseña actual ingresada coincida con la contraseña cifrada del usuario usando el método de validación de Spring Security Crypto. También se verifica que la contraseña actual cumpla con requisitos específicos: al menos 6 caracteres, una letra mayúscula, una letra minúscula, un dígito numérico y un carácter especial. Finalmente, se asegura que la confirmación de la nueva contraseña coincida con la nueva contraseña ingresada.

- *validarCredenciales*: Valida las credenciales de un miembro durante el inicio de sesión. Primero, busca el miembro con el DNI proporcionado, luego valida la contraseña ingresada contra la contraseña cifrada almacenada en la base de datos utilizando el método de validación de Spring Security Crypto.

Nota: Cabe destacar que las contraseñas almacenadas en la base de datos están cifradas. Esto se debe a que, al crear un nuevo miembro de la biblioteca o al cambiar la contraseña existente, se utiliza el método encode de la librería Spring Security Crypto para cifrar la contraseña antes de guardarla en la base de datos. Este proceso asegura que las contraseñas no se almacenan en texto claro, proporcionando una capa adicional de seguridad.

Servicio de Préstamos:

- *cerrarPrestamo*: Recibe un préstamo como parámetro y valida si la fecha de devolución es nula. Si es así, significa que el préstamo aún está abierto y se cierra automáticamente con la fecha actual. Este método facilita el cierre rápido de préstamos desde el módulo de gestión de préstamos sin necesidad de acceder al formulario de cierre manualmente.

- *validarPrestamo*: Recibe un préstamo y válida si se puede registrar en la base de datos. Verifica que el miembro no tenga más de cinco préstamos en curso, que ninguno de los préstamos actuales esté vencido, y que la copia del libro no esté prestada, perdida o sea una copia de referencia. Si alguna de estas condiciones no se cumple, el préstamo no se realiza.

6.4. Roles y Permisos

En el sistema, existen dos roles principales: usuario normal y bibliotecario.

- Bibliotecario: Tiene acceso completo para gestionar préstamos, libros, copias de libros, miembros de la biblioteca, racks, editoriales, categorías, idiomas y autores. Además, puede consultar los registros de auditoría para monitorear los cambios realizados en el sistema. También tiene la capacidad de agregar, modificar y eliminar usuarios, aunque no puede modificar ni eliminar a otros bibliotecarios. La modificación o eliminación de

bibliotecarios debe ser realizada por un administrador desde la base de datos para garantizar la integridad de los datos.

- Usuario Normal: Puede visualizar sus propios préstamos, filtrarlos por fecha de préstamo, fecha de devolución, cifra de multa, y todas las copias que ha retirado históricamente. También puede consultar los libros disponibles en la biblioteca. Para retirar un libro, el usuario debe solicitarlo al bibliotecario, quien verificará la disponibilidad de copias y las validaciones correspondientes antes de registrar el préstamo. En el caso de libros electrónicos o audiolibros, el bibliotecario proporcionará una licencia y un enlace a la plataforma donde esté disponible.

Ambos roles tienen la capacidad de modificar su propia contraseña, pero no pueden modificar sus datos personales. El bibliotecario tiene la autoridad para modificar los datos personales de los usuarios, pero los usuarios no pueden modificar sus propios datos personales.

---

6. Funcionamiento del Sistema

6.1. Uso del Sistema de Módulos en Java

Optamos por utilizar el sistema de módulos en Java por cuestiones de organización del código y para tener un mejor control de las dependencias.
Primero, declaramos el módulo 'com.tubiblioteca' en el archivo module-info.java. El nombre del módulo debe ser único dentro del proyecto y coincide con el nombre del paquete raíz del módulo. A continuación, se declaran las dependencias necesarias del módulo utilizando la palabra clave 'requires'. Estas son bibliotecas externas que el módulo necesita para funcionar correctamente, y así poder utilizarlas en sus clases. Para poder adquirirlas, primero las agregamos en el archivo pom.xml, donde gestionamos las dependencias con Maven. Para buscar una dependencia, lo hacemos desde el repositorio de Maven en internet.

Las bibliotecas que hemos requerido son:

- 'javafx.controls', 'javafx.fxml', 'javafx.graphics': Módulos de JavaFX necesarios para construir interfaces gráficas.

- 'eclipselink', 'jakarta.persistence': Para la persistencia y gestión de entidades JPA (Java Persistence API).

- 'org.slf4j': Biblioteca para el manejo de registros y logging.

- 'org.controlsfx.controls': Extensiones para JavaFX, como controles adicionales.

- 'java.prefs': Para la gestión de preferencias del sistema.

- ‘spring.security.crypto’: Para la criptografía y seguridad proporcionada por Spring Security.

Luego, abrimos los paquetes con la palabra clave ‘opens’, lo que indica que el módulo permite el acceso de otros módulos a través de la reflexión en tiempo de ejecución. En este caso, se abren los paquetes ‘com.tubiblioteca.controller’ y otros subpaquetes específicos de las ABMs para el módulo ‘javafx.fxml’, que es necesario para cargar los archivos FXML en JavaFX.
También se abren los paquetes ‘eclipselink’ y ‘javafx.base’ dentro del modelo, y la razón es la siguiente:

- ‘eclipselink’: Es una implementación de Jakarta Persistence (JPA) utilizada para mapear objetos Java a tablas en una base de datos relacional y viceversa. EclipseLink necesita acceso reflexivo a las clases del modelo para crear instancias de estas al leer datos de la base de datos y para acceder a los campos de estas clases al persistir y recuperar datos.

- ‘javafx.base’: Este es un módulo central que incluye funcionalidades como el manejo de propiedades y el binding (enlace de datos). En JavaFX, es común vincular (bind) las propiedades de la interfaz de usuario a las propiedades del modelo. Para hacer esto, JavaFX necesita acceder y observar los cambios en los campos y métodos de las clases del modelo. Cuando los datos en el modelo cambian, JavaFX usa reflexión para actualizar la interfaz de usuario en consecuencia. Por ejemplo, si una propiedad nombre de una entidad Miembro cambia, la interfaz vinculada a esa propiedad también debe reflejar el cambio.

Finalmente, se exporta el paquete ‘com.tubiblioteca’, lo que significa que las clases públicas en este paquete pueden ser utilizadas por otros módulos que requieran ‘com.tubiblioteca’.

6.2. Arranque del Sistema

El arranque de la aplicación se gestiona a través de la clase App.java. En esta clase, el método main(String[] args) sirve como el punto de entrada principal de la aplicación Java. Este método invoca launch(args), que es el encargado de iniciar la aplicación JavaFX.

6.2.1. Inicialización de la Persistencia

La función init() se ejecuta antes de que se inicie la interfaz gráfica de la aplicación. En esta etapa, se realiza la configuración necesaria para la persistencia de datos:

1. Inicialización del EntityManagerFactory: Se crea utilizando la unidad de persistencia 'TuBibliotecaPU', la cual está definida en el archivo persistence.xml.

2. Configuración del Repositorio: Se inicializa el repositorio con el EntityManagerFactory y se establece en la configuración de la aplicación a través de la clase AppConfig.

3. Manejo de Errores: Si se produce un error durante la inicialización del EntityManagerFactory, se registra el error utilizando SLF4J y se cierra la aplicación mediante Platform.exit().

6.2.2. Inicio de la Interfaz Gráfica

El método start(Stage stage) se ejecuta cuando la aplicación comienza. Recibe un objeto Stage, que representa la ventana principal de la aplicación. Aquí se realiza lo siguiente:

1. Configuración del Stage Principal: Se establece el Stage principal en la clase StageManager, que es responsable de gestionar las diferentes pantallas de la aplicación.

2. Cambio de Escena: Se cambia la escena actual a la vista de Login, la primera pantalla que verá el usuario.

6.2.3. Finalización de la Aplicación

El método stop() se llama cuando la aplicación está a punto de cerrarse, permitiendo realizar cualquier tarea de limpieza necesaria.

6.2.4. Configuración Global con 'AppConfig'

La clase AppConfig está diseñada para gestionar una instancia estática de la clase Repositorio, lo que permite que esta instancia sea accesible globalmente en toda la aplicación. Esta implementación facilita el acceso al repositorio sin la necesidad de crear múltiples instancias de AppConfig.
¿Por qué utilizar un atributo estático? Al declarar el atributo repositorio como estático, este se convierte en una propiedad de la clase AppConfig en lugar de ser una propiedad de una instancia específica. Esto permite a cualquier clase en la aplicación acceder al repositorio a través de AppConfig.getRepositorio() sin necesidad de instanciar AppConfig.
Si el atributo no fuera estático, se tendría que crear una instancia de AppConfig en cada lugar donde se necesite acceder al repositorio, lo que sería ineficiente, dado que el repositorio se utiliza en todos los controladores del sistema para instanciar los servicios.

6.3. Gestión de Pantallas y Modales en la Aplicación

La clase 'StageManager' es responsable de la gestión de pantallas y modales en nuestra aplicación. A continuación, se detallan los métodos más importantes de esta clase:

- *'cambiarEscena'*: Este método se encarga de cambiar la escena del Stage principal. En la primera llamada, se realizan configuraciones iniciales que incluyen:

    - Establecer el título del Stage.

    - Configurar el tamaño del Stage según la escena.
    - Centrar el Stage en la pantalla.
    - Deshabilitar la redimensión del Stage.
    - Configurar el ícono del Stage.

- *'abrirModal'*, *'mostrarModal'*, *'cerrarModal'*: Estos métodos son utilizados para la gestión de modales en la aplicación. Son fundamentales en formularios y selectores de búsqueda, proporcionando una interfaz emergente que permite al usuario interactuar con la aplicación sin modificar la vista principal.

- *'cargarVista'*: Este método se utiliza cuando se necesita actualizar el centro del BorderPane en la vista principal. Dado que el método setCenter requiere un nodo raíz de la vista, cargarVista facilita la carga de este nodo cuando se requiere modificar el contenido central. También se usa para cambiar contraseñas, debido a que es un modal y el método abrirModal recibe el nodo raíz. Y aunque este modal no necesita cargar el nodo raíz antes de ser mostrado, se decidió utilizar cargarVista para mantener la coherencia en el código y evitar la creación de un método adicional solo para cambiar contraseñas.

- *'cargarVistaConControlador'*: Este método es crucial para cargar modales que incluyen formularios y selectores de búsqueda, así como para cargar el controlador del menú. Permite la carga del archivo FXML y la inicialización de los controles visuales y variables del controlador, lo que es necesario para la interacción con ellos. Por ejemplo:

    - Formularios de Modificación: Se pasa el objeto a modificar, y los campos del formulario se completan automáticamente con los datos del objeto.

    - Selectores de Búsqueda: Se utilizan para filtrar elementos en un ComboBox. Se crea un modal con una tabla filtrable de elementos, permitiendo seleccionar un elemento que se envía automáticamente al ComboBox.

    - Menú: Se carga previamente para permitir que el controlador de la vista principal (un BorderPane) envíe este BorderPane al menú. Esto permite que el menú cambie el contenido central del BorderPane en función de las selecciones del usuario.

    - *'setTitulo'*: Se usa para establecer el título de la vista principal. Dado que la vista principal es un BorderPane que incluye dos archivos FXML diferentes (el menú y el contenido central ABM), el título se establece para el contenido central que se está mostrando en ese momento, no para el menú.

Para manejar las vistas y sus controladores, usamos un Pair, debido a que necesitamos guardar dos elementos: el nodo raíz de la vista (Parent) y el controlador asociado (T). De este modo,

podemos cargar y gestionar las vistas de manera flexible y reutilizable de forma genérica con distintos controladores.
Para gestionar los modales, utilizamos un Map, debido a la necesidad de identificar y manejar cada modal de manera única. En este mapa, usamos el título del modal como clave (String) y el Stage del modal como valor. Esto nos ayuda a identificar y manejar cada ventana modal de manera única, facilitando tareas como eliminar un modal cuando se cierra, basándonos en su título.

Nota: Los métodos estáticos en StageManager se utilizan para proporcionar acceso global y simplificar la gestión de vistas y modales. Esto evita la necesidad de instanciar StageManager en cada lugar donde se requiera, asegurando una gestión centralizada y coherente de las pantallas en toda la aplicación.

6.3.1. Ubicación de Archivos FXML

Se utiliza un enum llamado Vistas.java para definir los títulos y ubicaciones de los archivos FXML.

- Cada elemento del enum tiene un constructor privado que establece el título y la ruta del archivo FXML.

- La ruta de los archivos FXML se obtiene mediante App.class.getResource(rutaFxml). Esta ruta es relativa al directorio raíz de los recursos en el proyecto. El método getResource busca el archivo especificado por rutaFxml en el classpath de la aplicación, usando la raíz de la clase App como referencia.

- En la ruta FXML, se incluye el nombre del archivo FXML. Si se trata de una vista ABM, la ruta comienza con ABMNombre/, seguido del tipo de vista como Lista, Formulario, Selector, etc.

6.4. Uso de Controladores

Dentro de los controladores, se definen los controles visuales correspondientes al archivo FXML asociado. Cada carpeta ABM contiene dos archivos FXML: uno para la lista y otro para el formulario.
En ambos casos, se utiliza ObservableList de FXCollections para almacenar la lista de registros que se consultan en la base de datos a través del servicio. Estas listas permiten rellenar tablas o comboboxes de filtros, y son esenciales tanto para crear como para modificar elementos. Se elige ObservableList porque permite que cualquier cambio en la lista se refleje automáticamente en los controles de la interfaz de usuario. Además, su fácil vinculación con componentes como tablas y comboboxes facilita la manipulación de datos.

También se definen los loggers y los servicios que se utilizarán para realizar las consultas necesarias, así como para agregar, modificar y eliminar registros. Los servicios, junto con el modelo, se encargan de todas las validaciones requeridas.
Dentro de la vista de lista, se utiliza un Pair para almacenar el nodo raíz del formulario que se va a abrir, junto con su controlador. Esto permite interactuar con el controlador del formulario, enviarle un objeto seleccionado en la tabla para su modificación, y autocompletar los campos del formulario con los datos del objeto. Una vez modificado el objeto, o tras la adición de nuevos elementos, estos se devuelven a la tabla y se aplican los filtros correspondientes. Para modificar, solo se permite la edición de un elemento a la vez. Al hacer clic en el botón "Guardar", si todo está correcto, el modal se cierra automáticamente. Sin embargo, al agregar nuevos registros, el modal permanece abierto hasta que el usuario decida cerrarlo manualmente, permitiendo la adición de múltiples registros en una sola sesión. Para eliminar un registro, el usuario debe seleccionar un elemento en la tabla y luego proceder a la eliminación.
Todos los controladores incluyen un método initialize para inicializar los controles, como tablas o filtros, así como los servicios que se van a utilizar.
En el caso de las ABM de miembro, libro y copia, se incluye una vista adicional: un selector de búsqueda. Este selector es simplemente una tabla con filtros correspondientes, pero sin los botones de acción que se muestran en la lista principal de estos registros. El objetivo es permitir al usuario buscar de manera más detallada un miembro, libro o copia. Este modal se muestra al hacer clic en los botones situados junto a los comboboxes que muestran miembros, libros o copias (por ejemplo, en la ABM de préstamos para buscar un miembro o una copia, o en la ABM de copias para buscar un libro en particular). Estos selectores son gestionados por un helper llamado Selector.java, que se encarga de abrir el selector, pasarle el objeto seleccionado actualmente en el combobox para que se seleccione automáticamente en la lista, y luego devolver el objeto seleccionado al combobox para que se realice la selección correspondiente.
Además, hay controladores específicos, como el del login, que se explicará en la siguiente sección, o el de la vista principal, que es un BorderPane. Este controlador se encarga de configurar el menú y pasarle el BorderPane para que el controlador del menú, en caso de que el centro del BorderPane esté vacío (es decir, la primera vez que se ejecuta la aplicación), muestre la lista de préstamos por defecto.
En el controlador del menú se definen los diferentes botones que corresponden a las distintas ABMs disponibles en el sistema, junto con un HBox que actúa como un botón para el perfil del usuario. Este HBox contiene dos labels que muestran el nombre y tipo de usuario que ha iniciado sesión, además de un botón para cerrar sesión. Se lo trata como un “botón”, ya que al presionarlo se muestra un modal que permite al usuario cambiar su contraseña.
Se agregan listeners a los botones para detectar clics y disparar eventos que cambian el centro del BorderPane. También se emplean transiciones para cambiar de un contenido a otro, utilizando FadeTransition de la librería javafx.animation. Estas transiciones de desvanecimiento, tanto de entrada como de salida, tienen una duración de 0.3 segundos, lo que mejora la experiencia visual al cambiar entre vistas.

6.5. Proceso de Inicio de Sesión

En cuanto al proceso de inicio de sesión, lo primero que se realiza es la carga automática de las credenciales si están guardadas. ¿Cómo se almacenan estas credenciales? Si el checkbox "Recordar" está seleccionado, las credenciales se guardan utilizando la clase Preferences, que pertenece al paquete java.util.prefs de Java. Esta clase ofrece una manera de almacenar y recuperar configuraciones de usuario de manera persistente en un almacenamiento específico del sistema, como el registro en Windows, archivos en Linux, o el sistema de preferencias en macOS.
Para almacenar las credenciales de inicio de sesión, se crea o se accede a un "nodo de preferencias" específico para la clase LoginControlador. Los nodos de preferencias son como directorios o contenedores que almacenan pares clave-valor. En este caso, se almacenan dos pares clave-valor que corresponden al DNI y la contraseña del usuario.
Una vez que el usuario presiona "Entrar", se realizan las validaciones necesarias dentro del servicio correspondiente. Si todas las validaciones son satisfactorias, se procede con el inicio de sesión del usuario.
La información del usuario que ha iniciado sesión se almacena en la clase SesionManager.java, la cual contiene un atributo estático de tipo Miembro, que debe ser compartido globalmente entre todos los controladores, permitiendo que la información del usuario esté disponible de manera centralizada en toda la aplicación.
Dentro de SesionManager, se definen varios métodos estáticos clave, como:

- Comprobar si se ha iniciado sesión: Verifica que el miembro sea diferente de nulo.

- Abrir una sesión: Inicia la sesión para el usuario, almacenando la información correspondiente y redirigiendo a la vista principal.

- Cerrar sesión: Finaliza la sesión actual, limpia la información del usuario y redirige al usuario a la pantalla de inicio de sesión.

6.6. Uso de Helpers

Los helpers se utilizan en diversas funciones clave dentro de la aplicación, tales como mostrar alertas, realizar configuraciones específicas en los controles visuales, formatear fechas LocalDate a cadenas de texto más legibles para el usuario, y validar datos mediante expresiones regulares. Estas validaciones incluyen la verificación de un DNI, contraseña, nombre, ISBN específico, así como el cifrado de contraseñas o la validación de una cadena con una contraseña cifrada utilizando la librería Spring Security Crypto.
Es importante destacar que todos estos métodos se declaran como estáticos, ya que son de uso global por todos los controladores y solo dependen de los parámetros que se les pasan. De este modo, no es necesario instanciar estas clases, dado que no dependen de atributos de instancia. En los casos en que sí se utilizan atributos definidos en la clase, como en algunas validaciones, estos atributos también son estáticos, ya que mantienen un valor constante durante la ejecución del sistema y son compartidos entre todos los controladores.

7. Guía del Usuario:

1. **Introducción**

Esta guía está diseñada para ayudar a los usuarios a navegar y utilizar las funciones principales del sistema de manera eficiente.

2. **Roles en el Sistema**

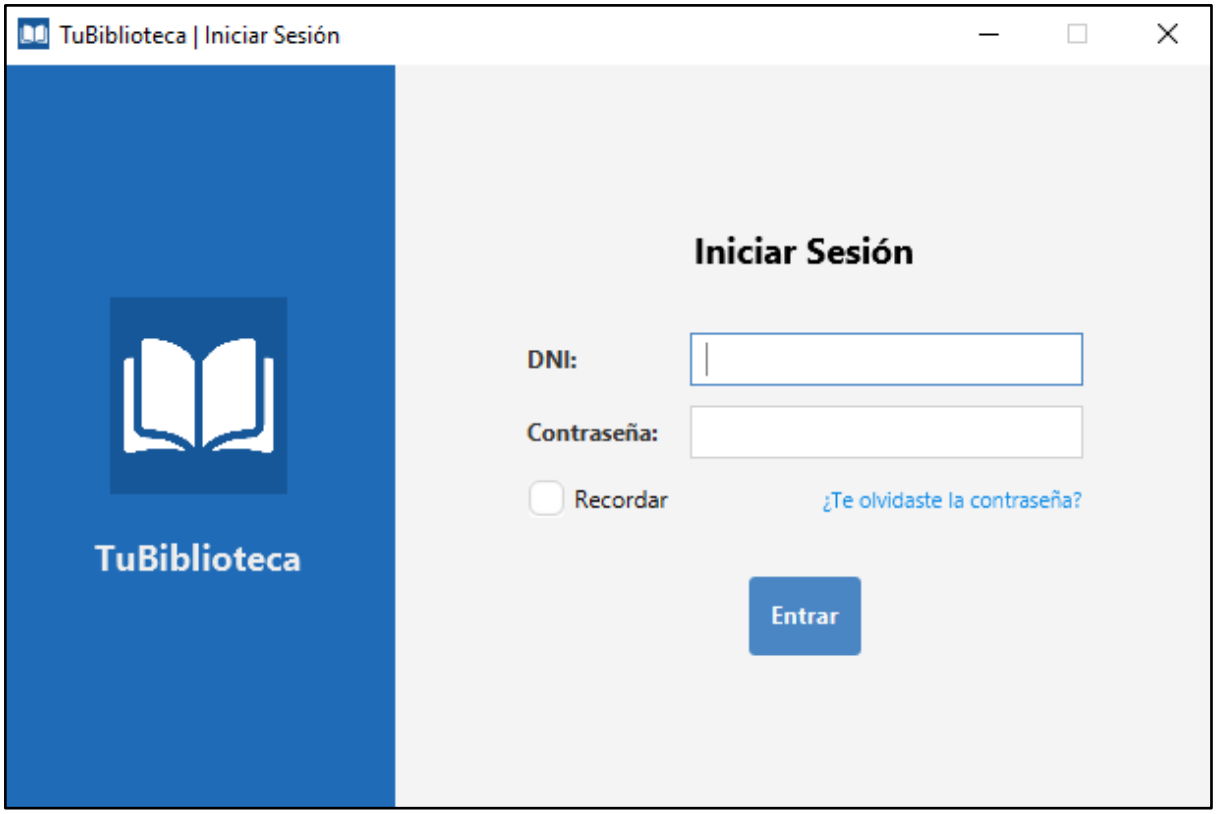
El sistema está diseñado para dos roles principales:

- **Administrador**: Gestiona el catálogo de libros, usuarios, políticas de préstamo y los préstamos
- **Usuario**: consulta su historial de préstamos y visualiza los libros disponibles

3. Acceso al Sistema

Iniciar Sesión

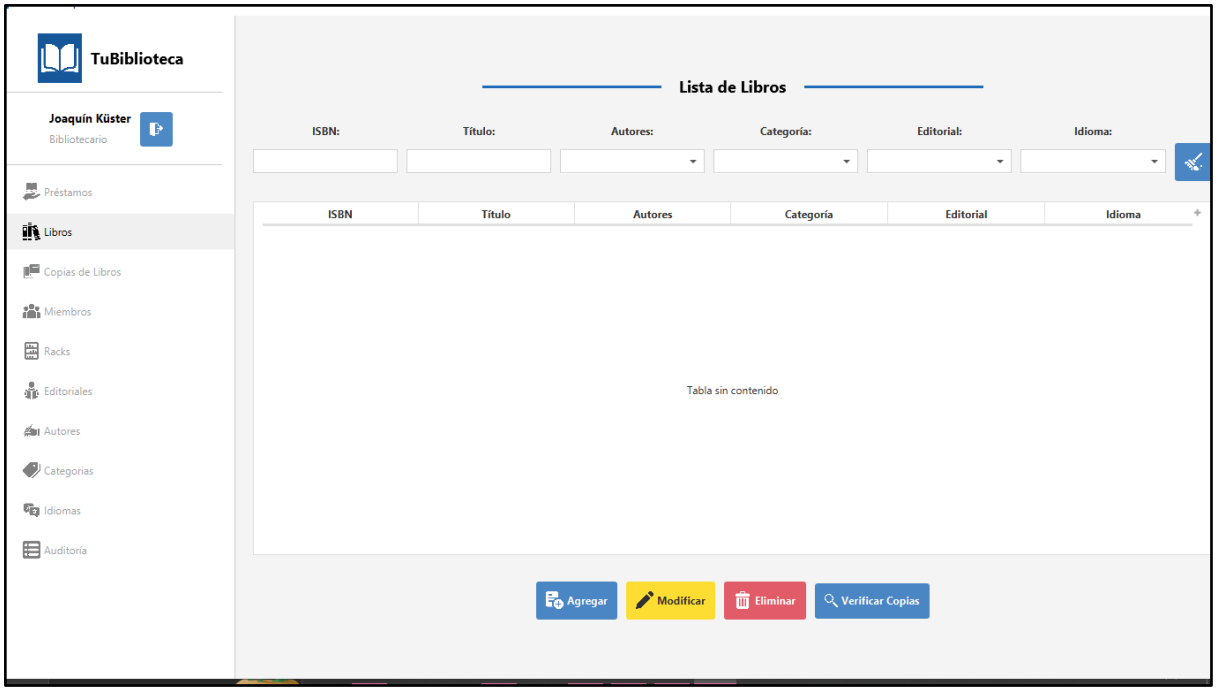
1. **Abrir el Sistema**:
   Acceda al sistema haciendo clic en el ícono de la biblioteca en su escritorio o en la barra de tareas.
2. **Ingreso de Credenciales**:
   - **DNI**: Ingrese su número de DNI en el campo correspondiente.
   - **Contraseña**: Ingrese su contraseña en el campo correspondiente.
3. **Hacer Clic en "Entrar"**:
   Una vez ingresadas sus credenciales, haga clic en el botón **"Entrar"** para acceder al sistema.
   Nota: Si no recuerda su contraseña, haga clic en **"¿Te olvidaste la contraseña?"** para recibir instrucciones sobre cómo restablecerla.



4. Funciones del Administrador

4.1. Gestión de Libros

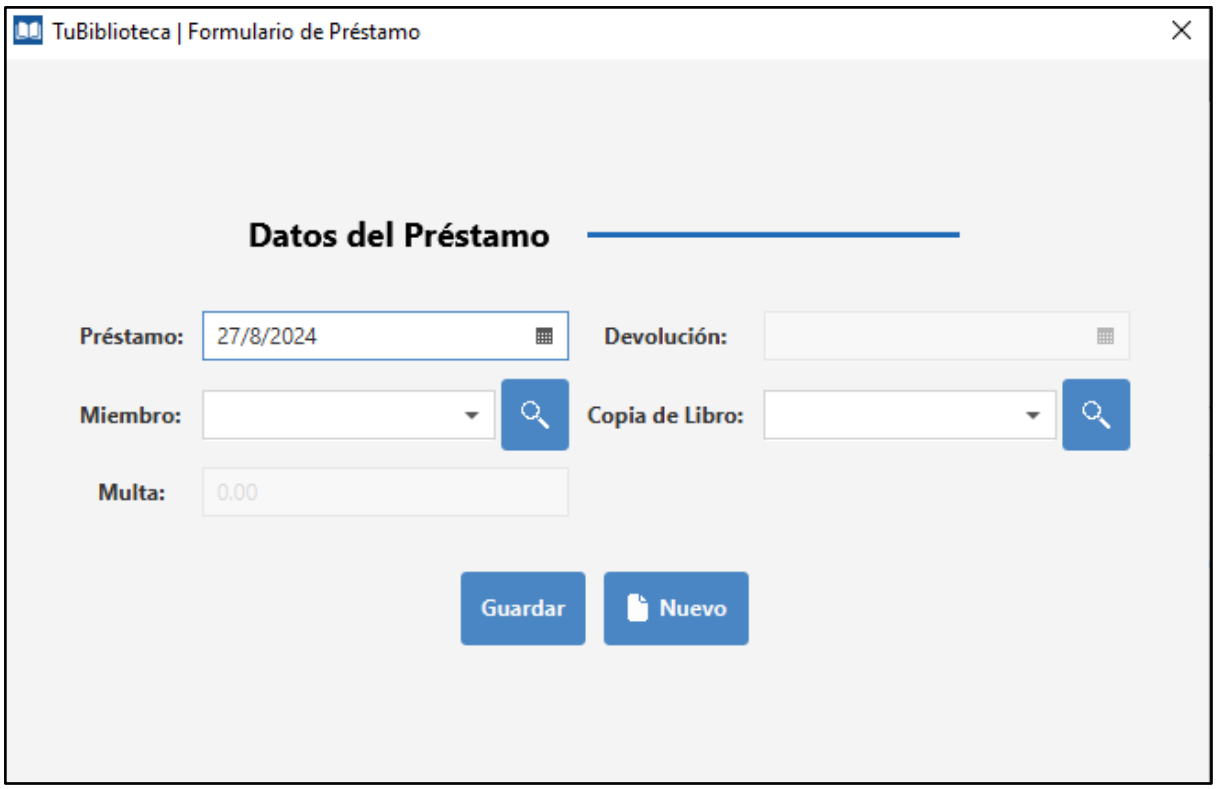
- **Agregar Nuevos Libros**:
  Acceda a la opción **"Libros"** en la barra lateral y seleccione **"Agregar"**. Complete los campos requeridos, como Título, Autor, Editorial, Categoría, ISBN, Idioma, y asigne un Rack.
- **Editar Información de Libros**:
  Seleccione un libro existente, haga clic en **"Modificar"**, realice los cambios necesarios y guarde la información.
- **Eliminar Libro**:
  Seleccione el libro que desea eliminar y confirme la acción haciendo clic en **"Eliminar"**.



4.2. Gestión de Préstamos

- **Registrar Préstamo**:
  Vaya a la sección **"Préstamos"** y seleccione **"Agregar"**. Ingrese los detalles del préstamo, incluyendo Fecha, Miembro, Multa, Fecha de Devolución, y Copia del Libro. Haga clic en **"Guardar"** para registrar el préstamo.

- **Eliminar Préstamo**:
  Seleccione el préstamo que desea eliminar y haga clic en **“Eliminar”**.
- **Modificar Préstamo**:
  Seleccione el préstamo que desea modificar, haga clic en **“Modificar”**, edite los campos necesarios, y luego haga clic en **“Guardar”** para actualizar el préstamo.
- **Confirmar Devolución**:

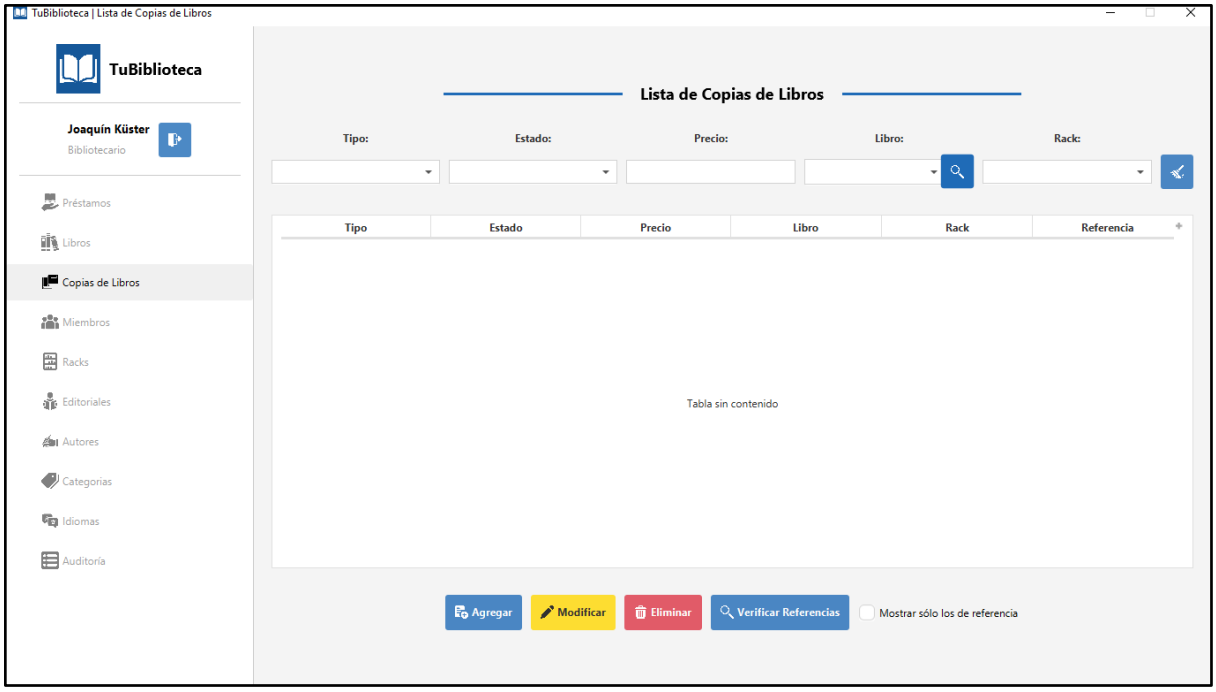


4.3. Gestión de Copias

Registrar Copia: En la sección **"Copias"**, seleccione **"Agregar"**,complete los campos solicitados y seleccione **"Guardar"**

Eliminar Copias : Seleccione la copia que desea eliminar y seleccione la opción "Eliminar"

Verificar copias:seleccione la opción "verificar copias" y visualice las copias de un libro



4.4. Gestión de Editoriales

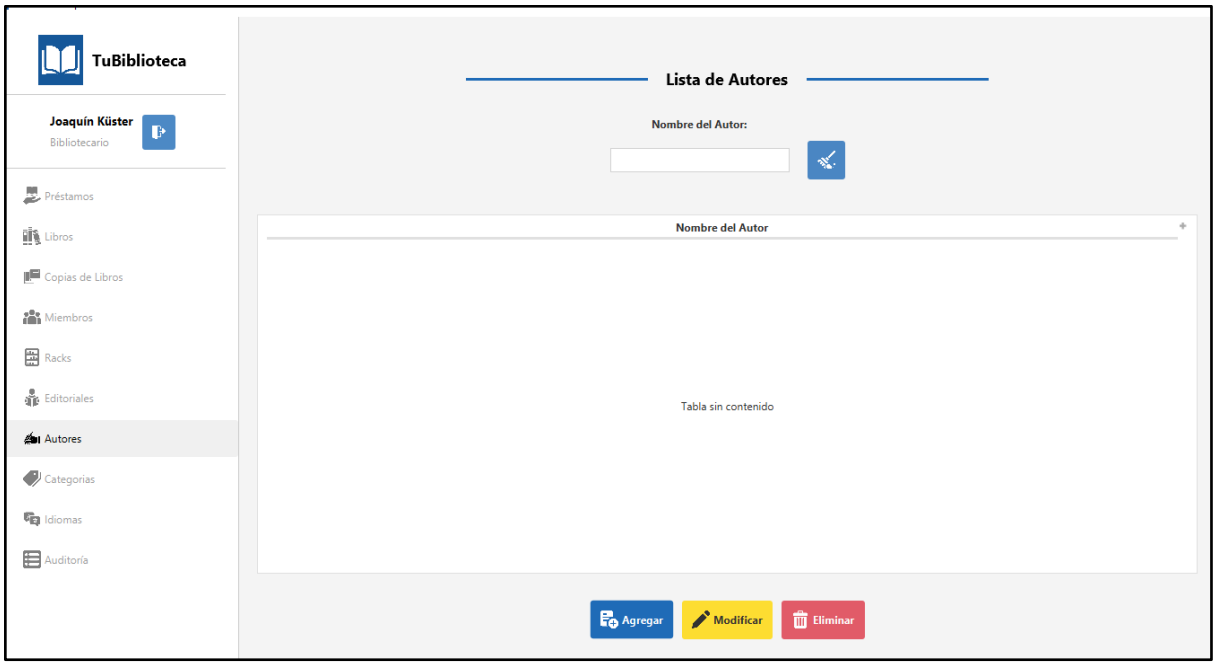
- **Registrar Editorial**:
  En la sección **"Editoriales"**, seleccione **"Agregar"**, ingrese el nombre de la nueva editorial y haga clic en **"Guardar"**.
- **Eliminar Editorial**:
  Seleccione la editorial que desea eliminar y haga clic en **"Eliminar"**.

- **Modificar Editorial**:
  Seleccione la editorial que desea modificar, haga clic en **“Modificar”**, realice los cambios necesarios, y haga clic en **“Guardar”**.

4.5. Gestión de Autores

- **Registrar Autor**:
  En la sección **“Autores”**, seleccione **“Agregar”**, ingrese el nombre del nuevo autor y haga clic en **“Guardar”**.

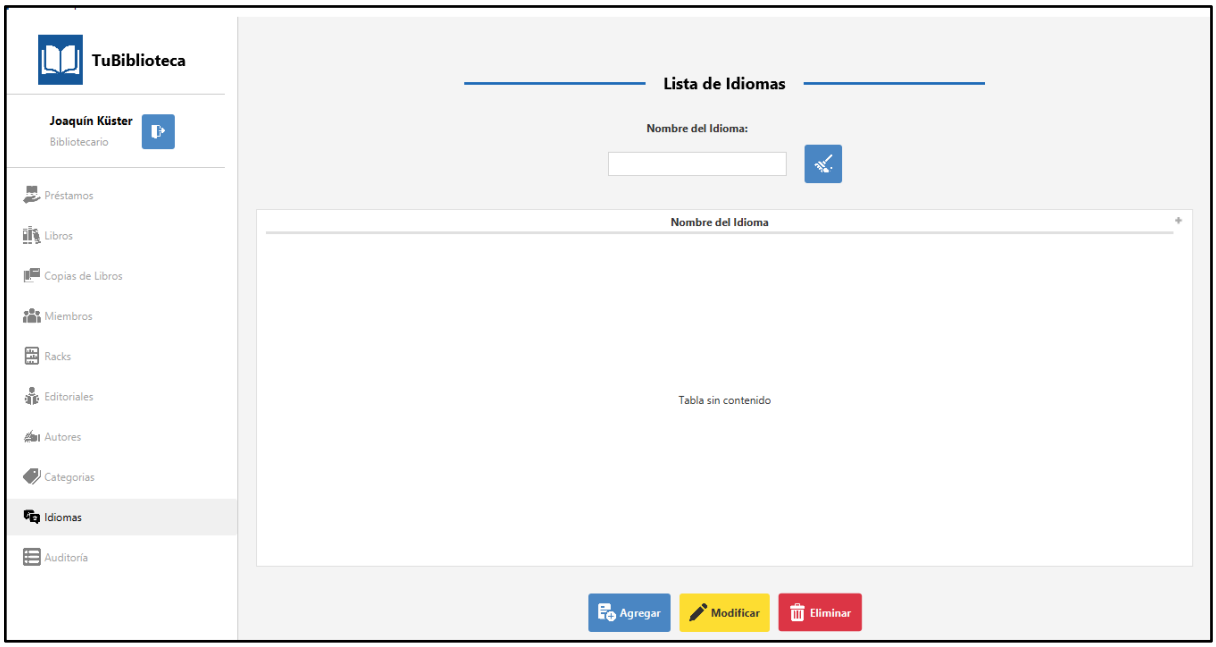
- **Eliminar Autor**:
  Seleccione el autor que desea eliminar y haga clic en **“Eliminar”**.
- **Modificar Autor**:
  Seleccione el autor que desea modificar, haga clic en **“Modificar”**, edite el nombre del autor, y haga clic en **“Guardar”**.

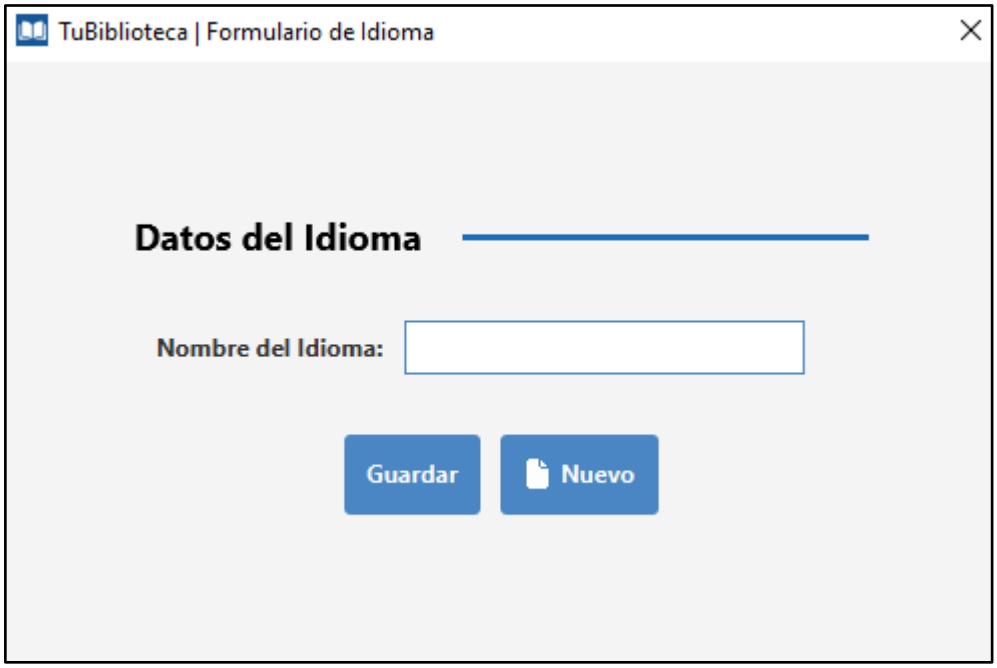


4.6. Gestión de Idiomas

- **Registrar Idioma**:
  En la sección **“Idiomas”**, seleccione **“Agregar”** e ingrese el nombre del nuevo idioma. Haga clic en **“Guardar”** para confirmar.
- **Eliminar Idioma**:
  Seleccione el idioma que desea eliminar y haga clic en **“Eliminar”**.

- **Modificar Idioma**:
  Seleccione el idioma que desea modificar, haga clic en **“Modificar”**, edite el nombre, y haga clic en **“Guardar”**.





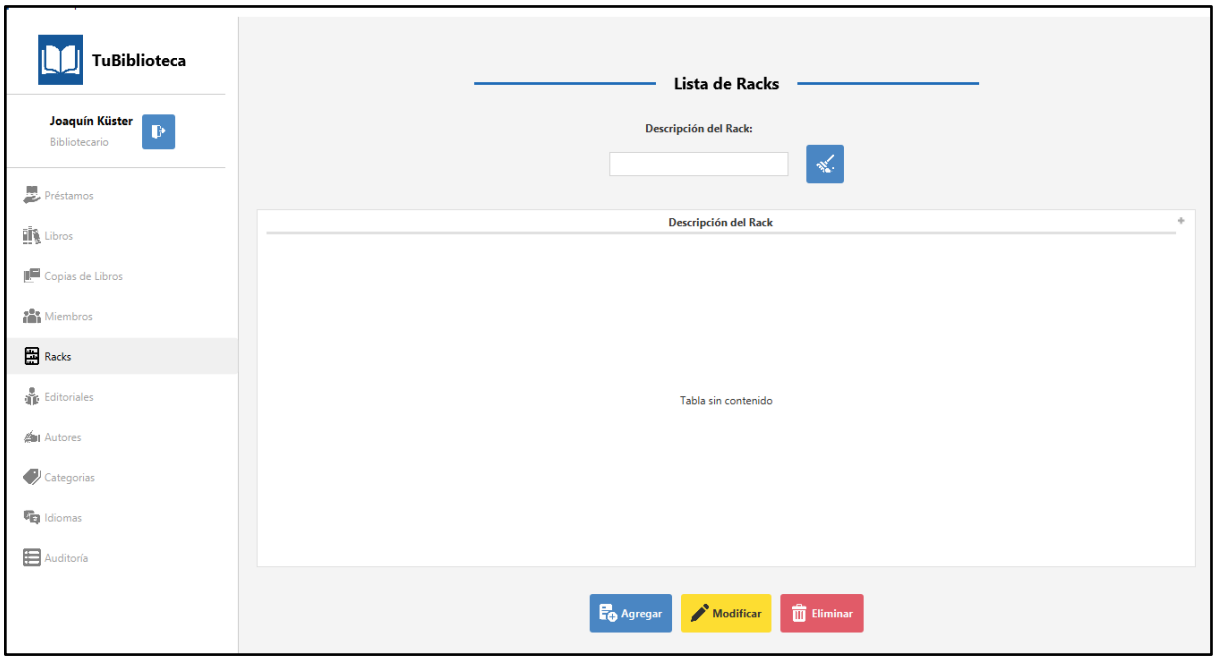
4.7. Gestión de Categorías

- **Registrar Categoría**:
  En la sección **“Categorías”**, seleccione **“Agregar”**, complete el campo con el nombre de la nueva categoría, y haga clic en **“Guardar”**.
- **Eliminar Categoría**:
  Seleccione la categoría que desea eliminar y haga clic en **“Eliminar”**.
- **Modificar Categoría**:
  Seleccione la categoría que desea modificar, haga clic en **“Modificar”**, edite el nombre de la categoría, y haga clic en **“Guardar”**.

4.8. Gestión de Racks

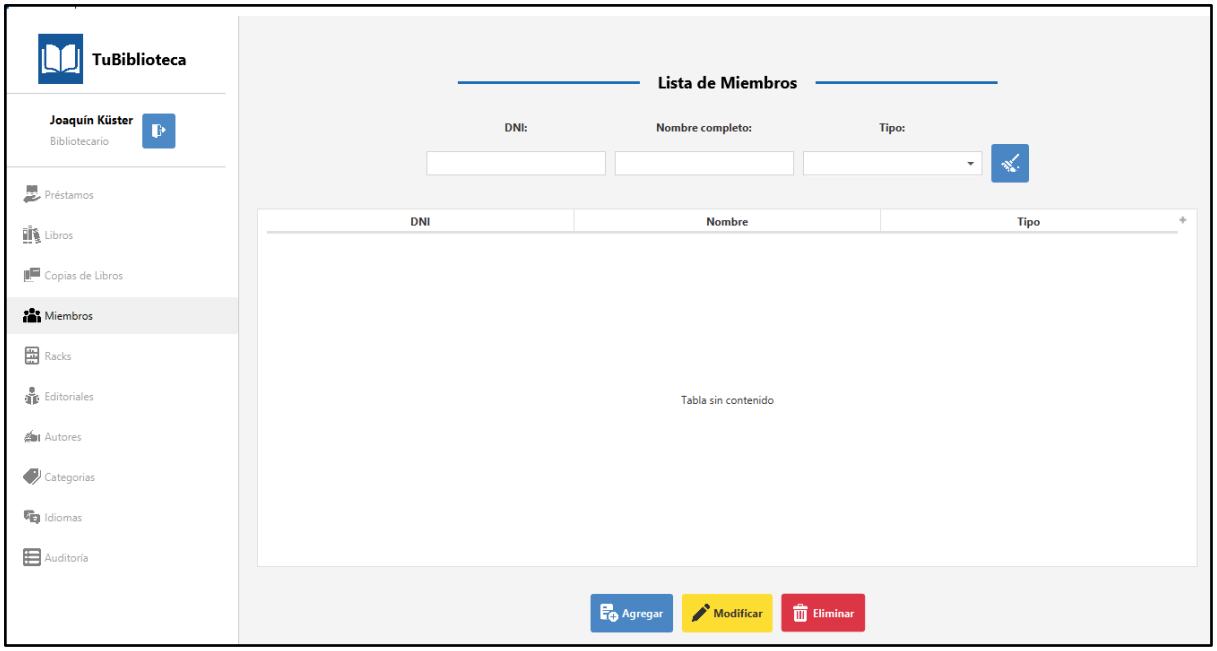
- **Registrar Rack**:
  En la sección **"Racks"**, seleccione **"Agregar"** y complete la descripción del nuevo rack. Haga clic en **"Guardar"** para confirmar.
  Nota: La descripción del rack ayuda a localizar fácilmente dónde se encuentra un libro específico o una copia dentro de la biblioteca.
- **Eliminar Rack**:
  Seleccione el rack que desea eliminar y haga clic en **"Eliminar"**.

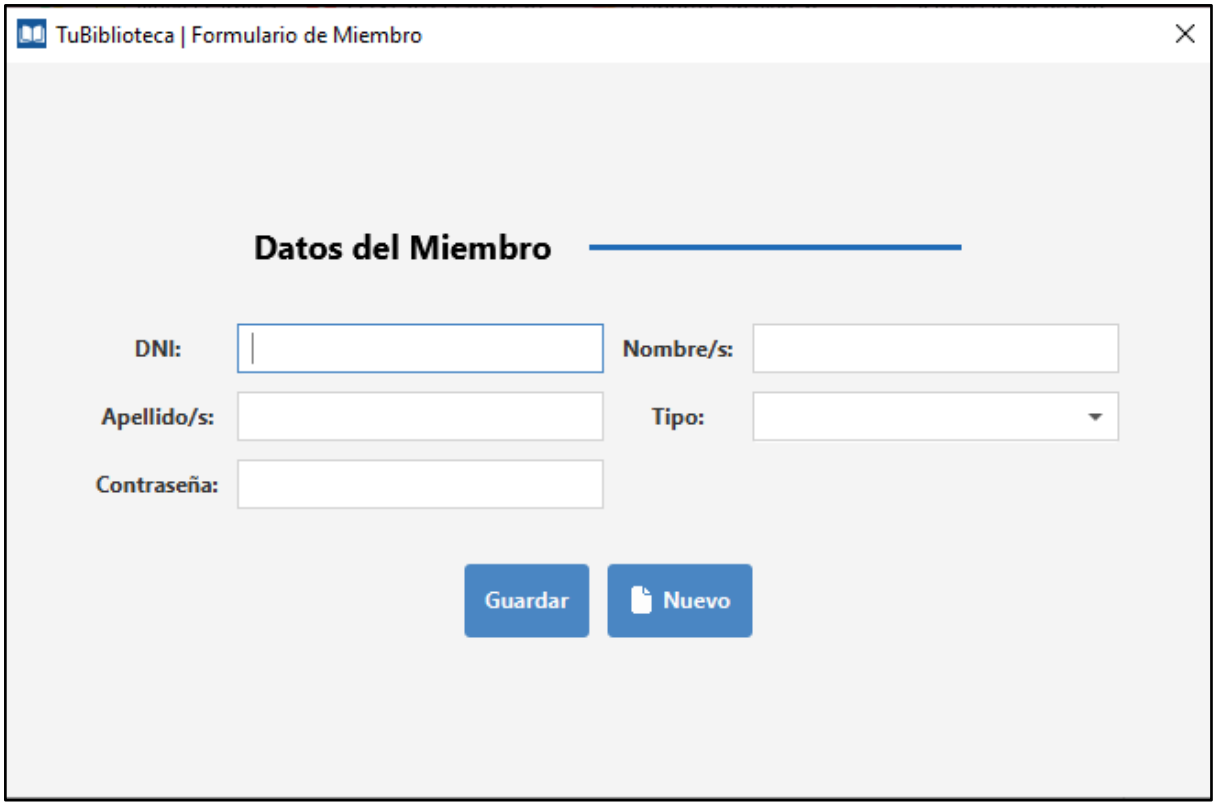
- **Modificar Rack**:
  Seleccione el rack que desea modificar, haga clic en **“Modificar”**, edite la descripción, y haga clic en **“Guardar”**.



4.9. Gestión de Usuarios

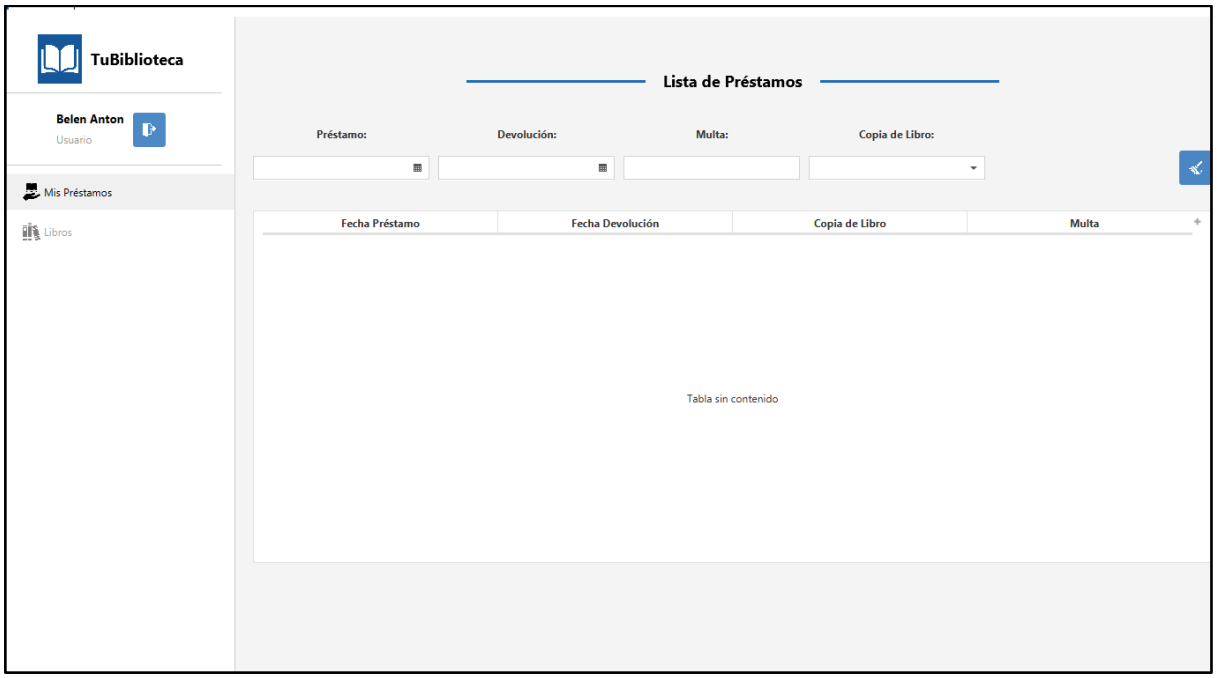
- **Agregar Nuevos Usuarios**:
  Diríjase a **“Miembros”** y seleccione **“Agregar”**. Ingrese los datos requeridos, como Nombre, Apellido, Contraseña, y el Tipo de usuario (Rol). Haga clic en **“Guardar”** para confirmar.
- **Actualizar Información de Usuarios**:
  Seleccione un usuario existente, haga clic en **“Modificar”**, edite la información necesaria, y haga clic en **“Guardar”** para aplicar los cambios.





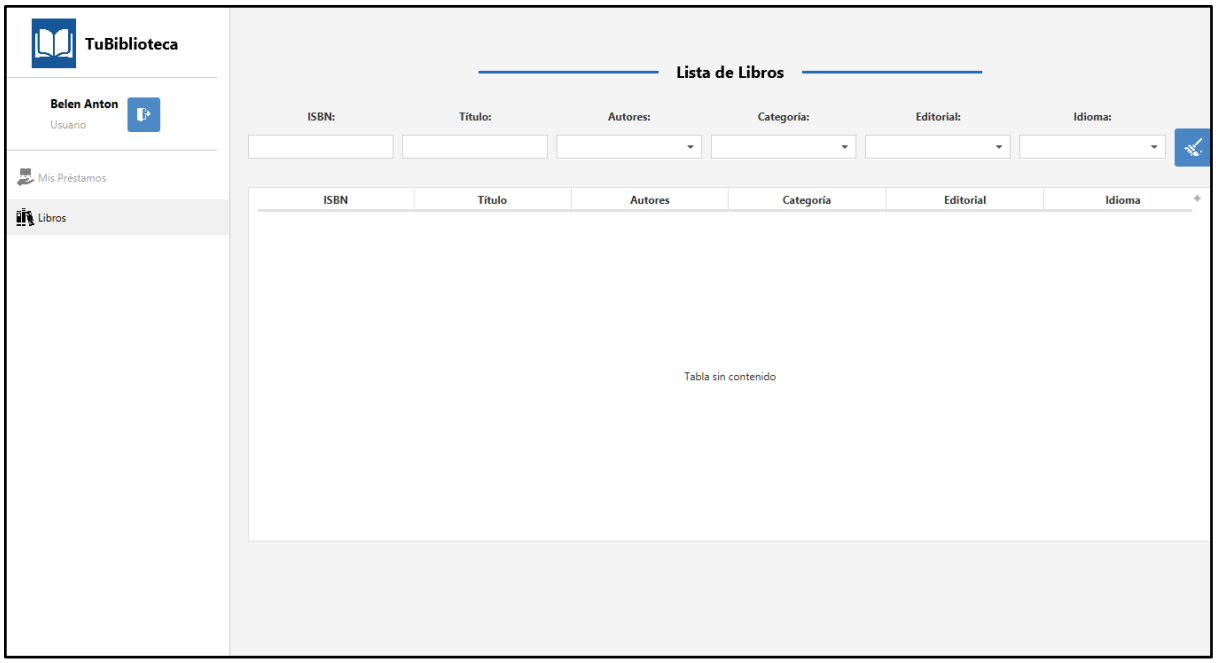
5. Funciones del Usuario

Mis Préstamos: En esta sección se puede visualizar el historial de préstamos obtenidos de libros los cuales se filtran por fecha de préstamos,devolución,multa,copia de libro



Sección Libros: En esta sección se puede visualizar los libros que se encuentran disponibles

Buscar Un Libro: Para realizar la búsqueda de algún libro en particular complete los campos que se encuentran en el margen superior.



8. Auditoría y Registro de Cambios

Este módulo es el encargado de registrar todas las acciones realizadas por el bibliotecario
La auditoría se almacena en una tabla específica en la base de datos, que tiene las siguientes características:

- **Tabla:** auditoria
- **Columnas:**
    1. **ID:** Identificador único para cada registro de auditoría.
    2. **Tabla Afectada:** Nombre de la tabla en la que se realizó la acción.
    3. **Acción Realizada:** Tipo de operación realizada (por ejemplo, inserción, actualización, eliminación).
    4. **Dato Afectado:** Representación textual (Método toString) de la entidad que fue afectada por la acción.
    5. **Fecha y Hora:** Momento exacto en que se realizó la acción.
    6. **DNI del Miembro**:DNI del bibliotecario que realizó la acción

Descripción del Proceso de Auditoría

- **Alcance:** La auditoría está enfocada exclusivamente en las acciones que pueden ser realizadas por un bibliotecario.
- **Limitaciones:** La información almacenada en la auditoría está en formato de cadena de texto (String). Aunque esta decisión simplifica el diseño, limita la capacidad de realizar análisis más detallados. Como mejora futura, se podrían crear tablas específicas para cada tipo de dato registrado en la auditoría.

9.Pruebas:

Durante el desarrollo del sistema, se empleó la tecnología Scenic View como una herramienta de depuración avanzada para analizar y comprender en profundidad las propiedades de los componentes de JavaFX. Esta herramienta facilitó la inspección en tiempo real de la estructura de la escena y sus nodos, permitiendo un análisis detallado de las propiedades aplicables a cada componente.

El análisis específico realizado con Scenic View permitió identificar que, en el caso de los componentes CheckComboBox de ControlsFX, los estilos definidos mediante clases CSS no se aplicaban como se esperaba. Sin embargo, al aplicar estilos directamente utilizando los identificadores (ID) de los componentes, se logró el comportamiento deseado. Esta observación fue crucial para el correcto estilizado del UI, asegurando que los componentes visuales se presentarán conforme a los requerimientos de diseño establecidos.