**UNIVERSIDAD NACIONAL DE INGENIERÍA**

**FACULTAD DE INGENIERÍA INDUSTRIAL Y DE SISTEMAS**



**“Sistema de Gestión de la Biblioteca de la FIIS”**

**Integrantes**

Fernandez Rojas Gary David 20222622B

Tapia Arevalo, Kevin Alexis 20212623F

Cornejo Palomino José Julián 20220353D

**Curso**

Programación Orientada a Objetos

**Docente**

Eric Gustavo Coronel Castillo

**Lima, 10 de junio de 2024**

Dedicatoria

Dedicamos este proyecto gracias a la formación académica por la universidad nacional de ingeniería, por parte de los docentes y el compañerismo obtenido por la facultad de ingeniería industrial de sistemas.

Índice

[Resumen 6](#_Toc168953551)

[Abstract 7](#_Toc168953552)

[Introducción 8](#_Toc168953553)

[Objetivos Generales 8](#_Toc168953554)

[Objetivos Específicos 8](#_Toc168953555)

[Alcances y limites 9](#_Toc168953556)

[Alcances 9](#_Toc168953557)

[Limites 9](#_Toc168953558)

[Requerimientos 9](#_Toc168953559)

[Requerimientos funcionales 9](#_Toc168953560)

[Requerimientos no funcionales 10](#_Toc168953561)

[Actores 11](#_Toc168953562)

[Casos de uso 11](#_Toc168953563)

[Análisis de Requerimientos 12](#_Toc168953564)

[CU01: Registro de un usuario: 12](#_Toc168953565)

[Casos de uso 12](#_Toc168953566)

[Especificación 12](#_Toc168953567)

[Diagrama E-R: 13](#_Toc168953568)

[Análisis de Caja Negra 13](#_Toc168953569)

[Programación del servicio 13](#_Toc168953570)

[Prueba del servicio 14](#_Toc168953571)

[CU02: Registro de préstamos: 16](#_Toc168953572)

[Casos de uso 16](#_Toc168953573)

[Especificación 16](#_Toc168953574)

[Diagrama E-R 17](#_Toc168953575)

[Análisis de la caja negra 17](#_Toc168953576)

[Programación del servicio 17](#_Toc168953577)

[Prueba del servicio 19](#_Toc168953578)

[CU03: Registro de devoluciones: 19](#_Toc168953579)

[Casos de uso 19](#_Toc168953580)

[Especificación 19](#_Toc168953581)

[Diagrama E-R 20](#_Toc168953582)

[Análisis de la caja negra 21](#_Toc168953583)

[Programación del servicio 21](#_Toc168953584)

[Prueba del servicio 22](#_Toc168953585)

[CU04: Agregar falta a un alumno 23](#_Toc168953586)

[Caso de Uso 23](#_Toc168953587)

[Especificación 23](#_Toc168953588)

[Diagrama 24](#_Toc168953589)

[E-R 24](#_Toc168953590)

[Análisis de la caja negra 24](#_Toc168953591)

[Programación del servicio 25](#_Toc168953592)

[Prueba de Servicio 26](#_Toc168953593)

[26](#_Toc168953594)

[CU05: Penalizar a un alumno 26](#_Toc168953595)

[Casos de uso 26](#_Toc168953596)

[Diagrama E-R 27](#_Toc168953597)

[Análisis de la caja negra 28](#_Toc168953598)

[28](#_Toc168953599)

[28](#_Toc168953600)

[28](#_Toc168953601)

[Programación del servicio 28](#_Toc168953602)

[Prueba del servicio 30](#_Toc168953603)

[CU06: Registrar pago penalización 30](#_Toc168953604)

[Casos de uso 30](#_Toc168953605)

[Especificación: 30](#_Toc168953606)

[Diagrama E-R: 32](#_Toc168953607)

[Análisis de Caja Negra: 32](#_Toc168953608)

[Programación del servicio 32](#_Toc168953609)

[Prueba de servicio 34](#_Toc168953610)

# Resumen

El proyecto "Sistema de Gestión de la Biblioteca de la FIIS" tiene como objetivo principal desarrollar una aplicación integral que mejore la organización y el acceso a la colección de recursos digitales de una biblioteca de la FIIS. Una biblioteca es un componente crucial para el desarrollo del conocimiento y la promoción de hábitos de estudio y lectura entre los estudiantes y el personal académico. Por ello, se hace necesario optimizar y automatizar los procesos internos para garantizar un servicio eficiente y accesible a todos los usuarios.

El sistema diseñado busca proporcionar una interfaz amigable y didáctica, que facilite a los usuarios el acceso y uso de los recursos disponibles. Entre las funcionalidades clave del sistema se encuentran la búsqueda avanzada de recursos, la gestión eficiente de préstamos y devoluciones de libros, la administración de usuarios con distintos roles, y la generación de informes detallados sobre el uso de la biblioteca. Estas características permiten una mayor eficiencia y efectividad en la operación diaria de la biblioteca.

La optimización de los procesos de préstamo y devolución es otra de las áreas de enfoque del proyecto. El sistema automatiza estos procesos para reducir el tiempo de espera y minimizar los errores humanos. Al registrar cada préstamo y devolución en la base de datos, el sistema mantiene un historial actualizado de las actividades de los usuarios, lo que permite un seguimiento efectivo y preciso de los recursos prestados.

El sistema de gestión de usuarios permite la creación de perfiles individuales que almacenan información relevante como el historial de préstamos y devoluciones. Además, se proporciona acceso personalizado a los recursos de la biblioteca según el tipo de usuario (estudiantes, bibliotecarios y administradores). Los administradores tienen la capacidad de gestionar el inventario de libros, añadir o actualizar registros, y asignar penalizaciones a los usuarios en caso de retrasos o daños en los ejemplares.

La implementación de un sistema de penalizaciones es esencial para mantener el orden y la disciplina en el uso de los recursos de la biblioteca. El sistema permite agregar faltas a los alumnos por retrasos en la devolución de libros, daños o pérdida de ejemplares, y asignar penalizaciones monetarias en función de la gravedad de la falta. Además, el registro de pagos de penalizaciones permite desbloquear el acceso a los alumnos sancionados, promoviendo un comportamiento responsable entre los usuarios.

Otro aspecto destacado del sistema es la capacidad de generar informes y estadísticas sobre el uso de la biblioteca. Esta funcionalidad proporciona a los administradores datos valiosos para la toma de decisiones informadas, permitiendo identificar patrones de uso, evaluar la efectividad de los recursos disponibles, y planificar mejoras continuas en el servicio. El proyecto ha sido desarrollado utilizando tecnologías modernas como Java para la lógica de negocio y SQL Server para la gestión de la base de datos.

# Abstract

The project "Library Management System for FIIS" aims to develop a comprehensive application that improves the organization and access to the digital resource collection of the FIIS library. A library is a crucial component for the development of knowledge and the promotion of study and reading habits among students and academic staff. Therefore, it is necessary to optimize and automate internal processes to ensure efficient and accessible service for all users.

The designed system seeks to provide a user-friendly and educational interface that facilitates access to and use of the available resources. Key functionalities of the system include advanced resource search, efficient management of book loans and returns, user administration with different roles, and the generation of detailed reports on library usage. These features enable greater efficiency and effectiveness in the daily operations of the library.

Optimizing the processes of lending and returning books is another focus area of the project. The system automates these processes to reduce waiting times and minimize human errors. By recording each loan and return in the database, the system maintains an updated history of user activities, allowing for effective and precise tracking of borrowed resources.

The user management system allows for the creation of individual profiles that store relevant information such as loan and return history. Additionally, personalized access to library resources is provided according to the user type (students, librarians, and administrators). Administrators have the ability to manage the book inventory, add or update records, and assign penalties to users in case of delays or damages to the items.

Implementing a penalty system is essential to maintaining order and discipline in the use of library resources. The system allows for adding infractions to students for late returns, damages, or loss of items, and assigning monetary penalties based on the severity of the infraction. Moreover, recording penalty payments allows for unlocking access for sanctioned students, promoting responsible behavior among users.

Another highlighted aspect of the system is its ability to generate reports and statistics on library usage. This functionality provides administrators with valuable data for informed decision-making, allowing them to identify usage patterns, evaluate the effectiveness of available resources, and plan continuous service improvements. The project has been developed using modern technologies such as Java for business logic and SQL Server for database management.

# Introducción

El proyecto Sistema de Gestión de Biblioteca para la Facultad de Ingeniería Industrial y de Sistemas (FIIS) pretende desarrollar un sistema web que mejore la organización y el acceso a la colección de recursos digitales de la biblioteca. Este sistema busca proporcionar una interfaz amigable y didáctica para que los usuarios puedan acceder y utilizar los recursos de manera eficiente y agradable.

Una biblioteca es un lugar importante para el desarrollo de conocimientos y la buena práctica del estudio y lectura. Por ello, los procesos dentro de ella son más automatizados de modo óptima para todos los usuarios.

## Objetivos Generales

* Crear un sistema web integral que mejore la organización y el acceso a la colección de recursos digitales de la biblioteca.
* Proporcionar una interfaz amigable y didáctica que permita a los usuarios acceder y utilizar los recursos de la biblioteca de manera eficiente y agradable.

## Objetivos Específicos

Implementar funcionalidades de búsqueda avanzada:

* Diseñar y desarrollar un motor de búsqueda que permita a los usuarios localizar recursos digitales de manera rápida y precisa mediante palabras clave, autores, categorías y otros filtros.

Optimizar el proceso de préstamo y devolución:

* Automatizar los procesos de préstamo y devolución de recursos digitales para reducir el tiempo de espera y minimizar errores humanos.
* Implementar un sistema que informe a los usuarios sobre fechas de vencimiento, renovaciones y disponibilidad de recursos.

Desarrollar un sistema de gestión de usuarios:

* Crear perfiles de usuario que almacenen información relevante, como historial de préstamos y devoluciones.
* Proporcionar acceso personalizado a los recursos de la biblioteca según el tipo de usuario (estudiantes, bibliotecarios, administradores).

Mejorar la administración de la biblioteca:

* Proporcionar informes y estadísticas sobre el uso de la biblioteca, permitiendo a los administradores tomar decisiones informadas.

Fomentar el uso de la biblioteca mediante una interfaz intuitiva:

* Diseñar una interfaz de usuario intuitiva y atractiva que motive a los usuarios a utilizar los recursos de la biblioteca con mayor frecuencia.
* Incorporar elementos didácticos y tutoriales para ayudar a los usuarios a familiarizarse con el sistema y maximizar su aprovechamiento.

# Alcances y limites

# Alcances

El proyecto desarrollará un sistema web para la gestión de la biblioteca de la Facultad de Ingeniería Industrial y de Sistemas (FIIS), permitiendo a bibliotecarios y alumnos gestionar libros y préstamos de manera eficiente. Las principales funciones incluyen el registro y actualización de libros, gestión de préstamos y devoluciones, administración de usuarios con diferentes roles, y generación de informes sobre el uso de la biblioteca y la actividad de los usuarios.

# Limites

1.Complejidad del Sistema:

* **Dependencia Tecnológica:** Este sistema solo se basa en una base de datos proporcionada por los desarrolladores.
* **Capacidad de Escalabilidad: El sistema está diseñado para la gestión de la biblioteca de la FIIS y no contempla escalabilidad para usarse en varias bibliotecas o para manejar muchos usuarios a la vez.**

2. Gestión de Penalizaciones:

* Las penalizaciones y faltas están limitadas a retrasos en la devolución de libros y daños o pérdida de ejemplares, siguiendo reglas predefinidas en el sistema.

3. Usabilidad:

* La funcionalidad está limitada a las características implementadas, y no incluye personalización avanzada por parte de los usuarios.

# Requerimientos

## Requerimientos funcionales

1. El acceso al sistema de biblioteca virtual será para personas ya registradas con un usuario y contraseña. Cada usuario debe tener su rol (Alumno, Bibliotecario, Administrador) que le permitirá realizar menos o más acciones en la página de la biblioteca.
2. El sistema debe permitir la búsqueda de libros de acuerdo con distintas características y obtener información detallada de cada libro u ejemplar.
3. El sistema debe permitir realizar solicitudes de préstamos de cualquier ejemplar de libro disponible, ya sea para leerlo en el mismo establecimiento (SALA) o afuera de él (LLEVAR).
4. El sistema debe permitir registrar los préstamos y devoluciones de los alumnos y obtener un historial de ambas acciones de acuerdo con distintos indicadores.
5. El sistema debe permitir agregar “faltas” a los alumnos y penalizarlos con un monto a pagar que varía según la gravedad de penalización. Además de bloquear el acceso a los alumnos penalizados y registrar el pago de dichas penalizaciones para habilitar nuevamente a los alumnos.
6. Reportes y consultas de usuarios, préstamos, devoluciones y penalizaciones.

## Requerimientos no funcionales

1. Los permisos varían según el rol que se tiene asignado en la página:

* Los alumnos podrán buscar libros según sus preferencias y solicitar el préstamo de estos para luego recogerlos en ventanilla.
* Los bibliotecarios registrarán los préstamos y devoluciones, además de buscar los ejemplares de libros y revisar el estado de estos; también podrán consultar por los alumnos.
* Los administradores podrán editar los libros y sus ejemplares. Además de agregarle faltas a un alumno por la demora en la devolución de libros y penalizarlo de acuerdo con la gravedad de la sanción. También se encargará del registro de los pagos de los alumnos para cada penalización.

1. Las faltas se agregan solo por la demora al devolver un libro/ejemplar, y estas varían según el número de días de la demora: la falta pesa más si demoró más tiempo.
2. Las penalizaciones son de tres tipos: por sobrepasar el límite de faltas, daño de ejemplar y pérdida de ejemplar. Los montos de penalización varían según el tipo.
3. Si un alumno es penalizado, se le niega el ingreso a la página hasta que pague el monto de penalización.
4. El tiempo máximo de devolución de libros y el número máximo de faltas está registrado a una tabla la cual solo puede ser modificada por los administradores. Estos mismos también pueden modificar la tarifa de penalizaciones.
5. Desarrollado en Lenguaje Java con base de datos SQL Server.

# Actores

|  |  |
| --- | --- |
| ACTOR | DESCRIPCION |
| Administrador | Responsable del cumplimiento de normativas en la biblioteca. Sanciona a los alumnos con un monto previamente establecido. Gestiona los libros y sus ejemplares. Gestiona a los alumnos y bibliotecarios. |
| Bibliotecario | Responsable de registrar los préstamos y devoluciones de los libros. Supervisa a los alumnos y los ejemplares de libro. |
| Alumno | Realiza acciones como búsqueda de libros y las solicitudes de préstamo de estos. |

# Casos de uso

|  |  |  |
| --- | --- | --- |
| Caso de Uso | Actor | DESCRIPCIÓN |
| CU01: Registro de usuario | Alumno/Bibliotecario/Administrador | Este proceso permite al usuario registrarse dentro de la página y recibir determinada identificación. |
| CU02: Registro de préstamos | Bibliotecario | Este proceso permite al bibliotecario registrar los préstamos de ejemplares de libros. |
| CU03: Registro y devoluciones | Bibliotecario | Este proceso permite al bibliotecario registrar las devoluciones de ejemplares de libros. |
| CU04: Agregar falta a un alumno | Administrador | Este proceso es aquel por el administrador realiza tras confirmarse que un alumno cometió una falta. |
| CU05: Penalizar alumno | Administrador | Este proceso permite a un administrador penalizar a un alumno tras haber cometido algún tipo de penalización registrado en la base de datos. |
| CU06: Registrar pago penalización | Administrador | Este proceso permite al administrador registrar el pago de la penalización para que un alumno deje de estar sancionado/bloqueado. |

# Análisis de Requerimientos

## CU01: Registro de un usuario:

### Casos de uso

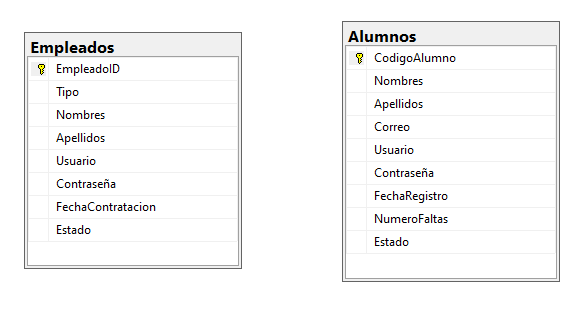


Usuario

### Especificación

|  |  |
| --- | --- |
| **IDENTIFICADOR** | CU01 |
| **NOMBRE** | Login o inicio de sesión |
| **DESCRIPCIÓN** | Representa el inicio de sesión de los usuarios en la aplicación. |
| **ACTORES** | Administradores, bibliotecarios y alumnos |
| **PRECONDICIÓN** | El usuario debe existir en la tabla Alumnos o Empleados en la base de datos. |
| **FLUJO** | 1. El usuario ejecuta la aplicación. 2. El usuario completa el formulario con su nombre de usuario y su clave de cuenta. 3. Sube su formulario y se valida si el formulario no tiene campos vacíos. 4. Se valida si el nombre de usuario esta registrado en la base de datos buscando en la tabla de Alumnos o en la tabla de Empleados. 5. Si el usuario está en la tabla de Alumnos se verifica su estado (“ACTIVO” o “BLOQUEADO”). Si se encuentra “BLOQUEADO” se le niega el acceso. 6. Si está en la tabla de Empleados o si se encuentra “ACTIVO” se valida su clave. 7. Si todos los datos son correctos, se le da acceso a la plataforma. |
| **POSTCONDICIÓN** | Dependiendo de que tipo de usuario es (Administrador, Bibliotecario o Alumno), se le permitirá ciertas tareas. |
| **EXCEPCIONES** | Ninguna. |
| **REQUERIMIENTOS ESPECIALES** | La aplicación no debe tener abierta otra cuenta de usuario. |
| **INFORMACIÓN ADICIONAL** | La aplicación solo debe ejecutarse desde la red interna de la biblioteca. |

### Diagrama E-R:



### Análisis de Caja Negra

VALIDAR

codigo

password

LoginDTO

En la clase LoginService se implementa el servicio validar, este servicio recibe como datos de entrada el nombre de usuario y su clave, si existen en la base de datos retorna un objeto de tipo LoginDTO con los datos del usuario y estado de la cuenta.

### Programación del servicio

**public** LoginDTO accesoUsuario(LoginDTO bean) **throws** SQLException {

// Validando que no esté vacia las credenciales

**if** (bean.getCodigo().isEmpty() || bean.getPassword().isEmpty()) {

**throw** **new** RuntimeException("Error, no ha insertado nada.");

}

// Validando que exista en la tabla Alumnos

String sql = "select count(1) cont from Alumnos where ";

sql += "CodigoAlumno = ? or Usuario = ?";

**int** cont = jdbcTemplate.queryForObject(sql, Integer.**class**, bean.getCodigo(), bean.getCodigo());

**if** (cont != 1) {

// Validando su existencia en la tabla Empleados

sql = "select count(1) cont from Empleados where ";

sql += "Usuario = ?";

cont = jdbcTemplate.queryForObject(sql, Integer.**class**,

bean.getCodigo());

**if** (cont != 1) {

**throw** **new** RuntimeException("Error, este usuario no existe.");

} **else** {

bean.setTipo("Empleados");

}

} **else** {

bean.setTipo("Alumnos");

// Validando que el alumno no este bloqueado

sql = "select count(1) cont from Alumnos where ";

sql += "Usuario = ? and Estado = 'ACTIVO' ";

cont = jdbcTemplate.queryForObject(sql, Integer.**class**, bean.getCodigo());

**if** (cont != 1) {

bean.setActiva(**false**);

**return** bean;

}

}

// Validando las credenciales si son correctas o no

sql = "select count(1) cont from " + bean.getTipo();

sql += " where Usuario = ? and Contraseña = ?";

cont = jdbcTemplate.queryForObject(sql, Integer.**class**,

bean.getCodigo(), bean.getPassword());

**if** (cont != 1) {

bean.setActiva(**false**);

} **else** {

bean.setActiva(**true**);

}

**return** bean;

}

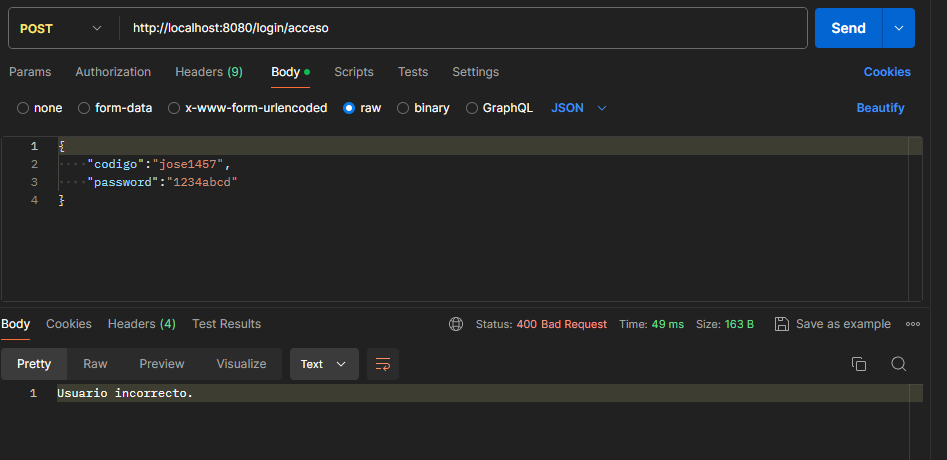
### Prueba del servicio

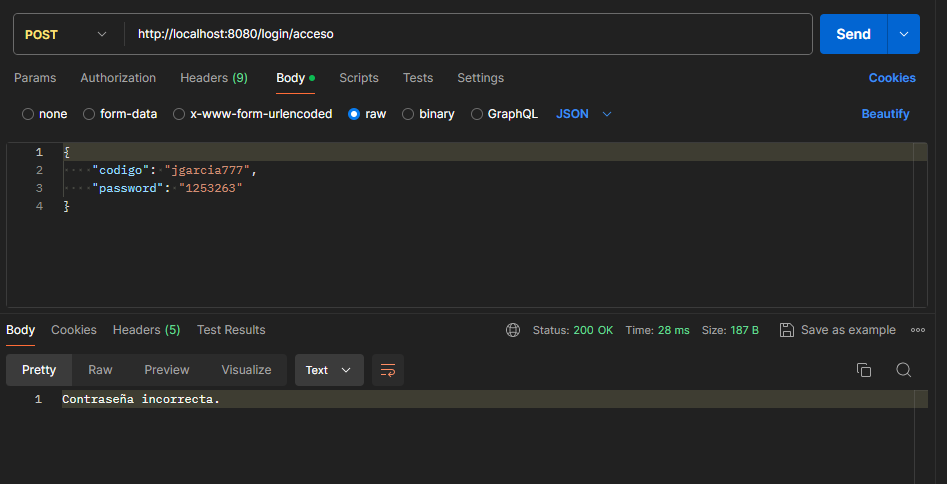
Captura de pantalla de un celular

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente





## CU02: Registro de préstamos:

### Casos de uso



Bibliotecario

### Especificación

|  |  |
| --- | --- |
| **CODIGO** | CU02 |
| **NOMBRE** | Registro de préstamos |
| **DESCRIPCION** | Este proceso permite al bibliotecario registrar los préstamos de ejemplares de libros. |
| **ACTOR** | Bibliotecario |
| **PRECONDICION** | El empleado (bibliotecario) debe estar registrado en la base de datos y debe estar “EN LABOR”. |
| **FLUJO PRINCIPAL** | |  |  | | --- | --- | | **PASO** | **ACCION** | | 1 | El bibliotecario rellena el formulario con los datos solicitados para registrar el préstamo (código alumno, id ejemplar, id empleado, tipo préstamo). | | 2 | Validación de los datos ingresados (existencia del código alumno, id ejemplar e id empleado). | | 3 | Revisar la tabla Guía para obtener los días en los que devolverá el libro, según el tipo de préstamo. | | 4 | Registrar el préstamo en la base de datos (Tabla préstamos). | | 5 | Actualizar la disponibilidad del ejemplar prestado. | |
| **POSTCONDICION** | Después de realizarse el préstamo, el alumno tiene 5 días (según lo establecido) máximos para devolver el libro |
| **EXCEPCIONES** | Ninguna. |
| **REQUERIMIENTOS ESPECIALES** | La aplicación debe estar funcionando correctamente. |
| **COMENTARIO** | El id generado para el préstamo registrado servirá para realizar otros servicios (devolver, renovar, agregar falta) |

### Diagrama E-R

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### Análisis de la caja negra

codigoAlumno

ejemplarID

**Validar**

PrestamoDTO

empleadoID

tipoPrestamo

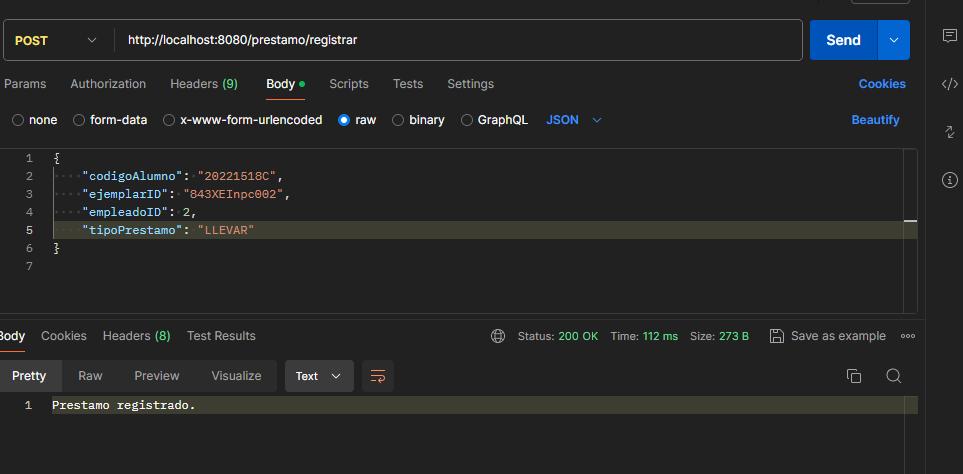
En la clase PrestamoService se implementa el servicio registrarPrestamo, este retorna un objeto de tipo PrestamoDTO luego de validar los datos de entrada.

En caso de algún error, se maneja una serie de mensajes que indicarán al bibliotecario en qué dato de ingreso se equivocó.

### Programación del servicio

public PrestamoDTO resgitrarPrestamo(PrestamoDTO dto){  
 // Validar que el alumno exista  
 String sql = "SELECT COUNT(1) filas FROM Alumnos WHERE CodigoAlumno = ?"**;**  
int filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getCodigoAlumno())**;**  
if (filas == **0**) {  
 throw new RuntimeException( "El código del alumno no existe.")**;**  
}  
 // Validar que el empleado exista  
 sql = "SELECT COUNT(1) filas FROM Empleados WHERE EmpleadoID = ?"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getEmpleadoID())**;**  
if (filas == **0**) {  
 throw new RuntimeException( "El id del empleado no existe.")**;**  
}  
 // Validar que el empleado labora  
 sql = "SELECT COUNT(1) filas FROM Empleados WHERE EmpleadoID = ? and Estado = 'EN LABOR'"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getEmpleadoID())**;**  
if (filas == **0**) {  
 throw new RuntimeException( "El empleado ya no labora en el establecimiento.")**;**  
}  
 // Validar que el ejemplar esté disponible  
 sql = "SELECT COUNT(1) filas FROM Ejemplares WHERE EjemplarID = ? AND Estado = 'DISPONIBLE'"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getEjemplarID())**;**  
if (filas == **0**) {  
 throw new RuntimeException("El ejemplar no está disponible para ser prestado.")**;**  
}  
 // Realizar el préstamo  
 if(dto.getTipoPrestamo().equals("SALA")){  
 sql = "INSERT INTO Prestamos (CodigoAlumno, EjemplarID, EmpleadoID, FechaPrestamo, FechaDevolucion, TipoPrestamo, Estado) VALUES (?, ?, ?, convert(varchar(10),getdate(),103), convert(varchar(10),getdate(),103), ?, 'NO DEVUELTO')"**;**  
jdbcTemplate.update(sql**,** dto.getCodigoAlumno()**,** dto.getEjemplarID()**,** dto.getEmpleadoID()**,** dto.getTipoPrestamo())**;**  
}  
 else{  
 sql = "INSERT INTO Prestamos (CodigoAlumno, EjemplarID, EmpleadoID, FechaPrestamo, FechaDevolucion, TipoPrestamo, Estado) VALUES (?, ?, ?, convert(varchar(10),getdate(),103), convert(varchar(10),DATEADD(day, (select Valor from Guia where Parametro='Tiempo de devolucion en dias') , GETDATE()),103), ?, 'NO DEVUELTO')"**;**  
jdbcTemplate.update(sql**,** dto.getCodigoAlumno()**,** dto.getEjemplarID()**,** dto.getEmpleadoID()**,** dto.getTipoPrestamo())**;**  
}  
 //actualizar estado del ejemplar  
 sql = "UPDATE Ejemplares SET Estado = 'PRESTADO' WHERE EjemplarID = ?"**;**  
jdbcTemplate.update(sql**,** dto.getEjemplarID())**;**  
  
return dto**;**  
}

### Prueba del servicio



## CU03: Registro de devoluciones:

### Casos de uso



Bibliotecario

### Especificación

|  |  |
| --- | --- |
| **CODIGO** | CU03 |
| **NOMBRE** | Registro de devolución |
| **DESCRIPCION** | Este proceso permite al bibliotecario registrar las devoluciones de ejemplares de libros. |
| **ACTOR** | Bibliotecario |
| **PRECONDICION** | El empleado (bibliotecario) debe estar registrado en la base de datos y debe estar “EN LABOR”. |
| **FLUJO PRINCIPAL** | |  |  | | --- | --- | | **PASO** | **ACCION**  **Validar** | | 1 | El bibliotecario rellena el formulario con los datos solicitados para registrar la devolución (código alumno, id empleado, id préstamo). | | 2 | Validación de los datos ingresados (existencia del código alumno, id empleado e id préstamo). | | 3 | A partir del id préstamo, se obtienen los otros campos para rellenar la tabla Devoluciones | | 4 | Registrar la devolución en la base de datos (Tabla devoluciones). | | 5 | Actualiza el estado del préstamo. | | 6 | Actualizar la disponibilidad del ejemplar prestado. | |
| **POSTCONDICION** | Luego de comparar el tiempo de devolución en la tabla Devoluciones y el tiempo de devolución en la tabla Prestamos, se determinará si se cometió falta o no. |
| **EXCEPCIONES** | Ninguna. |
| **REQUERIMIENTOS ESPECIALES** | La aplicación debe estar funcionando correctamente. |
| **COMENTARIO** | La tabla de Devoluciones funciona como una especie de historial para registrar la actividad de los alumnos al devolver los libros. |

### Diagrama E-R

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

### Análisis de la caja negra

prestamoID

codigoAlumno

DevolucionDTO

**Validar**

empleadoID

En la clase DevolucionService se implementó el servicio registrarDevolución, que retorna un objeto del tipo DevoluciónDTO luego de validar que código del alumno y el id del empleado existan y también el id del préstamo exista y este en estado “NO DEVUELTO”. Con el id del préstamo se realiza una consulta en la tabla Prestamos para obtener los campos con los que rellenar la tabla Devoluciones y se obtiene la fecha actual en la que se devolvió el libro. Luego de registrar el préstamo, se actualiza el estado del préstamo en “DEVUELTO”.

### Programación del servicio

public DevolucionDTO registrarDevolucion(DevolucionDTO dto){  
 //validar que el prestamo existe  
 String sql = "select count(1) filas from Prestamos P where P.PrestamoID=? and P.Estado = 'NO DEVUELTO'"**;**  
int filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getPrestamoID())**;**  
if (filas!=**1**){  
 throw new RuntimeException("Id de préstamo incorrecto. El préstamo no está registrado o ya ha sido cancelado.")**;**  
}  
 //validar codigo de alumno  
 sql = "select count(1) filas from Prestamos P where P.CodigoAlumno=? and P.PrestamoID=?"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,**dto.getCodigoAlumno()**,**dto.getPrestamoID())**;**  
if (filas!=**1**){  
 throw new RuntimeException("Código de alumno incorrecto. ")**;**  
}  
 //Validar empleado  
 sql = "select COUNT(1) filas from Empleados E where E.EmpleadoID=?"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,**dto.getEmpleadoID())**;**  
if (filas!=**1**){  
 throw new RuntimeException("Id de empleado no existe. ")**;**  
}  
 // Validar que el empleado labora  
 sql = "SELECT COUNT(1) filas FROM Empleados WHERE EmpleadoID = ? and Estado = 'EN LABOR'"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getEmpleadoID())**;**  
if (filas == **0**) {  
 throw new RuntimeException( "El empleado ya no labora en el establecimiento.")**;**  
}  
 //obtener los campos del prestamo a cancelar  
 sql = "select P.EjemplarID from Prestamos P where P.PrestamoID=?"**;**  
String EjemplarID = jdbcTemplate.queryForObject(sql**,** String.class**,** dto.getPrestamoID())**;**  
sql = "select P.FechaPrestamo from Prestamos P where P.PrestamoID=?"**;**  
String FechaPrestamo = jdbcTemplate.queryForObject(sql**,** String.class**,** dto.getPrestamoID())**;**  
//registrar devolucion  
 sql = "insert into Devoluciones values(?,?,?,?,?,convert(varchar(10),getdate(),103))"**;**  
jdbcTemplate.update(sql**,** dto.getPrestamoID()**,** dto.getCodigoAlumno()**,** dto.getEmpleadoID()**,** EjemplarID**,** FechaPrestamo)**;**  
//cambiar el estado del prestamo en la tabla  
 sql = "update Prestamos set Estado = 'DEVUELTO' where PrestamoID=?"**;**  
jdbcTemplate.update(sql**,** dto.getPrestamoID())**;**  
//cambiar el estado del ejemplar  
 sql = "update Ejemplares set Estado = 'DISPONIBLE' where EjemplarID=?"**;**  
jdbcTemplate.update(sql**,** EjemplarID)**;**  
  
return dto**;**  
  
}

### Prueba del servicio

## CU04: Agregar falta a un alumno

### Caso de Uso



Administrador

### Especificación

|  |  |
| --- | --- |
| **CODIGO** | CU04 |
| **NOMBRE** | Agregar falta a un alumno |
| **DESCRIPCION** | Este proceso es aquel por el administrador realiza tras confirmarse que un alumno cometió una falta. |
| **ACTOR** | Administrador |
| **PRECONDICION** | El empleado (administrador) debe estar registrado en la base de datos y debe estar “EN LABOR”.  El alumno debió demorarse en la devolución del libro con respecto a la fecha establecida de devolución. |
| **FLUJO PRINCIPAL** | |  |  | | --- | --- | | **PASO** | **ACCION**  **Validar** | | 1 | El administrador ingresa el código del alumno al que se le quiere agregar una falta y el id de préstamo al que corresponde la supuesta falta. | | 2 | Validación de los datos ingresados (existencia del código alumno e id préstamo simultáneamente). | | 3 | A partir del id préstamo, se obtienen las fechas de devoluciones en la tabla Prestamos y Devoluciones. Si al compararse, hay una diferencia de días, se cometió falta. | | 4 | Se obtiene el número de días de diferencia y se de acuerdo con eso se consulta en la tabla Faltas sobre el “peso” de la falta. | | 5 | Actualiza el número de faltas al alumno. | |
| **POSTCONDICION** | Si llega a sumar 5 faltas luego de agregarle, se le penaliza automáticamente (límite de faltas) |
| **EXCEPCIONES** | Ninguna. |
| **REQUERIMIENTOS ESPECIALES** | La aplicación debe estar funcionando correctamente. |
| **COMENTARIO** | Las faltas sirven como manera de controlar que los alumnos no demoren en devolver los libros que se prestan. |

### Diagrama

### E-R

Diagrama

Descripción generada automáticamente

### Análisis de la caja negra

Agregar falta

prestamoID

codigoAlumno

**Validar**

En la clase FaltasService se implementó un método AgregarFalta con la que se realiza el proceso de agregar falta a un alumno luego de validar los datos ingresados. Se valida que el id de préstamo y el código del alumno existan y coincidan en la tabla Devoluciones. Luego de comparar las fechas de devoluciones en las tablas Prestamos y Devoluciones gracias al id préstamo, se determina si el alumno cometió una falta. Si así fuese el caso, se determina el peso de la falta en la tabla Faltas buscando en el rango al que pertenece el número de días de diferencia entre las dos fechas de devolución. Luego se le suma a el número de faltas en la tabla Alumnos,

### Programación del servicio

public void AgregarFalta(String CodigoAlumno**,** int PrestamoID){  
 //verificar el ingreso del codigoalumno  
 String sql = "select count(1) filas from Alumnos where CodigoAlumno=?"**;**  
int filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** CodigoAlumno)**;**  
if (filas != **1**){  
 throw new RuntimeException("El código del alumno no existe.")**;**  
}  
 //validar el id de prestamo  
 sql = "select count(1) filas from Devoluciones where PrestamoID=? and CodigoAlumno=?"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,**PrestamoID**,**CodigoAlumno)**;**  
if (filas !=**1**){  
 throw new RuntimeException("El id de prestamo para el alumno no esta registrado en la tabla devoluciones.")**;**  
}  
 //verificar los dias de demora del prestamo  
 sql = "select convert(varchar(10),FechaDevolucion,103) FechaDevolucion from Devoluciones where PrestamoID=?"**;**  
String FechaDevolucionD = jdbcTemplate.queryForObject(sql**,** String.class**,** PrestamoID)**;**  
sql = "select convert(varchar(10),FechaDevolucion,103) FechaDevolucion from Prestamos where PrestamoID=?"**;**  
String FechaDevolucionP = jdbcTemplate.queryForObject(sql**,** String.class**,** PrestamoID)**;**  
DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("dd/MM/yyyy")**;**  
LocalDate FechaEstablecida = LocalDate.*parse*(FechaDevolucionP**,**formatter)**;**  
LocalDate FechaDevuelta = LocalDate.*parse*(FechaDevolucionD**,**formatter)**;**  
long diasDiferencia = ChronoUnit.*DAYS*.between(FechaDevuelta**,**FechaEstablecida)**;**  
if (diasDiferencia==**0**) {  
 throw new RuntimeException("No se encontró demora en la devolución del libro. No se puede agregar la falta.")**;**  
}  
 //fijar el peso de la falta segun los dias de demora  
 sql = "select PesoFalta from Faltas where ? between MinDias and MaxDias"**;**  
long PesoFalta = jdbcTemplate.queryForObject(sql**,** Long.class**,** diasDiferencia )**;**  
//actualizar el número de faltas  
 sql = "update Alumnos set NumeroFaltas + "+ PesoFalta +" where CodigoAlumno=?"**;**  
jdbcTemplate.update(sql**,**CodigoAlumno)**;**

}

### Prueba de Servicio

### 

## CU05: Penalizar a un alumno

### Casos de uso

Especificación



Administrador

|  |  |
| --- | --- |
| **CODIGO** | CU05 |
| **NOMBRE** | Penalizar a un alumno |
| **DESCRIPCION** | Este proceso permite a un administrador penalizar a un alumno tras haber cometido algún tipo de penalización registrado en la base de datos. |
| **ACTOR** | Administrador |
| **PRECONDICION** | El empleado (administrador) debe estar registrado en la base de datos y debe estar “EN LABOR”.  El alumno debió realizar alguna acción que amerite una penalización, ya sea por sobrepasar el límite de faltas, devolver el libro en mal estado o perder el libro prestado. |
| **FLUJO PRINCIPAL** | |  |  | | --- | --- | | **PASO** | **ACCION**  **Validar** | | 1 | El administrador rellena el formulario para penalizar a un alumno con los datos de ingreso (código del alumno, id emplead, tipo de penalización, monto inicial) | | 2 | Validación de los datos ingresados (existencia del código alumno e id del empleado). | | 3 | Si el tipo de penalización es por sobrepasar el límite de faltas, se verifique si es cierto esto. | | 4 | Se verifica el monto por penalización a pagar de acuerdo con el tipo de penalización. | | 5 | El monto obtenido se le suma al monto inicial y se registra con los demás datos de ingreso en la tabla Penalizaciones. | | 6 | Se modifica el estado del alumno en “BLOQUEADO”. | |
| **POSTCONDICION** | El alumno bloqueado no podrá ingresar a la página hasta que pague el monto penalizado. |
| **EXCEPCIONES** | Ninguna. |
| **REQUERIMIENTOS ESPECIALES** | La aplicación debe estar funcionando correctamente. |
| **COENTARIO** | Las penalizaciones son sanciones más severas justificadas por la gravedad de las acciones cometidas por los alumnos, para que justamente tengan cuidado en cometerlas. |

### Diagrama E-R

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

### Análisis de la caja negra

PenalizacionesDTO

codigoAlumno

## 

empleadoID

**Validar**

## 

montoInicial

tipo

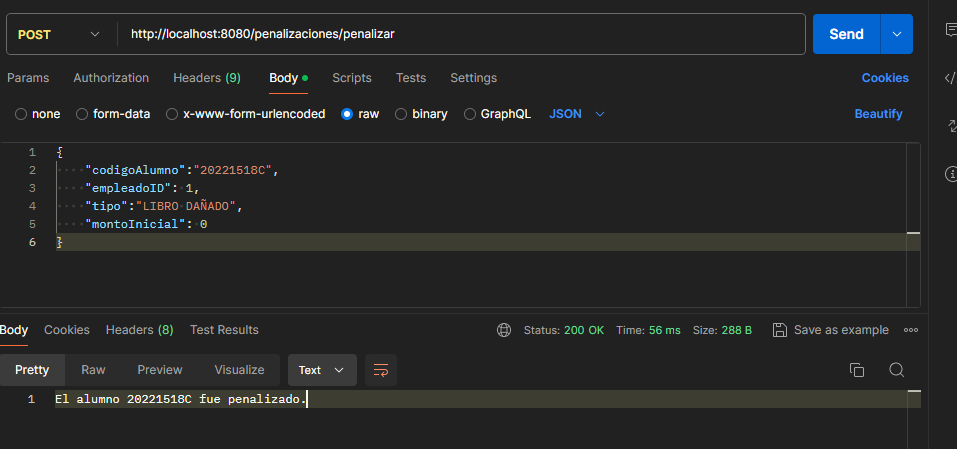
En la clase PenalizacionesService se implementó un método penalizarAlumno que devuelve un objeto de tipo PenalizacionesDTO luego de validar los datos de ingreso. Se valida la existencia del código del alumno y el id del empleado. El montoInicial ingresado depende del tipo de penalización. Si este es por “PERDIDA DE LIBRO” el monto inicial es el precio del libro perdido, en cualquier otro caso es 0. Si el tipo de penalización es por “LIMITE DE FALTAS”, se verifica si es cierto esto. Se realiza una consulta en la tabla alumnos para saber si se pasó del límite de faltas establecido en la tabla Guía. Luego de las validaciones, se obtiene el monto de penalización siguiendo lo registrado en la tabla TarifaPenalizaciones y se le suma al monto inicial. Finalmente se registra la penalización en la tabla Penalizaciones y se actualiza el estado del alumno como “BLOQUEADO”.

### Programación del servicio

public PenalizacionesDTO penalizarAlumno(PenalizacionesDTO dto){  
 //validar empleado  
 String sql = "select COUNT (1) filas from Empleados E where E.EmpleadoID=?"**;**  
int filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,**dto.getEmpleadoID())**;**  
if (filas!=**1**){  
 throw new RuntimeException("Id de empleado no existe. ")**;**  
}  
 // Validar que el empleado labora  
 sql = "SELECT COUNT(1) filas FROM Empleados WHERE EmpleadoID = ? and Estado = 'EN LABOR'"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getEmpleadoID())**;**  
if (filas == **0**) {  
 throw new RuntimeException( "El empleado ya no labora en el establecimiento.")**;**  
}  
 //verificar permisos de administrador del empleado  
 sql ="select count(1) filas from Empleados where EmpleadoID=? and Tipo = 'ADMINISTRADOR'"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,**dto.getEmpleadoID())**;**  
if (filas != **1**){  
 throw new RuntimeException("El empleado ingresado no tiene los permisos para realizar esta acción.")**;**  
}  
 //verificar existencia de alumno  
 sql = "select count(1) filas from Alumnos where CodigoAlumno=?"**;**  
filas = jdbcTemplate.queryForObject(sql**,** Integer.class**,**dto.getCodigoAlumno())**;**  
if (filas != **1**){  
 throw new RuntimeException("El código de alumno ingresado no existe.")**;**  
}  
 //verificar que se sobrepasó el límite de faltas (si fuese el caso)  
 if (dto.getTipo() == "LIMITE DE FALTAS") {  
 sql = "select NumeroFaltas from Alumnos where CodigoAlumno=?"**;**  
int NumeroFaltas = jdbcTemplate.queryForObject(sql**,** Integer.class**,** dto.getCodigoAlumno())**;**  
sql = "select Valor from Guia where Parametro='Faltas maximas'"**;**  
int Valor = jdbcTemplate.queryForObject(sql**,** Integer.class)**;**  
if (NumeroFaltas < Valor) {  
 throw new RuntimeException("El alumno no paso el límite máximo de faltas establecido.")**;**  
}  
 }  
 //obtener monto segun tipo de penalizaciones  
 sql = "select Monto from TarifaPenalizaciones where Tipo=?"**;**  
double Monto = jdbcTemplate.queryForObject(sql**,** Double.class**,**dto.getTipo())**;**  
Monto += dto.getMontoInicial()**;**  
//registrar penalizacion  
 sql = "insert into Penalizaciones values(?,?,?,convert(varchar(10),getdate(),103),?,'NO PAGADO')"**;**  
jdbcTemplate.update(sql**,**dto.getCodigoAlumno()**,**dto.getTipo()**,**dto.getEmpleadoID()**,**Monto)**;**  
//actualizar estado de alumno penalizado  
 sql = "update Alumnos set Estado = 'BLOQUEADO' where CodigoAlumno=?"**;**  
jdbcTemplate.update(sql**,**dto.getCodigoAlumno())**;**  
  
return dto**;**

}

### Prueba del servicio



## CU06: Registrar pago penalización

### Casos de uso

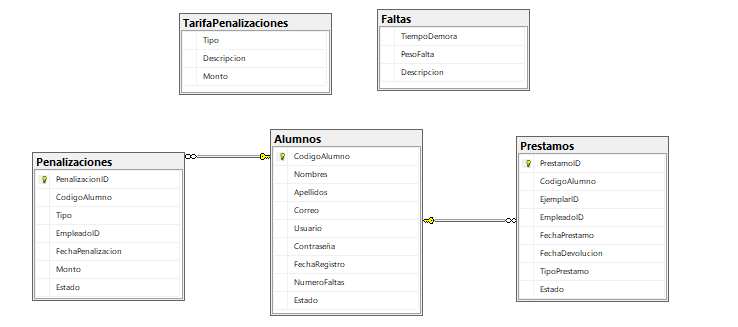


Administrador

### Especificación:

|  |  |
| --- | --- |
| **IDENTIFICADOR** | CU06 |
| **NOMBRE** | Registrar pago de una penalización |
| **DESCRIPCIÓN** | Representa el pago del monto que los usuarios en estado “BLOQUEADO” deberán pagar. |
| **ACTORES** | Administrador. |
| **PRECONDICIÓN** | Debe de existir la penalización con el código del usuario y los datos correspondientes en la tabla Penalización y su campo estado debe ser “NO PAGADO”. |
| **FLUJO** | 1. El empleado abre la aplicación y sube un formulario con su código de empleado, código de alumno, el código de la penalización. 2. La aplicación empieza validando la existencia del empleado en la Tabla de empleados y si su rol de empleado es Administrador. 3. Valida la existencia del registro con el código de penalización. 4. Valida si coincide el registro de penalización con el código del usuario. 5. Valida si el monto dado es el correcto. 6. De estar todos los datos correctos, actualiza el registro con el estado de “PAGADO”. 7. Actualiza el estado del alumno de “BLOQUEADO” a “ACTIVO”. |
| **POSTCONDICIÓN** | El usuario volvía al estado “ACTIVO” y podía volver a iniciar sesión con normalidad. |
| **EXCEPCIONES** | Cuando el tipo de penalización sea “PERDIDA DE LIBRO”, el administrador podrá decidir cuál será el monto que tendrá que pagar el alumno. |
| **REQUERIMIENTOS ESPECIALES** | Se debe de haber registrado la penalización. |
| **INFORMACIÓN ADICIONAL** | La aplicación solo debe ejecutarse desde la red interna de la biblioteca. |

### Diagrama E-R:



### Análisis de Caja Negra:

**VALIDAR Y**

**ACTUALIZAR**

registrarPagoPenalizacionDTO

registrarPagoPenalizacionDTO

En la clase PenalizacionesService se implementó un método registrarPago que devuelve un objeto de tipo registrarPagoPenalizacionDTO luego de validar los datos de ingreso. Se valida la existencia del código del alumno y el id del empleado y su rol de Administrador antes de realizar la transacción. El montoInicial ingresado depende de si el tipo de penalización es “PERDIDA DE LIBRO” el monto inicial es el precio del libro perdido, en caso sea 0 en el registro de Penalizaciones. Finalmente se actualiza el registro de la penalización en la tabla Penalizaciones con el estado “PAGADO” si todos los datos son correctos y se actualiza el estado del alumno como “ACTIVO”.

### Programación del servicio

@Transactional(propagation = Propagation.***REQUIRES\_NEW***, rollbackFor = Exception.**class**)

**public** registrarPagoPenalizacionDTO registrarPago(registrarPagoPenalizacionDTO dto){

//Validar empleado

String sql = "select COUNT(1) filas from Empleados E where E.EmpleadoID=?";

**int** filas = jdbcTemplate.queryForObject(sql, Integer.**class**,dto.getEmpleadoID());

**if** (filas!=1){

**throw** **new** RuntimeException("Id de empleado no existe. ");

}

//verificar permisos de administrador del empleado

sql ="select count(1) filas from Empleados where EmpleadoID=? and Tipo = 'ADMINISTRADOR'";

filas = jdbcTemplate.queryForObject(sql, Integer.**class**,dto.getEmpleadoID());

**if** (filas != 1){

**throw** **new** RuntimeException("El empleado ingresado no tiene los permisos para realizar esta acción.");

}

//validar que la penalizacion existe

sql = "select count(1) filas from Penalizaciones P where P.PenalizacionID=? and P.Estado = 'NO PAGADO'";

filas = jdbcTemplate.queryForObject(sql, Integer.**class**, dto.getPenalizacionID());

**if** (filas!=1){

**throw** **new** RuntimeException("Id de prenalización incorrecto. La penalización no está registrado o ya ha sido pagada.");

}

//validar codigo de alumno

sql = "select count(1) filas from Penalizaciones P where P.CodigoAlumno=? and P.PenalizacionID=?";

filas = jdbcTemplate.queryForObject(sql, Integer.**class**,dto.getCodigoAlumno(),dto.getPenalizacionID());

**if** (filas!=1){

**throw** **new** RuntimeException("Código de alumno incorrecto. ");

}

//verificar monto correcto

sql = "select Monto from Penalizaciones where PenalizacionID=?";

**double** montoRegistrado = jdbcTemplate.queryForObject(sql, Double.**class**,dto.getPenalizacionID());

**if**(dto.getMonto() != montoRegistrado){

**throw** **new** RuntimeException("El monto ingresado es incorrecto.");

}

//obtener el tipo de penalizacion

sql = "select Tipo from Penalizaciones where PenalizacionID=?";

String Tipo = jdbcTemplate.queryForObject(sql, String.**class**, dto.getPenalizacionID());

//actualizar el estado de la penalizacion

sql = "update Penalizaciones set Estado = 'PAGADO' where PenalizacionID=?";

jdbcTemplate.update(sql,dto.getPenalizacionID());

//actualizar estado del alumno

sql = "update Alumnos set Estado = 'ACTIVO' where CodigoAlumno=?";

jdbcTemplate.update(sql,dto.getCodigoAlumno());

**if** (Tipo.equals("LIMITE DE FALTAS")){

//reiniciar número de faltas

sql = "update Alumnos set NumeroFaltas = 0 where CodigoAlumno=?";

jdbcTemplate.update(sql,dto.getCodigoAlumno());

}

**return** dto;

}

### Prueba de servicio