Gestor de Biblioteca Escolar

Trabajo de Fin del Grado Superior de

Desarrollo de Aplicaciones Web

Álvaro Allén Perlines

Resumen

Este documento trata acerca del desarrollo completo de una aplicación web de gestión de bibliotecas escolares, la cual realiza el mantenimiento de préstamos de libros a usuarios del centro, controlando el catálogo que se oferta. A continuación, se detallará el motivo de esta idea, un estudio teórico sobre la tecnología usada y porque ha sido escogida, el apartado teórico de la aplicación (planos, modelos y estructura teórica de la aplicación), el apartado práctico de la aplicación con ejemplos de uso real y una conclusión en la que se detallarán mejoras posibles a futuro.

Objetivos

El presente proyecto tiene como objetivo el desarrollo de una aplicación web orientada a la gestión de una biblioteca escolar. Esta herramienta está diseñada para facilitar el trabajo de la persona encargada de la biblioteca, permitiendo gestionar de forma eficiente los préstamos realizados durante el curso escolar, el catálogo de libros junto con su disponibilidad en todo momento y el seguimiento de los usuarios (alumnos y profesores) afiliados a la biblioteca.

Uno de los principales objetivos del proyecto es implementar un sistema que permita registrar y controlar el ciclo completo de un préstamo, desde su inicio hasta su finalización, incluyendo la gestión de devoluciones y la detección de posibles retrasos. También se podrá dar información acerca del estado de los ejemplares en todo momento, dándole la opción al trabajador de dar la baja de ejemplares en mal estado o de notificar posibles incidencias.

Otro de los objetivos fundamentales es el diseño de una estructura de base de datos coherente y escalable, que permita representar correctamente las entidades implicadas en el sistema, como usuarios, libros, ejemplares y préstamos. Este diseño, a mayores, tienen implementado un sistema de auditoría para poder dar solución a posibles problemas administrativos. Con este diseño se podrán añadir ampliaciones de manera sencilla, sin tocar el resto de la estructura.

Además, el proyecto tiene como finalidad servir como medio de aprendizaje en el desarrollo de aplicaciones web utilizando tecnologías modernas como son los frameworks, herramientas esenciales en el desarrollo actual que facilitan el comienzo y, sobre todo, la configuración del proyecto.

Por último, se plantea como objetivo adicional la implementación de funcionalidades de importación y exportación de datos en formato JSON, con el fin de facilitar la carga masiva de información y simular procesos reales como la adquisición o donación de libros.

La elección de este proyecto se basa en la necesidad de resolver la problemática habitual en entornos educativos, donde la gestión de bibliotecas escolares suele realizarse de manera manual, usando cuadernos de cálculo que se van acumulando o incluso extraviando, o también suelen usar herramientas poco especializadas como una hoja de excel o access (está última sería una solución muy efectiva en entornos pequeños éste) las cuales no hacen mejor el flujo de trabajo.

El desarrollo de esta aplicación permite abordar dicha problemática mediante una solución sencilla pero funcional, adaptada a las necesidades de un centro educativo. Además, el proyecto resulta adecuado desde el punto de vista formativo, ya que permite aplicar y consolidar conocimientos acerca del desarrollo de un CRUD, tipo de aplicación bastante extendida en el mundo empresarial que, hoy en día se está digitalizando.

Asimismo, se trata de un proyecto escalable que puede evolucionar en el futuro, incorporando nuevas funcionalidades como sistemas recomendación, gestión de donaciones o mejoras en la experiencia de usuario, lo que lo convierte en una base sólida para futuros desarrollos.

Estudio

A continuación, se va a realizar un estudio bastante detallado acerca de las tecnologías usadas en este proyecto, escogidas para facilitar su configuración, desarrollo y puesta en marcha. La tecnología escogida es Spring, un framework bastante usado en la actualidad por muchas empresas de tecnológicas debido a su versatilidad y a su implementación sencilla. En concreto, se ha escogido Spring Boot, una tecnología procedente de Spring.

Spring Framework

¿Qué es?

Spring es un framework de código abierto orientado al desarrollo de aplicaciones empresariales en Java. Está diseñado para facilitar la creación de aplicaciones robustas, escalables y mantenibles que se ejecutan sobre la Máquina Virtual de Java (JVM).

Una de las características principales de Spring es la inversión de control y la inyección de dependencias, mecanismos que permiten desacoplar los distintos componentes de una aplicación. Gracias a ello, los objetos no crean directamente sus dependencias, sino que estas son proporcionadas automáticamente por el contenedor de Spring. Este enfoque facilita el desarrollo de aplicaciones modulares y estructuradas, mejorando su mantenimiento y escalabilidad.

Además, Spring proporciona herramientas y servicios integrados para tareas comunes en el desarrollo de aplicaciones, como la validación de datos, gestión de excepciones, acceso a base de datos, administración de recursos o tratamiento de peticiones web. También ofrece compatibilidad con numerosas tecnologías del ecosistema Java, convirtiéndose en uno de los frameworks más utilizados en el desarrollo backend.

IoC

La Inversión de Control (IoC) es un principio de diseño de software en el que la creación y gestión de los objetos deja de ser responsabilidad directa del programador y pasa a ser controlada por un contenedor externo.

En Spring Framework, este contenedor recibe el nombre de “IoC Container o “Application Context”. Su función principal es crear, configurar y administrar los distintos componentes de la aplicación, así como gestionar las relaciones existentes entre ellos.

Gracias a este enfoque, los diferentes módulos de la aplicación permanecen desacoplados, facilitando el mantenimiento, la reutilización y la escalabilidad del software.

DI

Para aplicar el principio IoC, Spring utiliza la Inyección de Dependencias (Dependency Injection o DI), un mecanismo mediante el cual el contenedor proporciona automáticamente las dependencias que necesita cada componente de la aplicación.

Habitualmente, en programación orientada a objetos, una clase depende de otras para poder realizar determinadas funciones. Por ejemplo, un controlador necesita un servicio para ejecutar la lógica de negocio y este, a su vez, un repositorio para acceder a la base de datos. Sin inyección de dependencias, cada clase tendría que crear manualmente las instancias de las que depende, aumentando el acoplamiento entre componentes.

Mediante DI, Spring se encarga de crear y administrar estas instancias, conocidas como beans, dentro de su contenedor IoC. Posteriormente, el framework inyecta automáticamente las dependencias necesarias en cada componente de la aplicación. Gracias a este enfoque, las clases permanecen desacopladas y cada una mantiene una responsabilidad concreta dentro de la arquitectura del sistema.

Spring permite realizar la inyección de dependencias de diferentes formas, siendo la más habitual la inyección mediante constructor. Actualmente, este método es el más recomendado debido a que mejora la inmutabilidad del código y facilita la realización de pruebas unitarias.

```java
@Service
public class UsuarioService {

    public List<Usuario> obtenerUsuarios() {
        return new ArrayList<>();
    }
}
```

```java
@Controller
public class UsuarioController(){

  @Autowired
    private UsuarioService usuarioService;

  @GetMapping("/Usuario")
  public String listarUsuarios(Model modelo){

    model.addAttribute("usuarios", usuarioService.obtenerUsuarios());

    return "mtoUsuarios";
  }
}
```

En este ejemplo, Spring detecta automáticamente la clase anotada con @Service y crea un bean gestionando por el contenedor IoC. Posteriormente, la dependencia es inyectada en el controlador mediante el constructor, permitiendo utilizar el servicio sin necesidad de instanciarlo manualmente.

Ventajas

Spring Framework ofrece varias ventajas relevantes en el desarrollo de aplicaciones empresariales en Java.

En primer lugar, destaca en el desacoplamiento entre componentes, conseguido mediante el IoC y el DI, lo que facilita la modificación y mantenimiento del código.

Otra ventaja es la organización en capas, que permite estructurar la aplicación en capas, de forma clara y controladores, servicios y repositorios, mejorando la legibilidad y escalabilidad del proyecto.

Además, Spring dispone de un amplio ecosistema de integración, lo que permite trabajar fácilmente con bases de datos, servicios web y otras tecnologías del entorno Java.

Por último, la facilidad de mantenimiento y evolución del sistema convierte a Spring en una opción adecuada para aplicaciones que requieren crecimiento progresivo, como es el caso del presente proyecto.

Spring Boot

¿Qué es?

Spring Boot es un framework basado en Spring Framework diseñado para simplificar el desarrollo de aplicaciones web en Java, reduciendo la configuración manual necesaria en los proyectos tradicionales.

Su principal objetivo es facilitar la creación de aplicaciones listas para producción de forma rápida, proporcionando una configuración predeterminada que permite al desarrollador centrarse en la lógica de negocio en lugar de en la configuración del entorno.

Spring Boot automatiza gran parte de la configuración del proyecto, incluyendo la gestión de dependencias y la configuración interna del framework, siguiendo un enfoque basado en convenciones.

Características

Autoconfiguración

La autoconfiguración es uno de los pilares fundamentales de Spring Boot. Este mecanismo permite que el framework configure automáticamente la aplicación en función de las dependencias incluidas en el proyecto.

De esta forma, Spring Boot analiza las librerías disponibles y configura de manera automática los componentes necesarios del framework Spring, reduciendo significativamente la necesidad de configuración manual por parte del desarrollador.

Este enfoque ayuda a acelerar el desarrollo y disminuye la probabilidad de errores derivados de configuraciones incorrectas.

Servidores embebidos

Spring Boot incorpora servidores embebidos como Apache Tomcat, Jetty o Undertow, lo que permite ejecutar la aplicación sin necesidad de desplegarla en un servidor externo.

Esto simplifica el proceso de despliegue, ya que la aplicación puede ejecutarse directamente como un archivo ejecutable (JAR), facilitando tanto el desarrollo como la puesta en producción.

Aplicaciones autónomas

gracias a la integración de servidores embebidos y la configuración automática, Spring Boot permite crear aplicaciones autónomas que no dependen de configuraciones externas complejas.

Esto facilita que la aplicación pueda ejecutarse en diferentes sin necesidad de ajustes adicionales, lo que mejora la portabilidad y simplifica el proceso de despliegue.

Spring Boot MVC

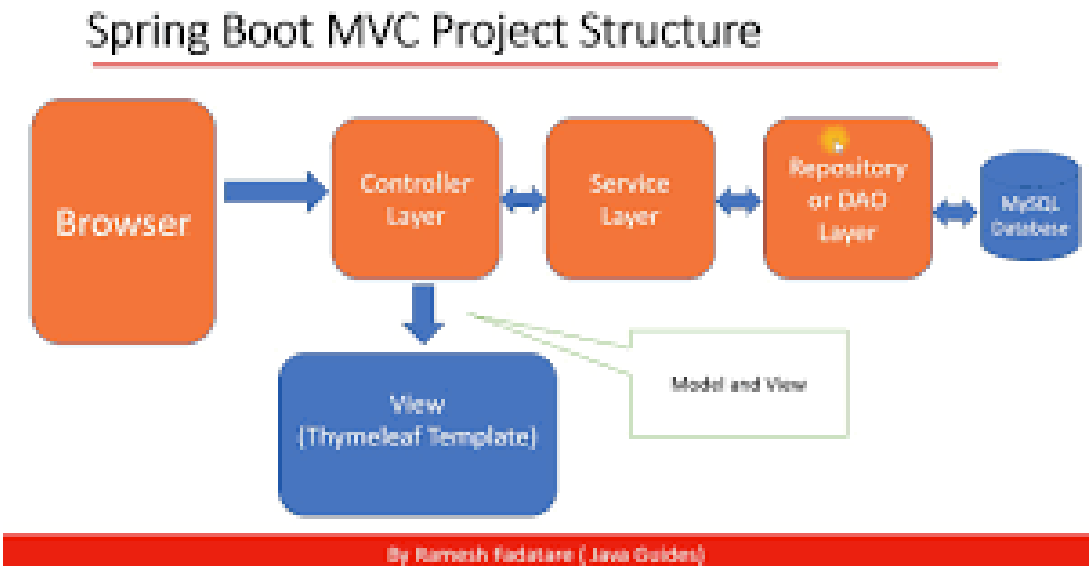
Arquitectura MVC

La arquitectura Modelo-Vista-Controlador es un patrón de diseño ampliamente utilizado en el desarrollo de aplicaciones web. En el caso de Spring Boot, este patrón se implementa a través de Spring MVC, lo que permite estructurar la aplicación en distintas capas con responsabilidades bien diferenciadas.

Esta separación favorece la organización del código, facilitando su mantenimiento, escalabilidad y reutilización.

Los componentes del patrón MVC son:

- **Modelo:** representa la estructura de datos de la aplicación y la lógica de negocio asociada. En aplicaciones basadas en Spring, suele estar formado por entidades que se relacionan directamente con la base de datos.
- **Vista:** es la capa encargada de la presentación de la información al usuario. En aplicaciones Spring Boot es habitual el uso de tecnologías como HTML junto con motores de plantillas como Thymeleaf, que permiten generar contenido dinámico a partir de los datos del modelo.
- **Controlador:** es el intermediario entre la vista y el modelo. Su función es recibir las peticiones HTTP realizadas por el usuario, procesarlas y coordinar la interacción entre las distintas capas de la aplicación para generar una respuesta adecuada.



DispatcherServlet

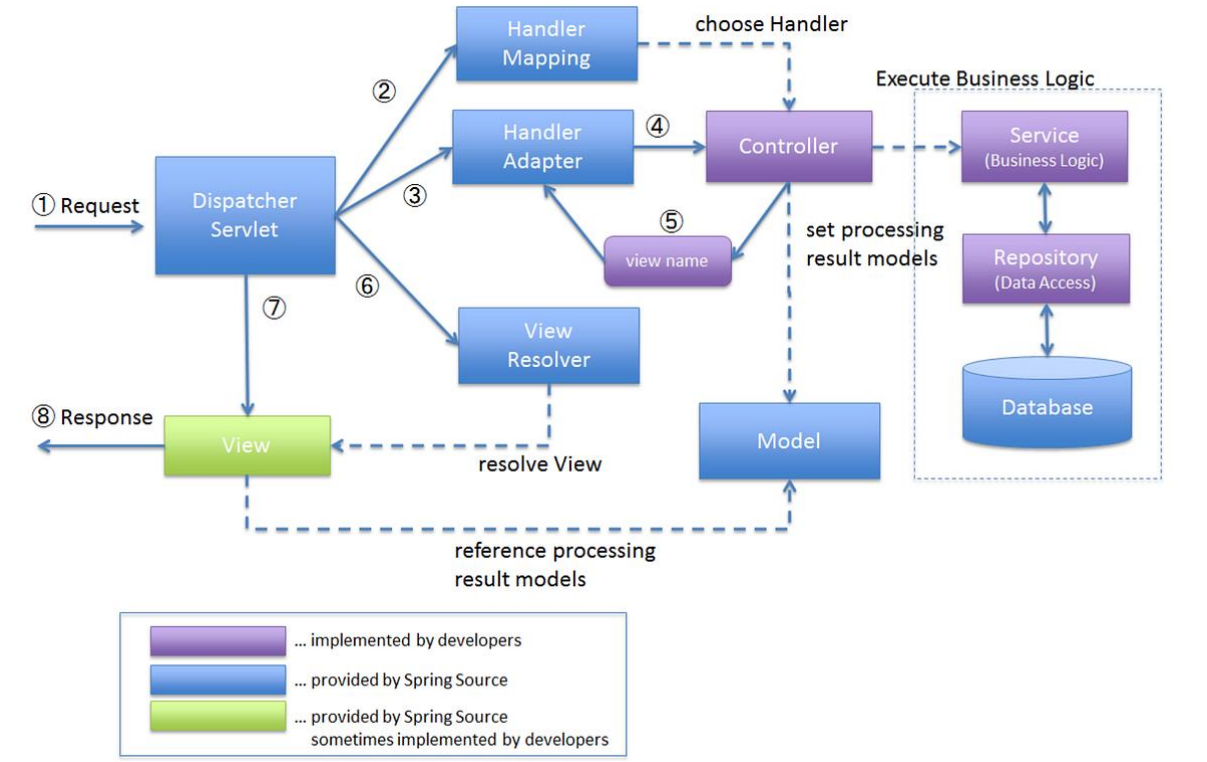
El DispatcherServlet es el componente central de Spring MVC y actúa como Front Controller, es decir, como punto único de entrada para todas las peticiones HTTP que recibe la aplicación. Su función principal es gestionar el flujo de las solicitudes dentro del framework, delegando cada petición al controlador correspondiente y coordinando la generación de la respuesta final.

En el caso de Spring Boot, este componente se configura automáticamente al incluir la dependencia de Spring Web, por lo que no es necesario realizar una configuración manual.

Flujo de petición HTTP

El procesamiento de una petición HTTP en Spring MVC sigue una secuencia bien definida gestionada por el DispatcherServlet:

1. El cliente (navegador o aplicación frontend) envía una petición HTTP (GET, POST, etc...).
2. El DispatcherServlet recibe la solicitud como punto de entrada único.
3. A través de Handler Mapping, se determina qué controlador es responsable de procesar la petición.
4. El controlador ejecuta la lógica de negocio necesaria, normalmente apoyándose en la capa de servicio.
5. El resultado se devuelve al DispatcherServlet.
6. Si la aplicación es una API REST, el resultado se convierte automáticamente en formato JSON mediante convertidores internos.
7. Si se trabaja con vistas, se devuelve el nombre de la vista que será renderizada por el motor de plantillas correspondiente.



Arquitectura implementada

Controller

En Spring MVC, un Controller es el componente encargado de gestionar las peticiones HTTP que llegan a la aplicación, actuando como intermediario entre la capa de presentación y la lógica de negocio.

Su principal función es recibir las solicitudes del usuario, procesar los parámetros asociados y delegar la ejecución de la lógica a la capa de servicio correspondiente. Posteriormente, devuelve una respuesta adecuada, ya sea en forma de vista o de datos estructurados.

En el contexto de Spring Boot, los controladores se definen mediante anotaciones como @Controller o @RestController, lo que permite a Spring identificarlos y gestionarlos automáticamente dentro del contenedor de IoC.

Los controladores no deben contener lógica de negocio compleja. Su responsabilidad se limita a: gestionar la entrada de datos, coordinar la ejecución de servicios y devolver la respuesta adecuada. De esta forma, la lógica de negocio se delega a la capa de servicio, favoreciendo una arquitectura desacoplada y mantenible.

Hay dos tipos de controladores en Spring Boot:

- **Controlador tradicional (@Controller):** este tipo de controlador se utiliza cuando la aplicación genera vistas. En este caso, el método del controlador devuelve el nombre de una plantilla, que posteriormente será renderizada por el motor de vistas como Thymeleaf.
- **Controlador REST (@RestController):** este tipo de controlador está orientado a la creación de APIs REST. En este caso, los métodos devuelven directamente datos estructurados, normalmente en formato JSON, que son enviados al cliente sin pasar por una vista intermedia.

Service

En Spring MVC, la capa Service es la encargada de implementar la lógica de negocio de la aplicación. Se trata de una capa intermedia entre los controladores y la capa de acceso a datos, cuya función principal es procesar la información recibida y aplicar las reglas de negocio correspondientes.

Las clases de esta capa se suelen definir mediante la anotación @Service en Spring Boot, lo que permite que sean gestionadas automáticamente por el contenedor de Spring como componentes del sistema.

La capa de servicio actúa como punto central de la lógica de la aplicación, coordinando las operaciones necesarias para cumplir con los requisitos funcionales del sistema. Entre sus principales responsabilidades se encuentran:

- Implementar la lógica de negocio de la aplicación.
- Coordinar la interacción entre controladores y repositorios.
- Gestionar procesos que impliquen varias operaciones sobre los datos.

El uso de esta capa aporta varias ventajas en el diseño de la arquitectura:

- **Separación de responsabilidades:** permite que los controladores se encarguen únicamente de la gestión de peticiones HTTP, evitando la inclusión de lógica de negocio en esta capa.
- **Reutilización de lógica:** la misma lógica de negocio puede ser utilizada por distintos controladores o servicios, evitando duplicación de código.
- **Mantenibilidad y testeo:** al centralizar la lógica de negocio, se facilita la realización de pruebas unitarias y la modificación del comportamiento de la aplicación sin afectar a otras capas.

Repository

En Spring MVC, la capa Repository es la encargada de gestionar el acceso a la base de datos de la aplicación. Su función principal es proporcionar una abstracción del sistema de persistencia, permitiendo realizar operaciones sobre los datos sin necesidad de implementar consultas SQL de forma manual.

En Spring Boot, los repositorios forman parte de la capa de acceso de datos y se apoyan en Spring Data JPA, lo que permiten trabajar directamente con entidades Java y simplificar las operaciones de persistencia.

Las características principales del Repository son:

- **Interfaz como base de implementación:** se definen normalmente como interfaces. Spring genera automáticamente su implementación en tiempo de ejecución, lo que reduce la necesidad de código repetitivo.
- **Uso de Spring Data JPA:** los repositorios suelen extender interfaces proporcionadas por Spring Data como JpaRepository o CrudRepository, que incluyen operaciones CRUD básicas y permiten la creación de consultas derivadas a partir del nombre de los métodos.

La capa repository actúa como punto de acceso a la base de datos, siendo utilizada por la capa Service para realizar operaciones de lectura y escritura sobre las entidades del sistema. De esta forma, se mantiene una separación clara entre la lógica de negocio y el acceso a datos.

Entity

En Spring MVC y Spring Boot, una Entity (entidad) es una clase Java que representa una tabla dentro de una base de datos relacional. Cada instancia de esta clase se corresponde con un registro de la tabla, mientras que sus atributos representan las columnas de dicha tabla. Las entidades forman parte de la capa de modelo de la arquitectura MVC y constituyen el núcleo de la representación de los datos de la aplicación.

Las entidades se utilizan junto con anotaciones del estándar JPA, lo que permite establecer la correspondencia entre objetos Java y tablas de la base de datos. Este proceso se conoce como mapeo objeto-relacional (ORM) y es gestionado por frameworks como Hibernate dentro del ecosistema de Spring.

Gracias a este enfoque, es posible trabajar con objetos Java sin necesidad de escribir consultas SQL para las operaciones básicas, ya que el framework se encarga de la persistencia de forma automática.

Las entidades constituyen la base del modelo de datos de la aplicación y son utilizadas por la capa Repository para realizar operaciones de acceso a la base de datos. A su vez, la capa Service utiliza estas entidades para aplicar lógica de negocio, lo que permite mantener una arquitectura estructurada y desacoplada.

Persistencia de datos

Spring Data JPA

Spring Data JPA es un módulo de Spring Framework orientado a simplificar el acceso y la gestión de datos en aplicaciones Java. Su principal objetivo es reducir la cantidad de código necesario para implementar la capa de persistencia, automatizando gran parte de las operaciones habituales sobre bases de datos relacionales.

Spring Data JPA se basa en el estándar JPA (Jakarta Persistence API), utilizando el mapeo objeto-relacional (ORM) para relacionar entidades Java con tablas de la base de datos.

Funcionamiento

El funcionamiento de Spring Data JPA se apoya en la definición de entidades y repositorios:

- Las entidades representan las tablas de la base de datos mediante anotaciones como @Entity o @Id.
- Los repositorios permiten realizar operaciones de persistencia extendiendo interfaces como JpaRepository o CrudRepository.

A partir de estas interfaces, Spring genera automáticamente la implementación necesaria en tiempo de ejecución, proporcionando métodos para realizar operaciones CRUD sin necesidad de escribir consultas SQL manualmente.

Características principales

- **Generación automática de operaciones CRUD:** Spring data JPA incorpora métodos predefinidos realizar operaciones básicas sobre los datos, como guardar, buscar, actualizar o eliminar registros.
- **Consultas derivadas de métodos:** permite generar consultas automáticamente a partir del nombre de los métodos definidos en el repositorio, simplificando la creación de búsquedas personalizadas.
- **Paginación y ordenación:** incluye soporte integrado para paginación y ordenación de resultados, facilitando el manejo eficiente de grandes volúmenes de datos.
- **Consultas personalizadas:** aunque automatiza gran parte de la persistencia, también permite definir consultas específicas mediante anotaciones como @Query, utilizando JPQL o SQL nativo cuando es necesario.

Hibernate

HIbernate es un framework de mapeo objeto-relacional (ORM) para Java que permite gestionar bases de datos utilizando objetos Java en lugar de trabajar directamente con consultas SQL.

Su función principal es establecer la correspondencia entre las clases de la aplicación y las tablas de la base de datos, facilitando la persistencia de datos y reduciendo la complejidad del acceso a información almacenada.

Hibernate es la implementación más utilizada en JPA, el estándar de Java para la persistencia de datos. En aplicaciones desarrolladas con Spring Boot, Hibernate actúa internamente como proveedor de persistencia, siendo utilizado automáticamente por Spring Data JPA para gestionar las operaciones sobre la base de datos.

Gracias a esta integración el desarrollador puede trabajar directamente con entidades Java mientas Hibernate se encarga de generar y ejecutar las consultas SQL necesarias.

Funcionamiento

Hibernate utiliza anotaciones como @Entity, @Table o @Column para realizar el mapeo entro objetos y tablas relacionales. A partir de esta configuración, el framework administra automáticamente operaciones como inserciones, actualizaciones, consultas o eliminaciones de datos.

Además, Hibernate proporciona su propio lenguaje de consultas orientado a objetos, denominado HQL (Hibernate Query Language), que permite realizar consultas sobre entidades Java en lugar de trabajar directamente sobre tablas físicas.

Ventajas principales

El uso de Hibernate aporta diversas ventajas en el desarrollo de aplicaciones:

- Reducción de código SQL manual.
- Mayor independencia respecto al sistema gestor de bases de datos utilizado.
- Simplificación de la persistencia de datos mediante ORM.
- Integración directa con Spring Boot y Spring Data JPA.

MySQL

MySQL es un sistema gestor de bases de datos relacional ampliamente utilizado en el desarrollo de aplicaciones web y empresariales. Su funcionamiento se basa en el almacenamiento de información mediante tablas relacionadas entre sí, utilizando el lenguaje SQL para la gestión y consulta de datos.

En aplicaciones desarrolladas con Spring Boot, MySQL se integra habitualmente junto a tecnologías como Spring Data JPA e Hibernate, permitiendo gestionar la persistencia de datos de forma estructurada mediante entidades Java y mapeo objeto-relacional.

La conexión entre Spring Boot y MySQL se realiza mediante el controlador JDBC correspondiente, normalmente incluido a través de la dependencia MySQL Connector. A partir de esta configuración, la aplicación puede realizar operaciones CRUD sobre la base de datos de forma automática utilizando repositorios JPA.

En este proyecto, MySQL se utiliza como sistema de almacenamiento persistente para gestionar la información de la aplicación y mantener las relaciones entre las distintas entidades del sistema.

Gestión de dependencias

Maven

Apache Maven es una herramienta de gestión y automatización de proyectos utilizada principalmente en aplicaciones Java. Su función principal es administras las dependencias del proyecto, automatizar el proceso de compilación y facilitar el empaquetado y despliegue de aplicaciones.

En proyectos desarrollados con Spring Boot, Maven se utiliza para gestionar de forma centralizada todas las librerías y módulos necesarias para el funcionamiento de la aplicación.

Gestión de dependencias

Una de las principales características de Maven es la gestión automática de dependencias. En lugar de descargar manualmente archivo .jar, el desarrollador únicamente debe indicar las librerías necesarias en el archivo de configuración del proyecto, permitiendo que Maven las descargue automáticamente desde repositorios remotos.

Este enfoque simplifica el desarrollo y garantiza una gestión más organizada y mantenible de las dependencias utilizadas.

Automatización del ciclo de vida

Maven también permite automatizar distintas fases del desarrollo mediante comandos predefinidos, incluyendo: compilación del código fuente, ejecución de pruebas, generación de paquetes ejecutables y limpieza de archivos temporales. De esta forma, se facilita la construcción y despliegue de aplicaciones de manera estandarizada.

Estructura estandarizada

Otra característica importante de Maven es la estandarización de la estructura de los proyectos Java. Esto permite mantener una organización común de carpetas y recursos, facilitando la compresión y mantenimiento del proyecto por parte de otros desarrolladores.

Pom.xml

El archivo pom.xml (Project Object Model) es el fichero principal de configuración utilizado por Apache Maven en proyectos Java. En él se define toda la información necesaria para la construcción y gestión del proyecto.

Este archivo contiene elementos como: las dependencias utilizadas por la aplicación, la configuración de compilación, los plugins empleados durante el desarrollo y la información general del proyecto.

En aplicaciones desarrolladas con Spring Boot, el archivo pom.xml tiene un papel especialmente importante, ya que permite incorporar fácilmente módulos y dependencias mediante los denominados Spring Starters. Estas dependencias agrupan configuraciones y librerías habituales para simplificar el desarrollo de aplicaciones web, persistencia de datos o APIs REST.

Además, Maven utiliza este archivo para descargar automáticamente las dependencias necesarias desde repositorios remotos y gestionar las versiones compatibles entre librerías. En este proyecto, el archivo pom.xml se emplea para configurar las dependencias principales del sistema, incluyendo Spring Web, Thymeleaf, Spring Data JPA y el conector de MySQL, entre otras herramientas necesarias para el funcionamiento de la aplicación

Spring Initializr

Spring Initializr es una herramienta oficial del ecosistema Spring Boot que permite generar automáticamente la estructura base de proyectos Spring Boot de forma rápida y simplificada. Su principal objetivo es facilitar la creación inicial del proyecto, evitando configuraciones manuales complejas y proporcionando una estructura preparada para comenzar el desarrollo de la aplicación.

La herramienta permite configurar distintos parámetros del proyecto antes de su generación, entre los que destacan: el sistema de construcción utilizado (Maven o Gradle), el lenguaje de programación, la versión de Spring Boot, la versión de Java, el tipo de empaquetado y las dependencias necesarias para el proyecto.

Uno de los aspectos más relevantes de Spring Initializr es la posibilidad de incorporar dependencias mediante Spring Starters, permitiendo añadir funcionalidades habituales como desarrollo web, persistencia de datos o motores de plantillas de forma automática. Una vez configurados los parámetros del proyecto, la herramienta genera un archivo comprimido con la estructura básica de directorios, el archivo pom.xml y las dependencias iniciales necesarias para comenzar el desarrollo de la aplicación.

Anotaciones principales

Definición

En Spring Boot, las anotaciones son metadatos que se incorporan al código fuente mediante el símbolo @ con el objetivo de configurar y definir el comportamiento de los distintos componentes de la aplicación. Estas anotaciones permiten indicar al framework cómo debe gestionar clases, métodos o dependencias, facilitando tareas como la configuración de controladores, servicios, entidades o repositorios de forma declarativa.

El uso de anotaciones constituye una de las características principales del ecosistema Spring, ya que reduce considerablemente la necesidad de realizar configuraciones manuales mediante archivos XML, simplificando el desarrollo y mejorando la legibilidad del código.

Además, las anotaciones permiten el contenedor de Spring identificar automáticamente los componentes de la aplicación y administrar aspectos como la inyección de dependencias, el acceso a datos o la gestión de peticiones HTTP.

Ejemplos

Spring Boot utiliza un amplio conjunto de anotaciones que permiten configurar y estructurar la aplicación de forma declarativa. Algunas de las anotaciones más utilizadas en este proyecto son las siguientes:

- **Configuración principal**
  - **@SpringBootApplication:** se utiliza en la clase principal del proyecto y permite habilitar la autoconfiguración, el escaneo de componentes y la configuración de Spring Boot.
- **Gestión de componentes e inyección de dependencias**
  - **@Autowired:** permite realizar la inyección automática de dependencias entre componentes gestionadas por Spring.
  - **@Component:** define una clase genérica gestionada por el contenedor de Spring.
  - **@Service:** identifica las clases encargadas de implementar la lógica de negocio.
  - **@Repository:** indica que una clase o interfaz pertenece a la capa de acceso de datos.
- **Desarrollo web y controladores**
  - **@Controller:** define un controlador encargado de gestionar peticiones web y devolver vistas.
  - **@RestController:** define un controlador REST que devuelve directamente datos en formato JSON.
  - **@GetMapping** y **@PostMapping:** permiten asociar métodos a peticiones HTTP de tipo GET y POST respectivamente.

- **@RequestMapping:** se utiliza para definir rutas bases a asociar URLs a controladores y métodos.

- **Persistencia de datos**
  - **@Entity:** indica que una clase representa una entidad persistente asociada a una tabla de la base de datos.
  - **@Id:** define el identificador principal de una entidad.
  - **@OneToMany** y **@ManyToOne:** permiten establecer relaciones entre entidades de la base de datos.
- **Validación de datos**
  - **@Valid:** permite validar automáticamente objetos recibidos desde formularios o peticiones HTTP.
  - **@NotNull, @NotBlank** y **@Size:** se utilizan para validar restricciones sobre los atributos de las entidades o formularios.
- **Seguridad y autenticación**
  - **@EnableWebSecurity:** habilita la configuración de seguridad web mediante Spring Security.
  - **@PreAuthorize:** permite restringir el acceso a métodos según roles o permisos definidos.

Ventajas e inconvenientes

Ventajas

Una de las principales ventajas de Spring Boot es la simplificación del desarrollo de aplicaciones de aplicaciones Java mediante la autoconfiguración y la gestión automática de dependencias. Esto permite reducir considerablemente el tiempo necesario para iniciar y configurar proyectos.

Otra característica destacable es la incorporación de servidores embebidos como Apache Tomcat, que facilitan la ejecución y despliegue de aplicaciones sin necesidad de instalar servidores externos adicionales.

Spring Boot también proporciona una arquitectura organizada basada en capas, favoreciendo el mantenimiento, escalabilidad y reutilización del código. Además, su integración con tecnologías como Spring Data JPA, Hibernate o Spring Security permite desarrollar aplicaciones completas de manera estructurada.

Por último, su amplia adopción en entornos profesionales y la gran cantidad de documentación disponible convierten a Spring Boot en una tecnología consolidada y ampliamente utilizada en el desarrollo backend.

Inconvenientes

A pesar de sus ventajas, Spring Boot también presenta algunos inconvenientes. La gran cantidad de configuraciones automáticas puede dificultar la comprensión interna del framework para desarrolladores con poca experiencia, ocultando parte del funcionamiento real de la aplicación.

Además, al incorporar numerosas dependencias y configuraciones predefinidas, las aplicaciones pueden consumir más recursos que otras soluciones más ligeras, especialmente en proyectos pequeños.

Otro aspecto para considerar es que la elevada abstracción del framework puede complicar la personalización avanzada de ciertos componentes, siendo necesario conocer el funcionamiento interno de Spring para realizar configuraciones específicas.

Finalmente, la curva de aprendizaje inicial puede resultar elevada debido a la gran cantidad de módulos, anotaciones y tecnologías que forman parte del ecosistema Spring.

Justificación

La elección de Spring Boot para el desarrollo de este proyecto se basa principalmente en su capacidad para simplificar y agilizar la creación de aplicaciones web robustas en Java.

En primer lugar, Spring Boot permite reducir significativamente el tiempo de configuración inicial del proyecto gracias a su sistema de autoconfiguración y al uso de dependencias predefinidas (starters). Este resulta especialmente adecuado en el contexto de este TFG, donde se busca centrarse en la implementación de la lógica de negocio más que en la configuración del entorno.

Otro factor determinante en su integración con tecnologías del ecosistema Spring, como Spring MVC, Spring Data JPA o Hibernate, lo que facilita la implementación de una arquitectura por capas clara y bien estructurada, basada en controladores, servicios y repositorios.

Además, el hecho de incluir servidores embebidos como Apache Tomcat simplifica el despliegue de la aplicación, permitiendo su ejecución sin necesidad de configuraciones externas complejas. Esto mejora la portabilidad del proyecto y facilita su ejecución en distintos entornos.

Por último, su amplia utilización en el ámbito profesional convierte a Spring Boot en una tecnlogía relevante y alineada con las prácticas actuales de desarrollo backend, lo que aporta valor añadido al proyecto desde un punto de vista formativo y profesional.

En conjunto, estas características hacen que Spring Boot sea una elección adecuada para el desarrollo de una aplicación de gestión como la planteada en este TFG, al combinar facilidad de uso, escalabilidad y buenas prácticas de desarrollo.

Caso práctico: Análisis y Diseño

La aplicación de este proyecto es un CRUD de biblioteca escolar, es decir, un gestor de préstamos de libros a usuarios del centro, alumnos o profesores, mantenimiento del catálogo de libros, ejemplares de cada libro y usuarios de la biblioteca y, todo ello, separado en páginas web cada una con una funcionalidad específica y que convergen en un "dashboard" donde tendremos información resumida en forma de API REST.

Entornos de desarrollo

Para desarrollar correctamente una aplicación Java, debemos usar un servidor Tomcat dado que actúa como un contenedor de servlets y servidor web, traduciendo peticiones HTTP del navegador en código Java ejecutable y viceversa. Entre las características de Apache Tomcat podemos encontrar:

- **Contenedor de Servlets/JSP:** ejecuta componentes Java y páginas que generan contenido dinámico, algo que un servidor web estático no pude hacer por sí solo.
- **Gestión de peticiones HTTP:** Tomcat actúa como intermediario, recibiendo peticiones del cliente, procesándolas mediante Java y enviando la respuesta.
- **Gestión del Ciclo de vida:** administra automáticamente la creación, ejecución y destrucción de los servlets, liberando al desarrollador de esta tarea.
- **Servidor Web Integrado:** funciona como un servidor independiente para servir aplicaciones web, gestionando sesiones, seguridad y escalabilidad.
- **Entorno libero y abierto:** es gratuito, de código abierto y fácil de configurar, lo que lo hace ideal tanto para desarrollo local como para producción.

En este caso, al usar Spring Boot y tener en cuenta una de las ventajas que tiene frente a Spring Framework, tenemos un servidor Tomcat embebido en el código al instalar la dependencia Spring Web. Este servidor funciona iniciando automáticamente una instancia de servidor web dentro de la JVM al ejecutar la aplicación, eliminando la necesidad de instalar un servidor externo. Se incluye a través de la dependencia Spring Web, permitiendo empaquetar la aplicación como un archivo JAR ejecutable que contiene tanto el código como el servidor.

Este servidor tiene una configuración por defecto, que puede ser modificada, con el puerto 8080 preestablecido y maneja hasta 200 hilos de ejecución por defecto pudiendo ser modificados ambos valores. Se puede personalizar el servidor desde el archivo de application.properties o application.yml, o mediante clases de configuración java.

Aunque este entorno sea el escogido para le ejecución de este proyecto, no hay que descartar otras opciones como servidores Jetty o Undertow que también vienen embebidos con Spring Boot. También cabe recalcar que este servidor es perfectamente funcional para entornos de explotación.

IDE

El entorno de desarrollo integrado seleccionado para la realización de este proyecto es Apache NetBeans. La elección de este entorno se debe principalmente a su buena integración con el lenguaje Java y a las herramientas que ofrece para facilitar el desarrollo de aplicaciones web basadas en Spring Boot.

Apache NetBeans es un IDE gratuito y de código abierto que proporciona funcionalidades como edición de código, compilación, depuración, ejecución de programas y gestión de proyectos. Además, incluye herramientas que permiten mejorar la productividad durante el desarrollo, como autocompletado, detección de errores en tiempo real y administración de dependencias.

Otra de las razones por las que he escogido NetBeans es su facilidad de uso y el conocimiento previo adquirido durante el ciclo formativo, ya que ha sido el entorno utilizado en gran parte de las prácticas y proyectos realizados anteriormente. Esto permite reducir la curva de aprendizaje y centrar el trabajo en el desarrollo de la aplicación y en el aprendizaje del framework Spring Boot.

Asimismo, NetBeans ofrece compatibilidad con tecnologías utilizadas en el proyecto, como Java, HTML, CSS y JavaScript, permitiendo trabajar de manera centralizada sobre todos los componentes de la aplicación. También facilita la integración con sistemas gestores de bases de datos y herramientas de construcción como Maven.

Aunque existen otros entornos de desarrollo ampliamente utilizados, como Eclipse o IntelliJ IDEA, Apache NetBeans ha sido considerado una opción adecuada debido a su equilibrio entre simplicidad, funcionalidad y compatibilidad con las tecnologías empleadas.

Lenguajes de programación, lenguajes de marca utilizados

El lenguaje que usa Spring Boot es Java, lenguaje de programación orientado a objetos, fuertemente tipado y compilado, ampliamente utilizado en el desarrollo de aplicaciones empresariales y servicios web debido a su robustez, escalabilidad y compatibilidad multiplataforma.

Su utilización en el proyecto permite estructurar la aplicación siguiendo una arquitectura en capas y aplicar conceptos como encapsulación, reutilización de código y separación de responsabilidades. Además, la integración con Spring Boot facilita el desarrollo de aplicaciones web modernas mediante el uso de controladores, servicios y acceso a base de datos.

El siguiente lenguaje es JavaScript, orientado a objetos, débilmente tipado e interpretado por el lado cliente (navegador). En este proyecto se utiliza para añadir dinamismo e interactividad a la interfaz web. Su uso permite realizar validaciones, actualizar contenido de manera dinámica y mejorar la comunicación entre el frontend y el backend desarrollado por Spring Boot.

El siguiente lenguaje, en este caso de marca, es HTML, usado para definir la estructura y el contenido de las páginas web de la aplicación. Su función principal es organizar los distintos elementos visuales de la interfaz, como formularios, tablas, botones o menús de navegación.

Por último, CSS es el lenguaje encargado en definir el diseño y la apariencia visual de la aplicación. Definirá el estilo de los botones de formularios, campos, tablas, iconos, etc... También modificará la estructura visual de la página para mejorar la experiencia del trabajador.

Sistema de Gestión de Bases de Datos

Se ha escogido MySQL como sistema gestor de bases de datos relacional. La elección de esta tecnología se debe principalmente a su fiabilidad, facilidad de integración con aplicaciones Java y amplio uso en entorno profesionales y educativos.

MySQL es uno de los sistemas gestores de bases de datos más utilizados a nivel mundial en el desarrollo de aplicaciones web. Su popularidad se debe a factores como su estabilidad, buen rendimiento, sencillez de uso y compatibilidad con múltiples tecnologías y frameworks, entre ellos Spring Boot. Gracias a esta integración, es posible gestionar de forma eficiente las entidades principales del sistema, como usuarios, libros, ejemplares y préstamos.

El uso de una base de datos relacional resulta especialmente adecuado debido a la naturaleza estructurada de la información que se maneja. Las relaciones existentes entre distintas entidades de la aplicación requieren mantener integridad y coherencia en los datos. Debido a la existencia de estas relaciones entre entidades, es importante usar claves primarias y foráneas compatibles con MySQL, facilitando así el control y la organización de la información almacenada.

Otra de las razones por las que se ha escogido MySQL es su capacidad para trabajar correctamente con operaciones CRUD, que constituye la base funcional de la aplicación. El sistema permitirá registrar nuevos libros, modificar usuarios, gestionar préstamos y consultar información de manera rápida y eficiente.

Además, MySQL ofrece un entorno adecuado para proyectos escalables. Aunque la aplicación desarrollada en este TFG tiene un alcance académico y funcional relativamente sencillo, el modelo de datos planteado permite futuras ampliaciones. El uso de MySQL facilita dichas ampliaciones manteniendo una estructura sólida y organizada.

Por último, cabe destacar que MySQL dispone de una gran comunidad de soporte y abundante documentación técnica, lo que facilita el aprendizaje y resolución de problemas durante el desarrollo del proyecto. Asimismo, existen herramientas gráficas como MySQL Workbench que permite diseñar, visualizar y administrar la base de datos de forma intuitiva.

Entornos de explotación

El entorno de explotación seleccionado para este proyecto es Google Cloud, una plataforma de computación en la nube que permite desplegar aplicaciones y servicios accesibles desde Internet.

La utilización de un entorno cloud permite ejecutar la aplicación en un servidor remoto, facilitando el acceso desde distintos y simulando un escenario más cercano a un entorno real de producción. Este enfoque aporta ventajas relacionadas con la disponibilidad, accesibilidad y centralización de la aplicación.

Google Cloud proporciona infraestructura compatible con aplicaciones desarrolladas en Java y Spring Boot, ofreciendo distintas opciones de despliegue adaptadas a aplicaciones web y servicios backend. Además, su arquitectura escalable permite ampliar recursos y capacidades en función de las necesidades futuras del proyecto.

Otro aspecto relevante es la posibilidad de separar el entorno de desarrollo del entorno de explotación, permitiendo publicar la aplicación en un servidor independiente y accesible desde Internet. Este facilita la realización de pruebas externas y acerca el funcionamiento del sistema a un contexto profesional real.

La elección de Google Cloud se debe principalmente a su integración con tecnologías modernas de desarrollo, su facilidad de despliegue y la amplia utilización de plataformas cloud en el ámbito profesional. Asimismo, el uso de este tipo de servicios permite adquirir conocimientos relacionados con la administración y publicación de aplicaciones web en entornos remotos, aspecto especialmente en el desarrollo backend actual.

Framework, librerías...

Para el desarrollo de la aplicación se ha utilizado principalmente Spring Boot, junto con diversas dependencias y librerías integradas mediante Apache Maven. Estas herramientas permiten simplificar el desarrollo de aplicaciones web en Java y facilitar aspectos como la persistencia de datos, la seguridad, la validación de información o la integración con servicios externos.

La gestión de todas las dependencias utilizadas en el proyecto se realiza mediante el archivo pom.xml, donde se definen las librerías necesarias para el funcionamiento de la aplicación.

Dependencias principales utilizadas:

- **Spring Web:** permite desarrollar la capa web de la aplicación y gestionar las peticiones HTTP realizadas por los usuarios. Incluye soporte para controladores MVC, APIs REST y servidor Apache Tomcat embebido.
- **Spring Data JPA:** se utiliza para gestionar la persistencia de datos mediante entidades Java y repositorios. Facilita las operaciones CRUD y la interacción con la base de datos utilizando ORM.

- **Spring Security:** proporciona mecanismos de autenticación y autorización para proteger el acceso a distintas funcionalidades de la aplicación. Además, la dependencia "thymeleaf-extras-springsecurity6" permite integrar el sistema de seguridad directamente en las vistas desarrolladas con Thymeleaf.
- **Thymeleaf:** incorpora el motor de plantillas Thymeleaf, utilizado para generar las vistas HTML dinámicas de la aplicación y facilitar la interacción entre el backend y la interfaz web.
- **MySQL Connector:** permite establecer la conexión entre la aplicación y la base de datos MySQL utilizada en el proyecto.
- **Validación de datos:** permite realizar validaciones automáticas sobre formularios y entidades, garantizando la integridad de los datos introducidos por el usuario.
- **Herramientas de desarrollo y pruebas:** facilita tareas de desarrollo como la recarga automática de la aplicación durante la programación. Por otro lado, están las dependencias de test (starter y security) que proporcionan herramientas para la realización de pruebas unitarias y pruebas relacionadas con la seguridad de la aplicación.
- **Integración con cloud:** "spring-cloud-gcp-starter-sql-mysql" permite integrar la aplicación con servicios de Google Cloud, facilitando la conexión y gestión de bases de datos MySQL desplegadas en entornos cloud. Además, el bloque "spring-cloud-gcp-dependencies" se utiliza para implementar de forma centralizadas las versiones compatibles de las librerías relacionadas con Google Cloud.

Análisis y diseño

Catálogo de requisitos

Glosario

A continuación, se listan todos los términos principales de mi aplicación de gestor de bibliotecas escolares. Dichos términos pertenecen, principalmente, a las entidades de la aplicación y los diferentes tipos de usuarios.

- **Usuario no registrado:** no necesita identificarse para utilizar la aplicación web. Está restringido a un pequeño apartado de nuestra aplicación, es decir, tiene muchas limitaciones.
- **Perfil registrado:** necesita tener unas credenciales guardades en base de datos e identificarse con ellas. Dichas credenciales son nombre de usuario y contraseña. Este usuario no tiene limitaciones para realizar cualquier acción en la aplicación ya que está asociado a la trabajadora de la biblioteca.
- **Perfil administrador:** puede crear, editar y eliminar perfiles trabajadores. También puede realizar el resto de las acciones de la aplicación. Puede transformar a un perfil trabajador en perfil administrador.
- **Perfil trabajador:** puede realizar todas las acciones principales de la aplicación:
    - Mantenimiento de usuarios: crear, editar o suspender (baja lógica) a usuarios del centro (alumnos y profesores).

- Mantenimiento de libros: crear, editar o suspender (baja lógica) libros del catálogo de la biblioteca. A mayores, podrá añadir o eliminar ejemplares de cada libro.
- Mantenimiento de préstamos: crear, editar o finalizar préstamos realizados.

- **Usuario (alumno/profesor):** es el alumno o profesor perteneciente el centro educativo que se da de alta en el sistema de biblioteca. No sirven como cuentas de la aplicación, solamente son entidades de lógica de negocio.
- **Libro:** manuscrito encuadernado que será solicitado por el usuario para su lectura.
- **Ejemplar:** copia de libro el cual será prestado cuando un usuario solicite un libro. Estos ejemplares tienen un estado definido según el momento: disponible, ocupado, etc...
- **Préstamo:** acción principal de la aplicación en la cual el usuario solicita un libro, se le entrega un ejemplar disponible de dicho libro y se le presta hasta una fecha máxima. Los estados tienen estados según su situación: activo, finalizado, retraso, etc...

Requisitos del sistema

A continuación, se listan los requisitos de la aplicación según el tipo de usuario que esté usando la aplicación.

1. **Web pública:**
   - 1.1. Los usuarios no registrados podrán consultar la página inicial de la aplicación.
   - 1.2. Los usuarios no registrados podrán consultar la API de la aplicación web.
   - 1.3. Los usuarios no registrados podrán acceder a la documentación de la aplicación.
   - 1.4. Los usuarios no registrados podrán iniciar sesión introduciendo sus credenciales.
     - 1.4.1. Dicho inicio de sesión tendrá medidas de seguridad para evitar ataques por inyección de código. En este caso, se limitará el número máximo de intentos de inicio de sesión a tres. En caso de superar dicho límite, se bloqueará el inicio de sesión durante 10 minutos.
   - 1.5. Los usuarios registrados podrán cerrar sesión.
   - 1.6. Los usuarios registrados podrán ser trabajadores o administradores.
   - 1.7. Los usuarios registrados podrán editar su contraseña y datos personales. No podrán modificar el nombre de usuario, el dni ni el rol.
2. **Mantenimiento de usuarios:**
   - 2.1. Los usuarios registrados podrán realizar el mantenimiento de la información sobre los usuarios del centro (alumnos y profesores).
   - 2.2. El sistema almacenará la siguiente información de cada departamento:
     - 2.2.1. **IdUsuario**: número identificativo del usuario el cual estará compuesto por la primera letra del nombre del usuario, las tres primeras letras del primer apellido y las tres primeras letras del segundo apellido.
     - 2.2.2. **NombreUsuario**: nombre del alumno o profesor escrito con la primera letra en mayúscula y con los signos de puntuación correctos.
     - 2.2.3. **Apellido1Usuario:** primer apellido del alumno o profesor escrito con la primera letra en mayúscula y con los signos de puntuación correctos.

2.2.4. **Apellido2Usuario:** segundo apellido del alumno o profesor escrito con la primera letra en mayúscula y con los signos de puntuación correctos.

2.2.5. **TipoUsuario:** tipo de usuario a elegir entre alumno o profesor. Los profesores tendrán beneficios en los préstamos.

2.2.6. **EstadoUsuario:** define el estado de un usuario. Aquel usuario inactivo no podrá realizar ningún préstamo.

2.3. El sistema permitirá la creación de un nuevo usuario. **Registrar usuario.**

2.4. El sistema permitirá la modificación de datos asociados al departamento salvo el código identificativo del usuario. **Modificar usuario.**

2.5. El sistema permitirá el borrado lógico de un usuario que quiera darse de baja. **Baja de usuario.**

2.6. El sistema permitirá la rehabilitación lógica de un usuario que quiera darse de alta o el trabajador haya cometido un fallo. **Alta de usuario**.

2.7. El sistema permitirá la consulta de los datos de un usuario junto con su historial de préstamos resumido. Se podrá ver información de préstamos, la cantidad que hay en activo, etc...

2.8. El sistema permitirá la paginación y búsqueda compleja de usuarios por nombre y dni y filtrado por activos.

3. **Mantenimiento de libros:**

3.1. Los usuarios registrados podrán realizar el mantenimiento de la información sobre los libros del catálogo de la biblioteca del centro.

3.2. El sistema almacenará la siguiente información de cada libro:

3.2.1. **IdLibro:** número identificativo del libro el cual estará compuesto por el tejuelo. Esto es: un número identificativo que clasifica los libros en la biblitoca según su tipo, las tres primeras letras del primer apellido del autor y las tres primeras letras del nombre del libro.

3.2.2. **NombreLibro:** nombre del libro. Se guardará el nombre completo en el mismo campo.

3.2.3. **AutorLibro:** autor del libro. Se guardará el nombre y los apellidos en el mismo campo.

3.2.4. **GeneroLibro:** el género que define el contenido del libro. Se podrá elegir entre varias opciones: novela, poesía, teatro, ensayo, etc...

3.2.5. **ISBN:** el código identificativo de cada libro el cual no podrá ser duplicado. Dicho código también está validado.

3.2.6. **EstadoLibro:** define el estado de un libro. Aquel libro inactivo no podrá ser prestado, ni modificar el número de ejemplares.

3.3. El sistema permitirá la creación de un nuevo libro. **Añadir libro.**

3.4. El sistema permitirá la modificación de los datos asociados al libro salvo el código de libro. **Modificar departamento.**

3.5. El sistema permitirá el borrado lógico de un libro cuando éste deje de estar disponible en el catálogo de la biblioteca. **Baja libro.**

3.6. El sistema permitirá la rehabilitación lógica de un libro cuando éste vuelva a estar disponible en el catálogo de la biblioteca. **Rehabilitar libro.**

3.7. El sistema permitirá la consulta de la información de un libro junto al número de ejemplares que tiene y su estado actual.

3.8. El sistema permitirá añadir ejemplares a cada libro asignando un código automático a cada registro. **Añadir ejemplar.**

3.9. El sistema permitirá el borrado lógico de un libro cuando éste deje de estar disponible por su estado deplorable. **Baja ejemplar.**

3.10. El sistema permitirá el paginado de libros junto a una búsqueda múltiple de título, autor o isbn y filtrado por estado de libro.

4. **Mantenimiento de préstamos:**

4.1. Los usuarios registrados podrán realizar el mantenimiento de la información sobre los préstamos realizados en la biblioteca.

4.2. El sistema almacenará la siguiente información de cada préstamo:

4.2.1. **IdPrestamo:** número identificativo del préstamo el cual será auto incremental.

4.2.2. **IdEjemplar:** número identificativo del ejemplar.

4.2.3. **IdUsuario:** número identificativo del usuario.

4.2.4. **FechaInicio:** fecha y hora en la que comienza el préstamo.

4.2.5. **FechaFin:** fecha y hora límite la cual define el plazo de devolución. En caso de superar dicho límite se tendrá controlado y se especificará. Dicho plazo será establecido automáticamente dando como límite 5 días hábiles para devolverlo.

4.2.6. **FechaDevolucion:** fecha y hora en la que finaliza el préstamo.

4.2.7. **ObservacionesPrestamo:** observaciones/anotaciones realizadas por el trabajador acerca del préstamo.

4.3. El sistema permitirá la creación de un nuevo préstamo. Se podrá registrar un préstamo si el ejemplar escogido está disponible. No se podrá prestar un ejemplar a un usuario que ya tenga cinco préstamos en activo. **Registrar préstamo.**

4.4. El sistema permitirá la modificación de los datos asociados al préstamo salvo el código identificativo de préstamo. En caso de intentar modificar el ejemplar prestado por otro, solo se podrá cambiar si el nuevo ejemplar está disponible. No se podrá prestar un ejemplar a un usuario que tenga ya cinco préstamos en activo. **Modificar préstamo.**

4.5. El sistema permitirá finalizar un préstamo cuando el usuario devuelve el ejemplar al trabajador. Deberá introducir el código del ejemplar para que la devolución sea correcta. **Finalizar préstamo.**

4.6. El sistema permitirá consultar la información de un préstamo: usuario, libro, fecha y estado. **Consultar préstamo.**

4.7. El sistema permitirá buscar y filtrar los préstamos según su estado. **Buscar préstamo.**

4.8. El sistema paginará los préstamos en grupos de cinco en cinco préstamos. Habrá botones para moverse por las diferentes páginas, número de la página actual y número total de páginas. **Paginación préstamos.**

5. **Perfil**

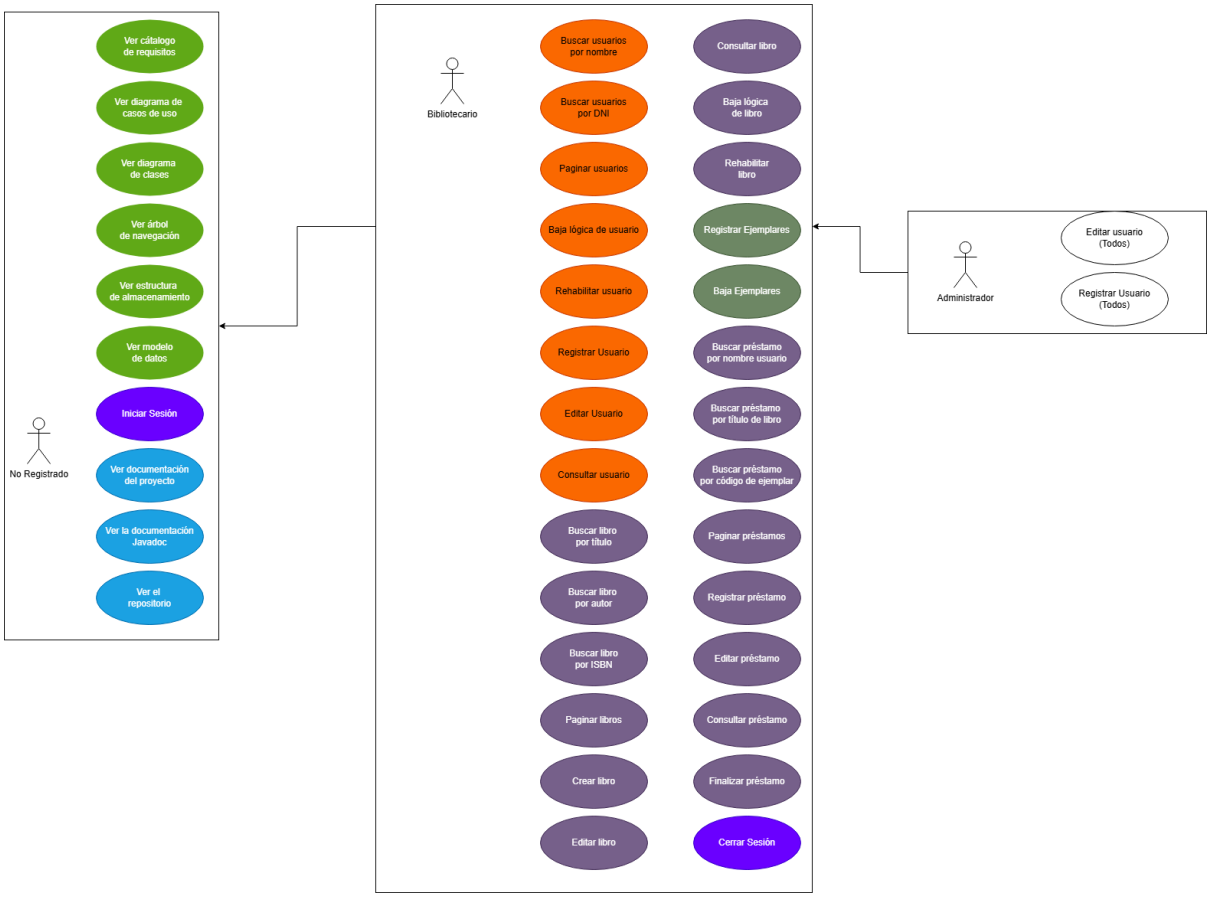
5.1. El sistema permitirá la edición de los datos personales del trabajador con la sesión iniciada. Se podrá cambiar:

5.1.1. **Nombre y apellidos**

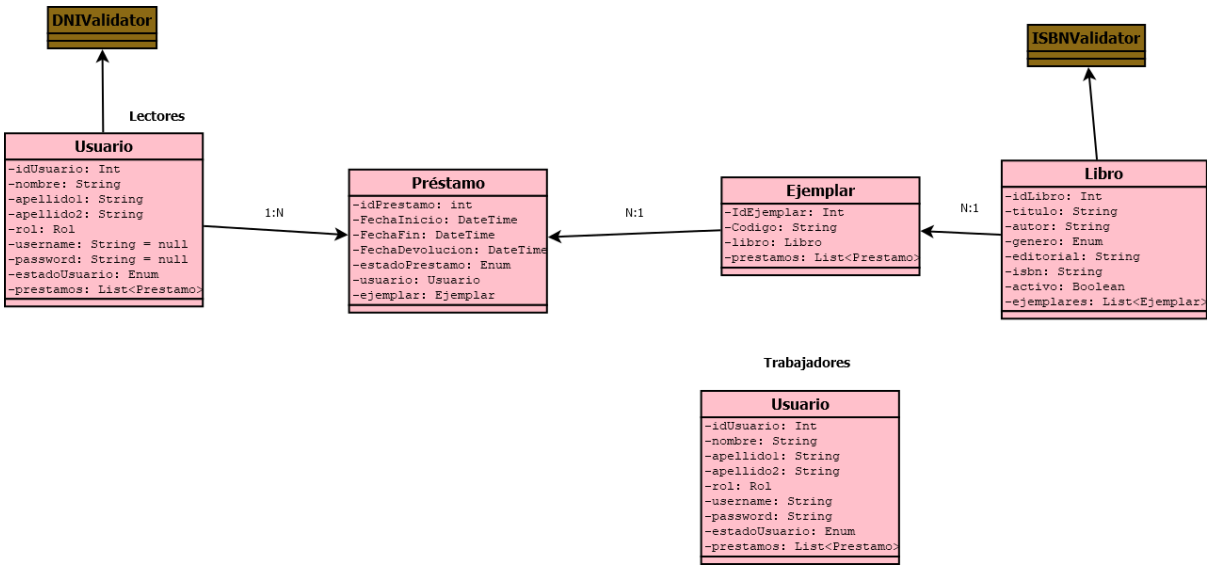
5.1.2. **Contraseña**

5.2. El sistema obligará repetir la contraseña para asegurar su correcto cambio.

Diagramas de casos de uso



Diagramas de clases



Estas son las entidades principales de la aplicación: usuarios, entidad fuerte asociada a un validador de DNIs, libro, entidad fuerte asociada a un validador de ISBN y el cual debe ser creado por un usuario bibliotecario o administrador, un ejemplar, entidad débil dependiente de libro y préstamo entidad asociativa dependiente de ejemplar y usuario. El motivo por el cual sale repetida la entidad de usuario es porque hay que diferenciar entre dos grupos de usuarios: los lectores que realizan préstamos, es decir, son los clientes del sistema, y los trabajadores que son los que operan en la aplicación y se autentican mediante usuario y contraseña.

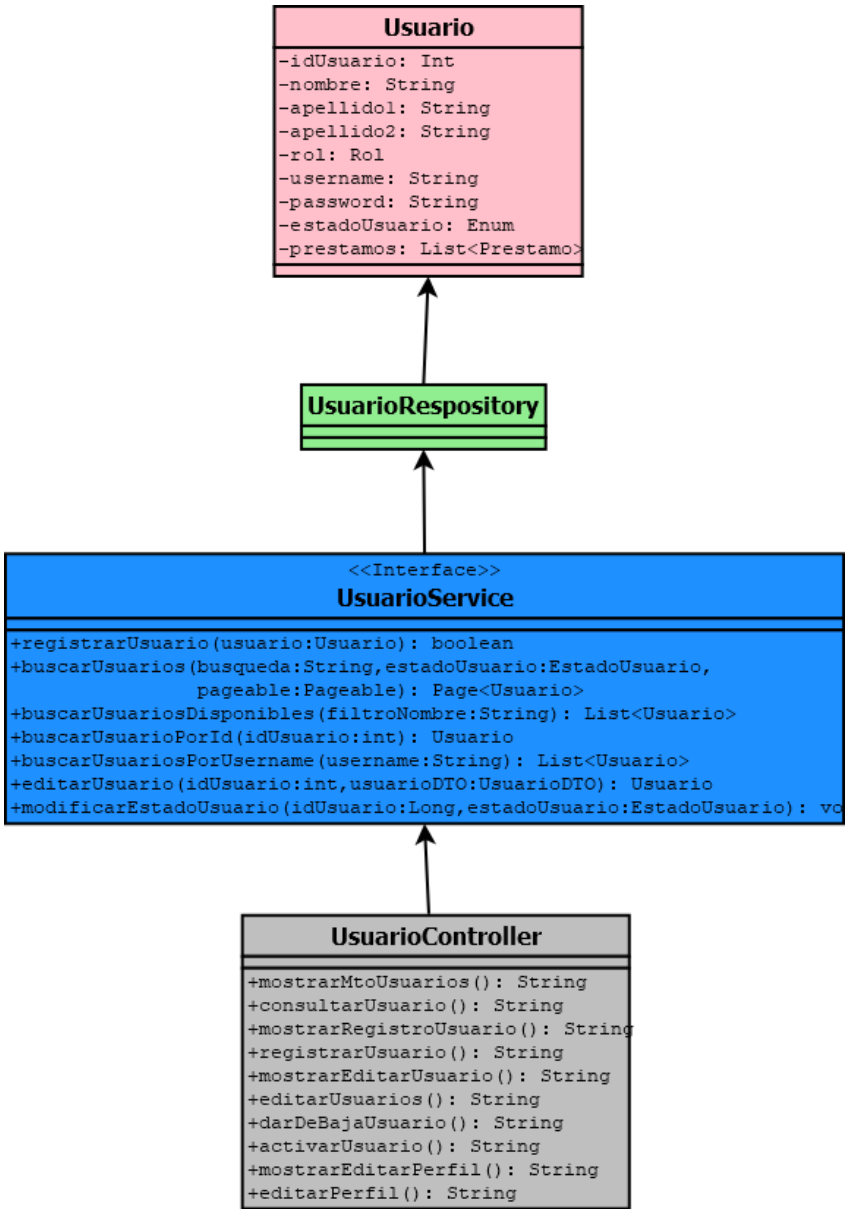


Diagrama de la clase Usuario con la conexión a la base de datos (Repository), lógica de negocio y controlador de la vista.

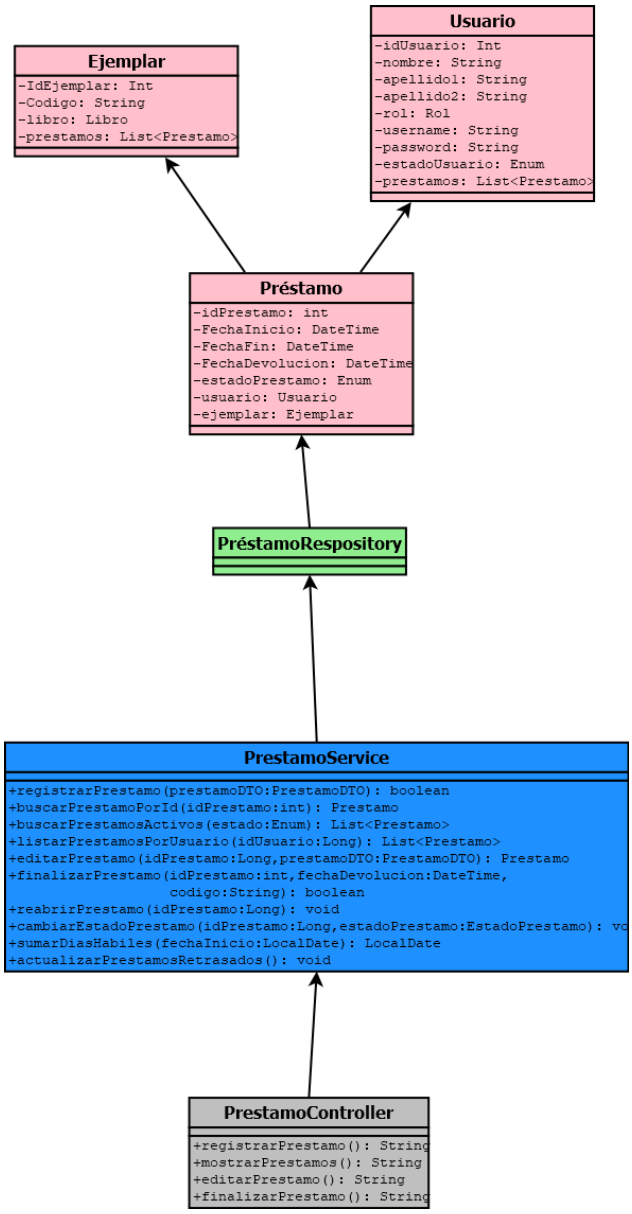


Diagrama de la clase préstamo relacionado con Usuarios y Ejemplares. Está conectado a la base de datos (Repository), a la lógica de negocio y controlador de la vista.

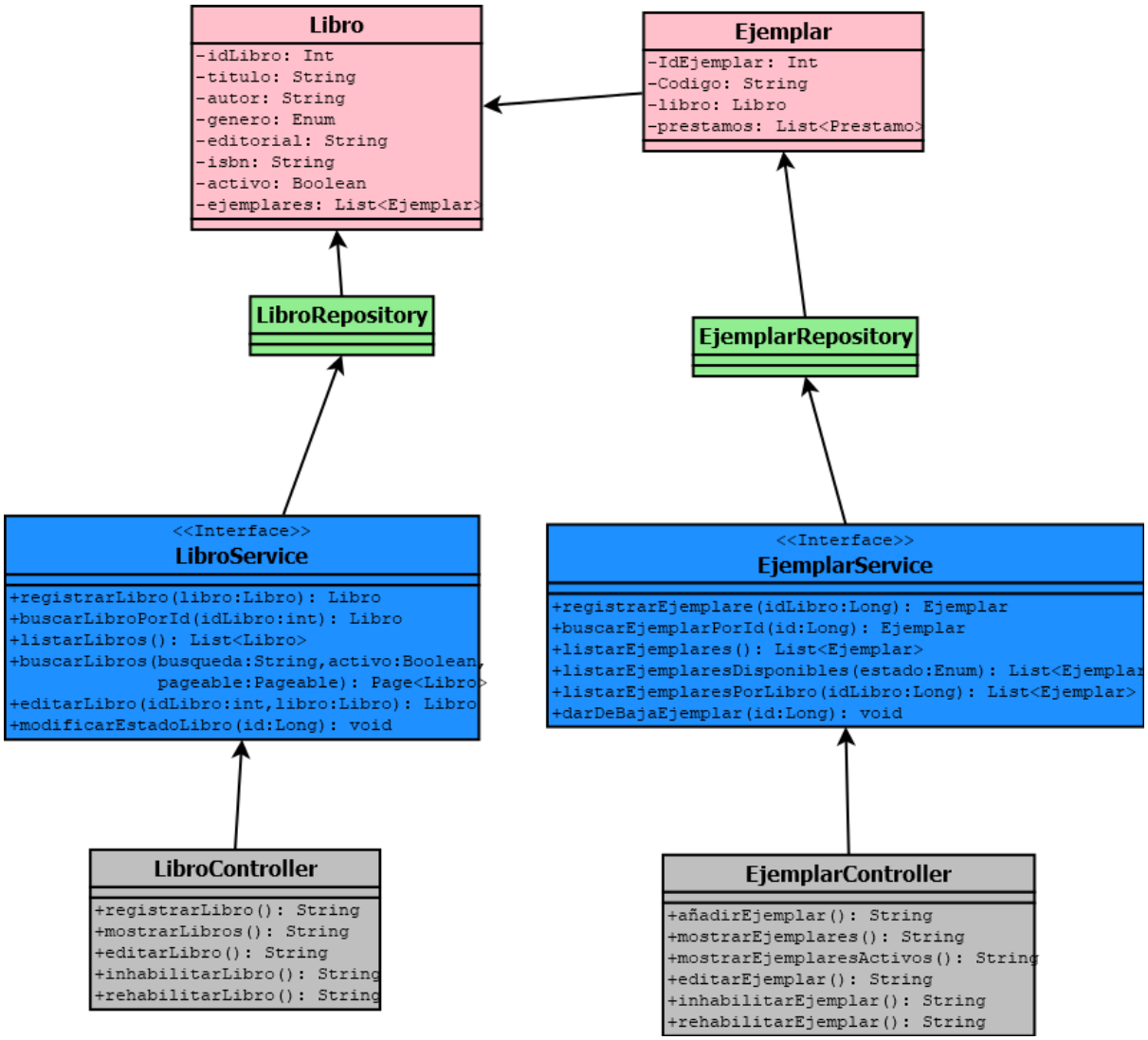


Diagrama de clases Libro y Ejemplar con su relación NEjemplares:1Libro. Están conectadas a la base de datos con el repository, a la lógica de negocio y al controlador de la vista.

Modelo físico de datos

El siguiente diagrama entidad-relación representa la estructura lógica de la base de datos diseñada para el sistema de gestión de biblioteca. En él se definen las entidades principales del sistema, sus atributos y las relaciones existentes entre ellas. El objetivo de este modelo es garantizar una organización eficiente de la información relacionada con los libros, los ejemplares disponibles, los usuarios registrados y los préstamos realizados.

Entidad libro

La entidad libro almacena la información bibliográfica general de cada obra disponible en la biblioteca. Cada libro se identifica de manera única mediante el atributo IdLibro, que actúa como clave primaria. Además, contiene los siguientes atributos:

- **Título**: nombre de la obra.
- **Autor:** autor o autores del libro.
- **Género:** ENUM que permite clasificar los libros según su categoría literaria.
- **Editorial:** editorial responsable de la publicación.
- ISBN: código internacional normalizado del libro, definido como único para evitar duplicidades.

En el modelo, la relación "Tiene" conecta la entidad "Libro" con la entidad "Ejemplar". Esta relación indica que un libro puede tener cero o muchos ejemplares asociados, mientras que cada ejemplar pertenece obligatoriamente a un único libro. La cardinalidad representada es, por tanto:

- Un libro (0,N) Ejemplar/es
- Un ejemplar (1,1) Libro

Entidad ejemplar

La entidad "Ejemplar" representada cada copia física disponible de un libro dentro de la biblioteca. Aunque varios ejemplares pueden pertenecer al mismo libro, cada uno debe identificarse individualmente. Sus atributos son:

- **IdEjemplar:** identificador único del ejemplar.
- **IdLibro:** clave foránea que referencia el libro al que pertenece.
- **Código:** identificador compuesto formado por el ISBN del libro y el identificador del ejemplar.

Esta separación entre Libro y Ejemplar permite gestionar correctamente múltiples copias de una misma obra, controlar préstamos individuales y mantener el inventario actualizado.

Entidad Usuario

La entidad "Usuario" almacena la información de las personas registradas ene l sistema y autorizadas para realizar préstamos. Dichas personas están divididas en roles como alumno o profesor que pueden solicitar un préstamo y bibliotecario y administrador que son los verdaderos actores de la aplicación y realizan todas las funciones. Estos últimos tienen un username y un password que les identifica para acceder al sistema. Los atributos de usuario son:

- **IdUsuario:** clave primaria que identifica al usuario.
- **DNI:** documento nacional de identidad del usuario.
- **Nombre:** nombre del usuario.
- **Apellido1 y Apellido2:** primer y segundo apellido del usuario.
- **Username:** nombre del usuario con el que se inicia sesión.
- **Contraseña:** clave cifrada para la autenticación.
- **Rol:** determina el nivel de permisos del usuario dando acceso exclusivo a bibliotecarios y al administrador.
- **Estado:** determina la situación del usuario (activo, suspendido o castigado y de baja).

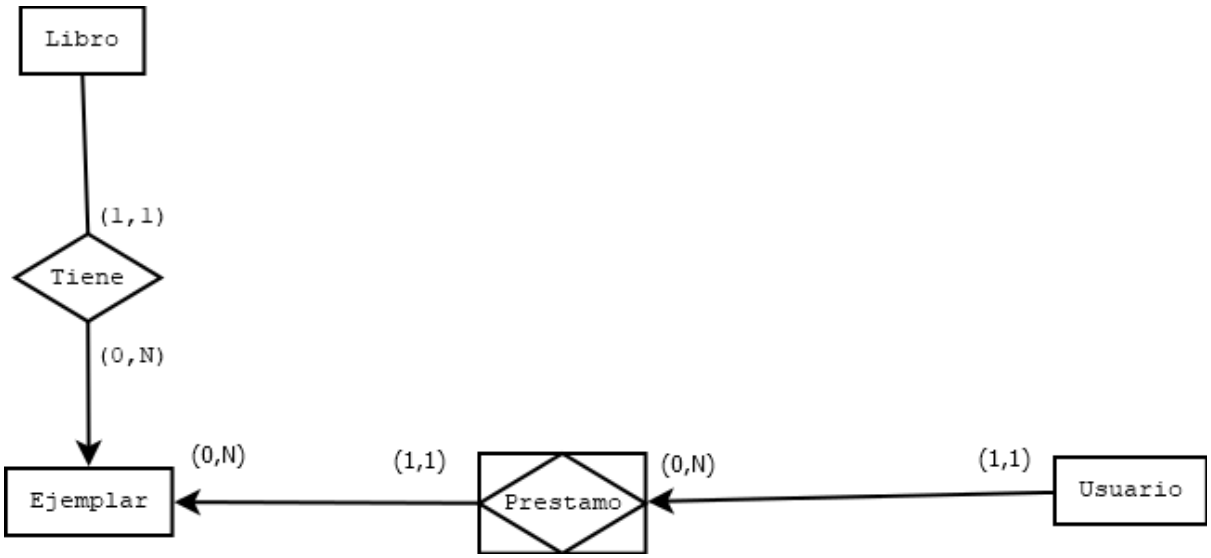
Entidad préstamo

La entidad relacional “Préstamo” gestiona las operaciones de préstamo de ejemplares realizadas por los usuarios. Esta entidad actúa como intermediaria entre “Usuario” y “Ejemplar”, permitiendo registrar el historial de préstamos y devoluciones. Sus atributos son:

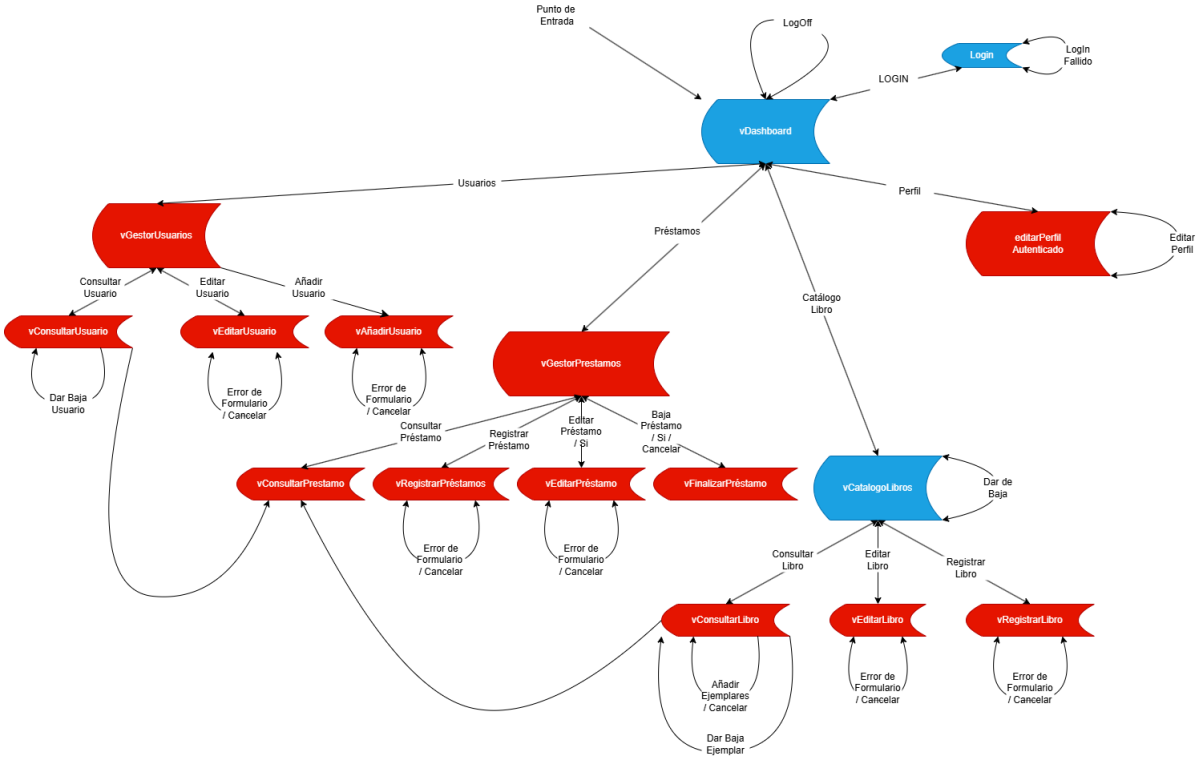
- **IdPrestamo**: identificador único del préstamo.
- **IdUsuario**: clave foránea que referencia al usuario que realiza el préstamo.
- **IdEjemplar**: clave foránea que referencia al ejemplar prestado.
- **fechaInicio**: fecha en la que se realiza el préstamo.
- **fechaFin**: fecha límite de devolución.
- **fechaDevolucion**: fecha real en la que el ejemplar es devuelto.
- **estado**: atributo *ENUM* que indica la situación del préstamo (activo, retrasado, finalizado, cancelado, etc.).

Las relaciones representadas en el diagrama indican que:

- Un usuario (0,N) préstamo/s
- Un préstamo (1,1) usuario
- Un ejemplar (0,N) préstamo/s
- Un préstamo (1,1) ejemplar



Árbol de navegación



Estructura de almacenamiento

GestorBiblioteca/

.git/

.mvn/

src/

target/

.gitattributes

.gitignore

HELP.md

README.md

Mvnw

Mvnw.cmd

Nbaction.xml

Nb-configuration.xml

Pom.xml

Web Services – API

El web service desarrollado para esta aplicación consiste en la obtención de datos del sistema de carácter informativo tales como la cantidad de libros, préstamos y usuarios activos hay en todo momento. Dicho servicio está disponible en la página inicial y es accesible para todo tipo de usuarios y para usuarios no registrados.

La API REST desarrollada solicita a la base de datos mediante sentencias SQL la cantidad de libros activos y con ejemplares que hay en la aplicación, la cantidad de usuarios de tipo profesor y alumnos activos que hay y la cantidad de préstamos activos (no devueltos) que hay. Dicha información llega al controlador REST de Spring Boot y transforma los datos en un JSON que es llevado a la vista para su exposición.

Presupuesto y financiación

Para estimar el coste de desarrollo de la aplicación se ha realizado una valoración aproximada de las horas dedicadas a las distintas fases del proyecto, considerando un coste medio de 15€/hora correspondiente a un perfil junior de desarrollo de software.

A continuación, se muestran los costes desglosados por apartados:

- Análisis y diseño: 20 horas totales -> 300€
- Desarrollo de la aplicación: 60 horas totales -> 900€
- Pruebas y validación: 10 horas totales -> 150€
- Documentación: 20 horas totales -> 300€
- Total estimado: 1650€

Las herramientas utilizadas duranta el desarrollo, como Java, Spring Boot, Maven, MySQL y NetBeans, son gratuitos o de código abierto, por lo que no han supuesto costes adicionales. Del mismo modo, el despliegue de la aplicación previsto en Google Cloud puede realizarse mediante recursos gratuitos temporales o créditos promocionales. Sin embargo, suponiendo que la aplicación está pensada para uso profesional realista podemos considerar un coste mensual de 30 euros al mes por alojamiento pudiendo ser superior según las exigencias.

El proyecto ha sido desarrollado íntegramente por el autor como parte del Trabajo de Fin de Grado, sin financiación ni participación de empresas colaboradoras. Todos los recursos utilizados han sido aportados por el propio estudiante o mediante herramientas de uso gratuito para fines educativos.

Caso Práctico: Implementación

Explicación del trabajo realizado

El desarrollo del proyecto ha seguido todas las fases habituales de creación de una aplicación informática, comenzando por el análisis de requisitos y el diseño de la solución propuesta. A partir de esta fase inicial se definió la estructura de la base de datos, la arquitectura de la aplicación y las funcionalidades necesarias para la gestión de la biblioteca escolar.

Posteriormente se llevó a cabo la implementación utilizando Java y Spring Boot, desarrollando los distintos módulos encargados de la gestión de usuarios, libros, ejemplares y préstamos. También se incorporaron mecanismos de validación de datos, control de acceso mediante autenticación y diferentes funcionalidades destinadas a facilitar la administración del sistema.

Una vez completado el desarrollo, se realizaron pruebas funcionales para verificar el correcto funcionamiento de la aplicación y se preparó la documentación técnica y de usuario necesaria para su mantenimiento y reutilización.

Repositorio de software

El código fuente del proyecto se encuentra almacenado en un repositorio GitHub, utilizado como sistema de control de versiones durante todo el proceso de desarrollo. El uso de esta plataforma ha permitido mantener un historial completo de cambios, gestionar distintas versiones de la aplicación y disponer de una copia de seguridad permanente del código desarrollado.

Además de facilitar el seguimiento de la evolución del proyecto, GitHub permite compartir el código de forma sencilla y garantiza la disponibilidad del repositorio para futuras tareas de mantenimiento, ampliación o consulta de la implementación realizada.

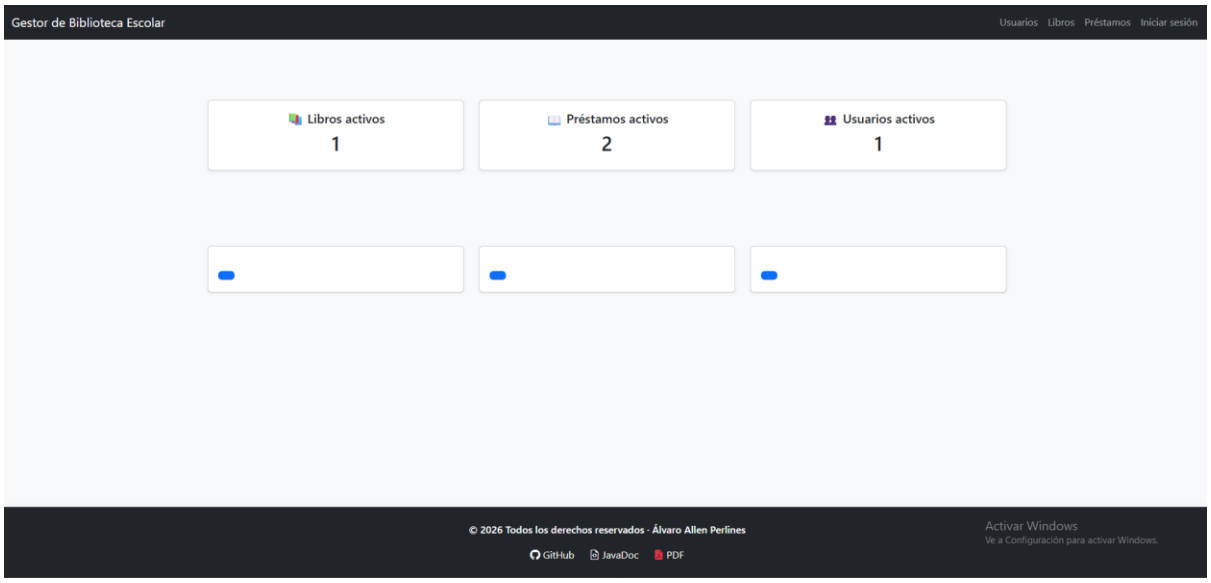
Pulsando aquí se puede acceder al repositorio

Documentación

Manual de usuario

Guía oficial de la aplicación Gestor de Biblioteca Escolar.

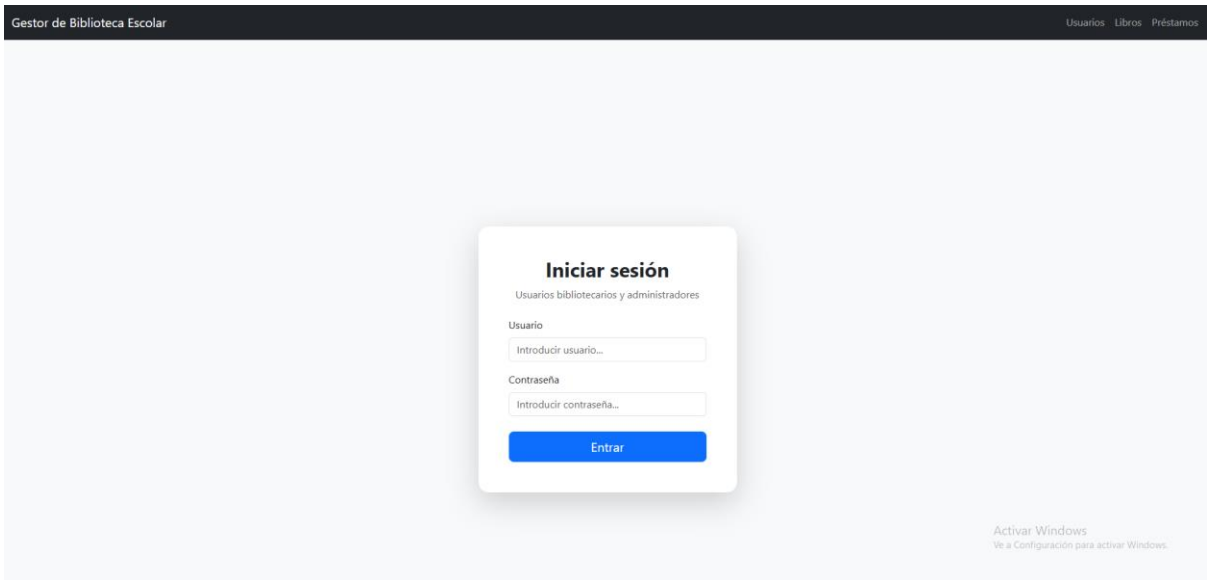
Inicio



En la ventana de inicio se encuentra una cabecera con los distintos apartados principales de la aplicación: Usuarios, Libros, Préstamos e Iniciar Sesión. A mayores, texto de Gestor de Biblioteca Escolar es un enlace que nos redirige siempre a esta página.

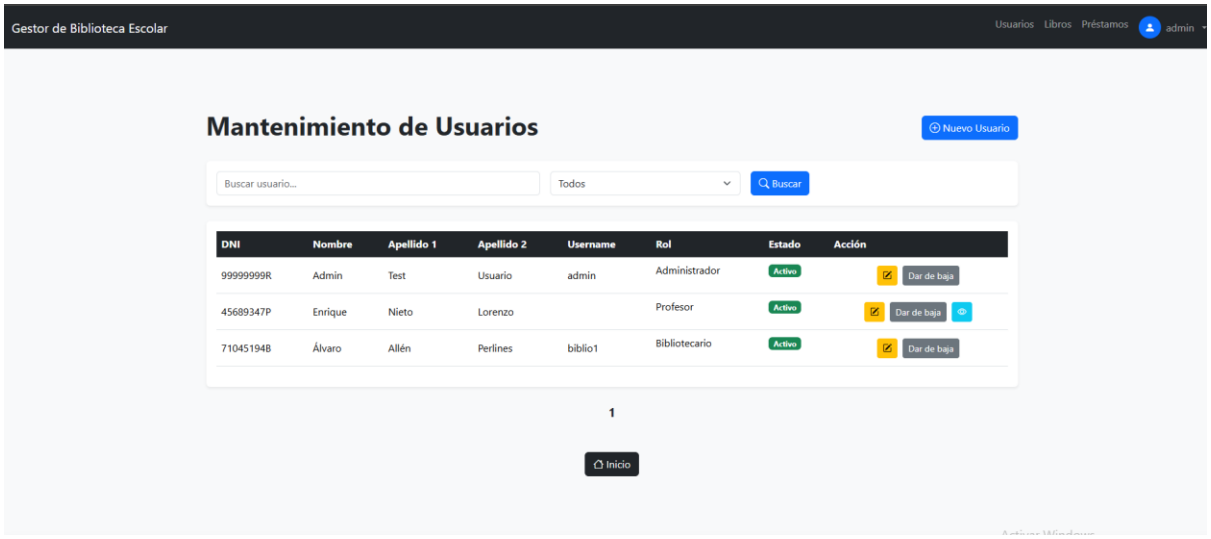
El contenido principal de la página está compuesto por una API propia que muestra las cifras de los libros activos, los prestamos activos y no devueltos y los usuarios que están asociados a la biblioteca y son activos. Dichas cifras están actualizadas en todo momento. Inmediatamente debajo podemos encontrar la documentación teórica de la aplicación: diagramas de casos de uso, modelo de datos, requisitos mínimos, etc... Por último, en el pie de página podemos encontrar el nombre del autor y tres enlaces diferentes referenciados a: al repositorio en GitHub, la documentación generada por JavaDoc y el pdf de la documentación oficial del TFG.

Para acceder a la zona restringida deberemos iniciar sesión con una cuenta de usuario bibliotecario o administrador, completando el formulario de login con el username y la password. Si se intenta acceder a la zona restringida sin autenticarse, se redirigirá automáticamente al login. En caso de introducir datos incorrectos se mostrará un mensaje de error especificando el motivo.



Este es el formulario de login en el cual se deben introducir las credenciales de usuarios autorizados.

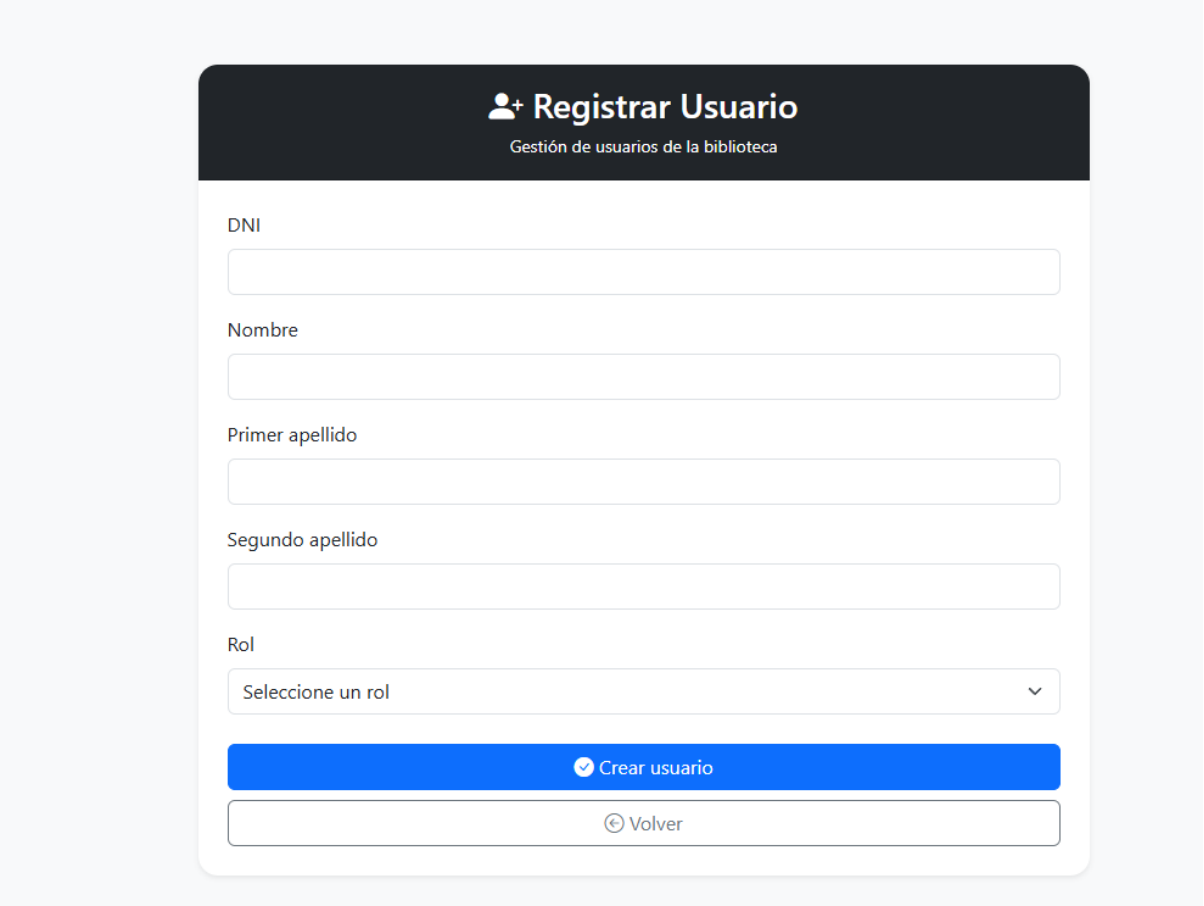
Usuarios



Este es el mantenimiento de usuarios donde se encuentran y gestionan todos los usuarios del sistema. El contenido principal de la página es una tabla donde se listan los usuarios de forma paginada y de cinco en cinco. Dicho listado se puede modificar y buscar de forma personalizada usando el buscador con filtro que encontramos encima de la tabla. Dicho buscador diferencia DNI y nombre, y filtra por el estado de los usuarios siendo, activos, baja, suspendidos y todos.

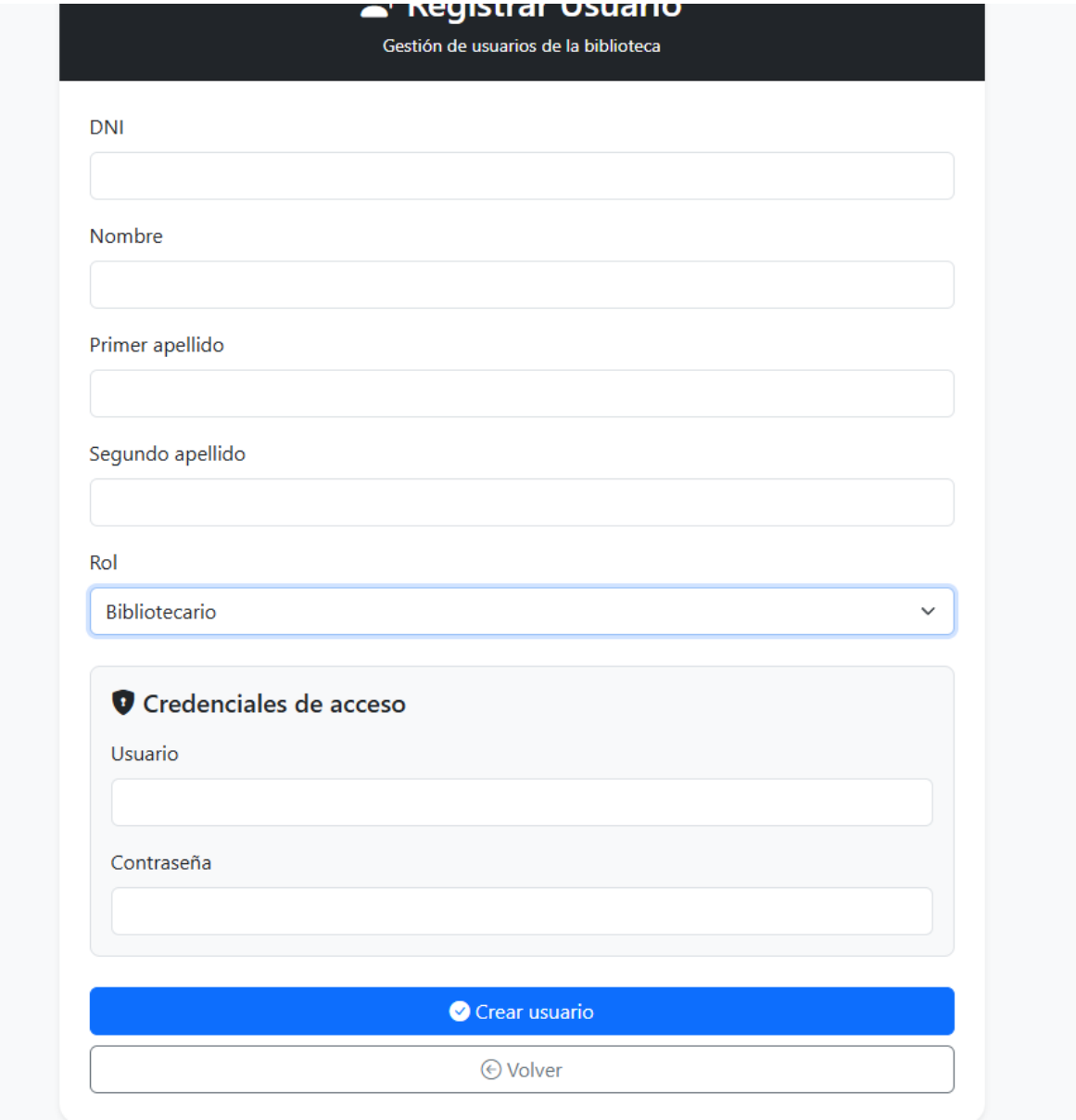
Por encima del filtro tenemos un botón en el que pone “Nuevo Usuario”. Es el encargado abrir el formulario de registro de usuario.

Registro de Usuario



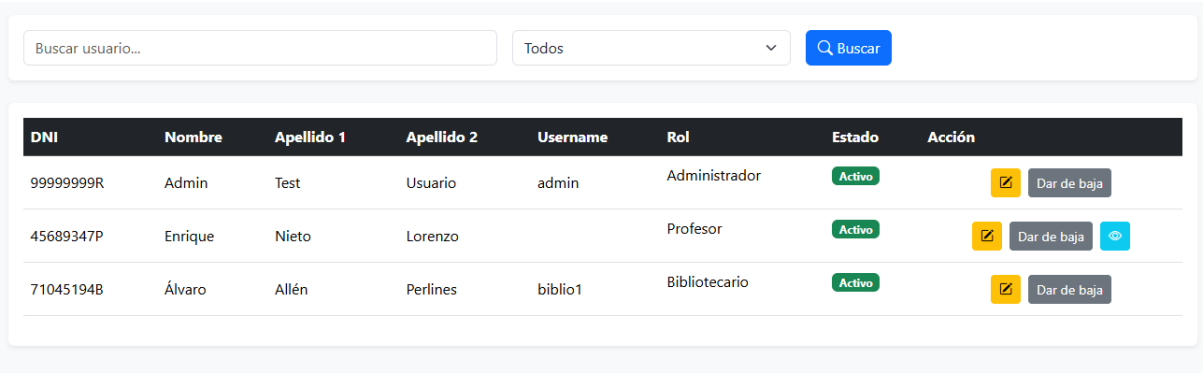
Este es el formulario de registro de usuario en el cual deberemos rellenar los siguientes campos:

- DNI el cual es clave única útil para buscar a un usuario mediante y validado para que cumpla el patrón establecido de DNI español.
- Nombre del usuario.
- Primer apellido del usuario.
- Segundo apellido del usuario.
- Rol para escoger entre alumno, profesor, bibliotecario o administrador.
    - En caso de escoger o bibliotecario o administrador se desplegarán dos apartados más: username y password para establecer unas credenciales de acceso.



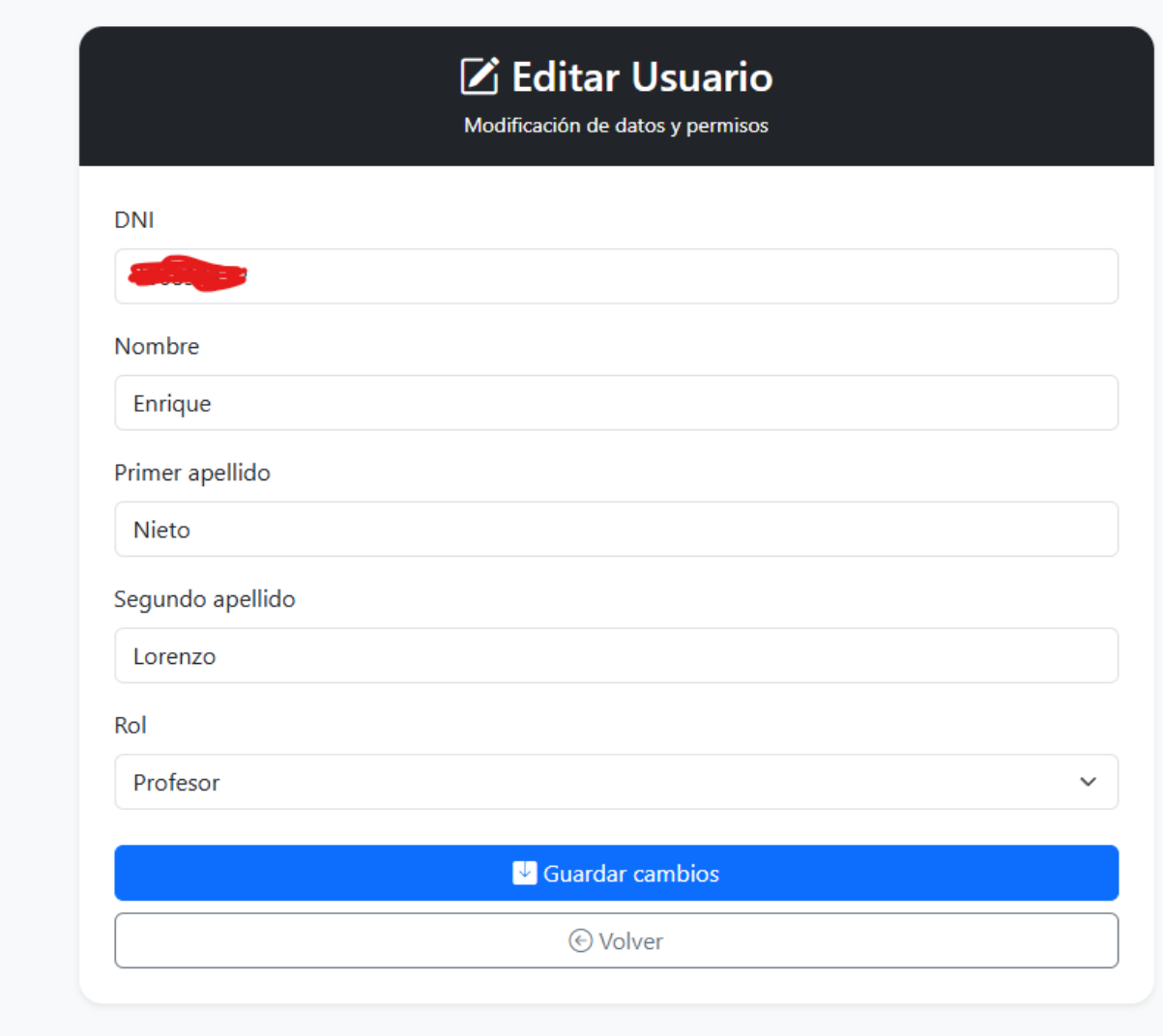
En caso de haber introducido algún dato incorrecto el formulario nos avisará mediante mensajes del error o errores concretos.

Tabla de usuarios



Cada fila representa a un usuario guardado en la base de datos y cada columna muestra un dato distinto siendo DNI, Nombre, Primer Apellido, Segundo Apellido, Username (en caso de tenerlo), Rol y Estado. La última columna está reservada para las acciones que se pueden realizar sobre el usuario. Dichas acciones son: Editar y Dar de baja las cuales son funcionales para todos los usuarios y consultar que solo está disponible para los usuarios de tipo Profesor y Alumno ya que son los únicos que pueden realizar préstamos.

Editar usuarios

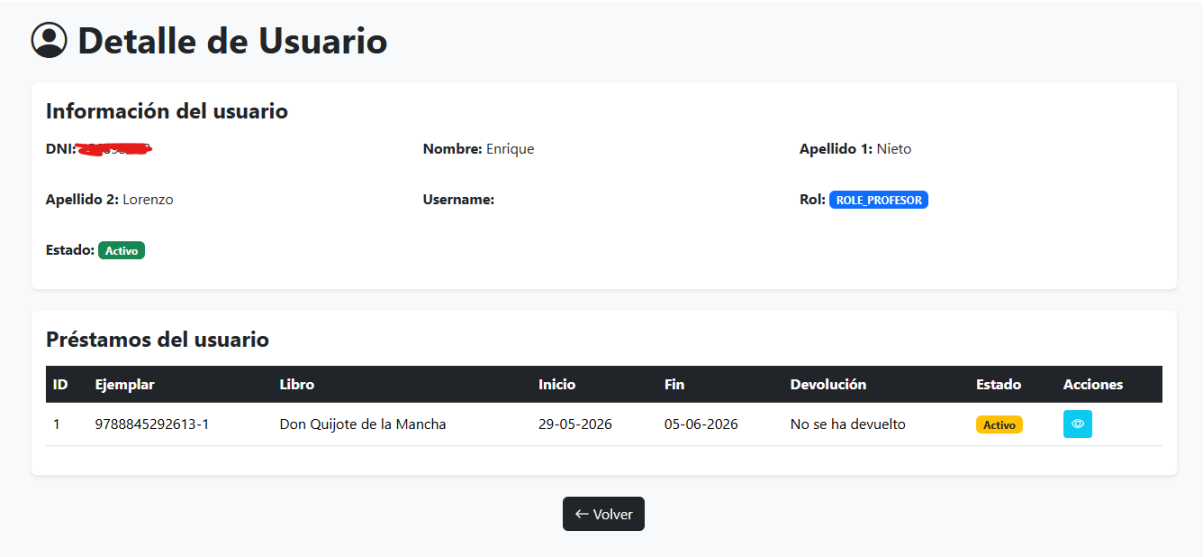


Este es el formulario de edición de usuario, exactamente igual que el registro de usuarios. En caso de cambiar el rol del usuario o ser originalmente un bibliotecario o un administrador aparecerán los campos de username y password.

Botón de baja / Rehabilitar

En caso de querer dar de baja a un usuario por cualquier motivo, se debe pulsar el botón de “Dar de baja”. Este cambiará el estado del usuario y cambiará su texto y funcionalidad ya que aparecerá un botón con el texto “Rehabilitar” y, al ser pulsado, reactivará al usuario en cuestión.

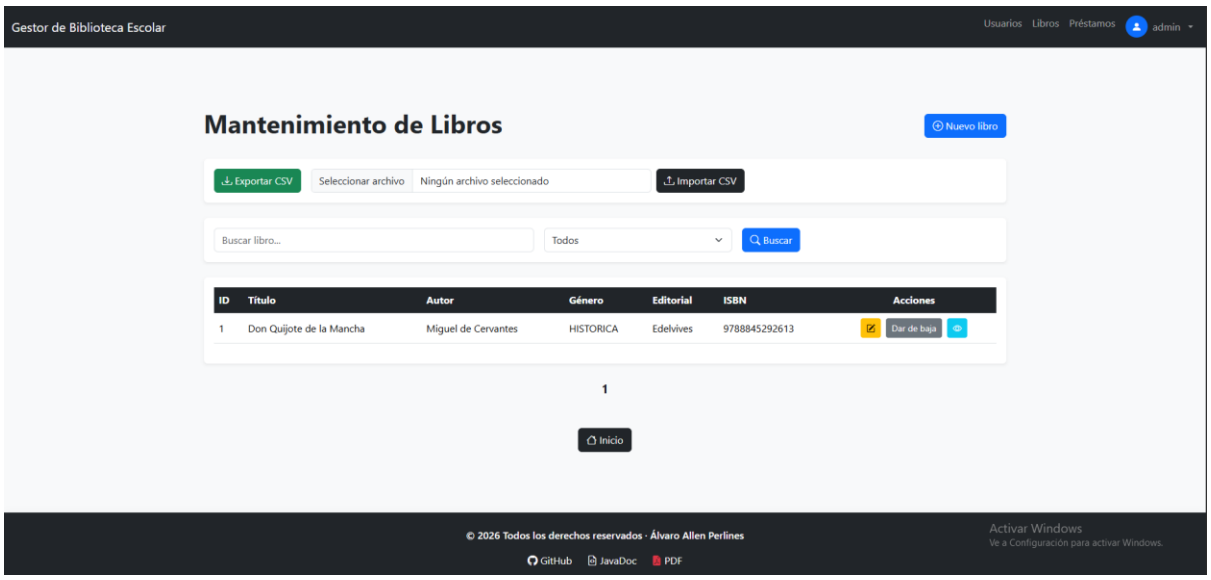
Consultar usuario



En la página de consulta de usuario podemos encontrar un cuadro con todos los datos del usuario especificados y el estado en el que se encuentra al igual que en la tabla de mantenimiento. Cabe destacar que, en el caso de que el usuario se encuentre suspendido por haber devuelto un préstamo fuera de plazo, aparecerá un botón al lado del estado el cual nos permitirá levantar la suspensión al usuario para que pueda volver a solicitar préstamos.

Debajo del cuadro se encuentra un listado con todos los préstamos realizados por el usuario pudiendo pulsar el botón de consultar y dirigirse a la consulta de préstamo.

Libros



Esto es el mantenimiento de libros donde se encuentran y gestionan todos los libros y ejemplares guardados en el sistema. El contenido principal de la página es una tabla en la que se listan los libros según el criterio establecido en el buscador con filtro. Igual que en el mantenimiento de usuarios, el

buscador es capaz de diferenciar entre ISBN, título y autor aparte de poder filtrar por activos, inactivos o todos.

Registrar usuario

Pulsando el botón superior “Nuevo libro” se accede al formulario de registro de libro.

En dicho formulario se deben rellenar los campos de forma obligatoria. Todos ellos están validados para no recibir ningún valor incorrecto y, a mayores, el campo de ISBN está validado con el patrón de ISBN 10 e ISBN 13. En caso de haber algún error se mostrará un mensaje explicativo debajo del campo correspondiente o en la parte superior de la tarjeta del formulario.

Exportar e importar libros

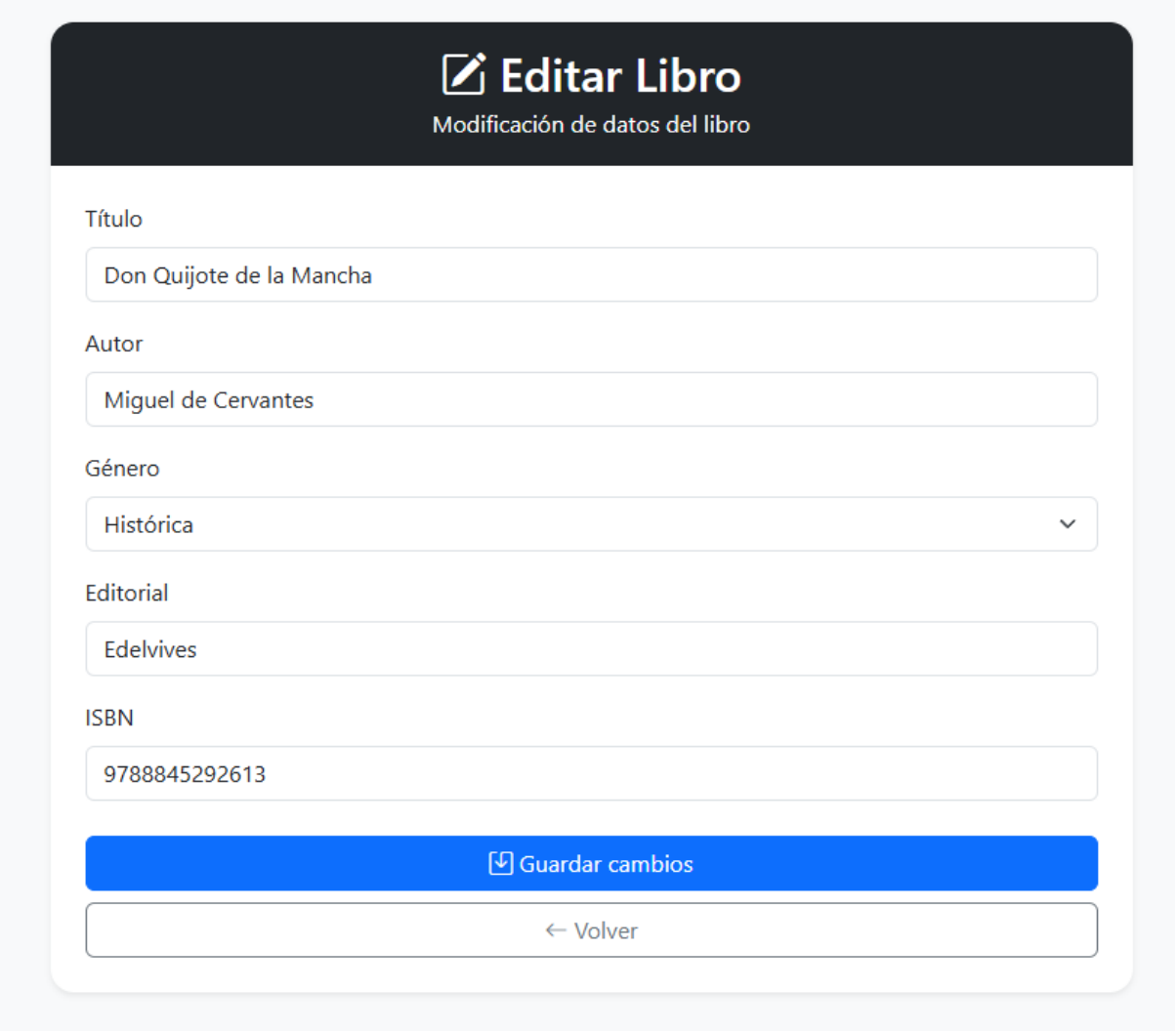
Volviendo de nuevo al mantenimiento de libros, se puede ver un apartado de exporta e importado de libros el cual exporta todo el listado en csv e importa el mismo tipo de archivo. En caso de haber algún

problema durante la importación se mostrará un mensaje de error explicando el motivo y deteniendo la acción.

Tabla de libros

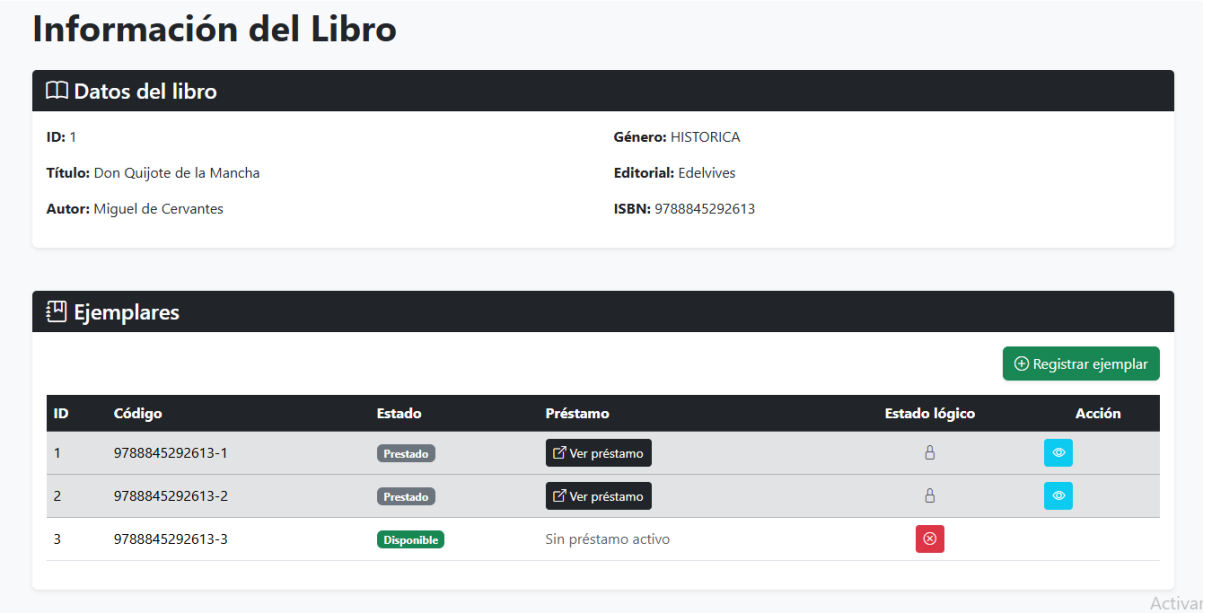
Como en todos los mantenimientos, tenemos una tabla en la que cada fila representa un libro y las columnas muestran cada valor siendo en este caso: título, autor, genero, editorial e ISBN. En la columna de acciones se puede dar de baja el libro en caso de no tenerlo en la biblioteca o no tener ejemplares de este. También se puede editar el libro abriendo un formulario idéntico al del registro, pero con los campos completados con los valores actuales del libro. Esta tabla también esta paginada de cinco en cinco libros.

Editar Libro



Cabe resaltar que todo aquel campo que no se modifique mantendrá el valor antiguo. También se puede decir que el ISBN es un campo identificativo y, en el caso de introducir un ISBN ya existente en el sistema, no se podrá completar la modificación saltando un error en el proceso.

Consultar libro

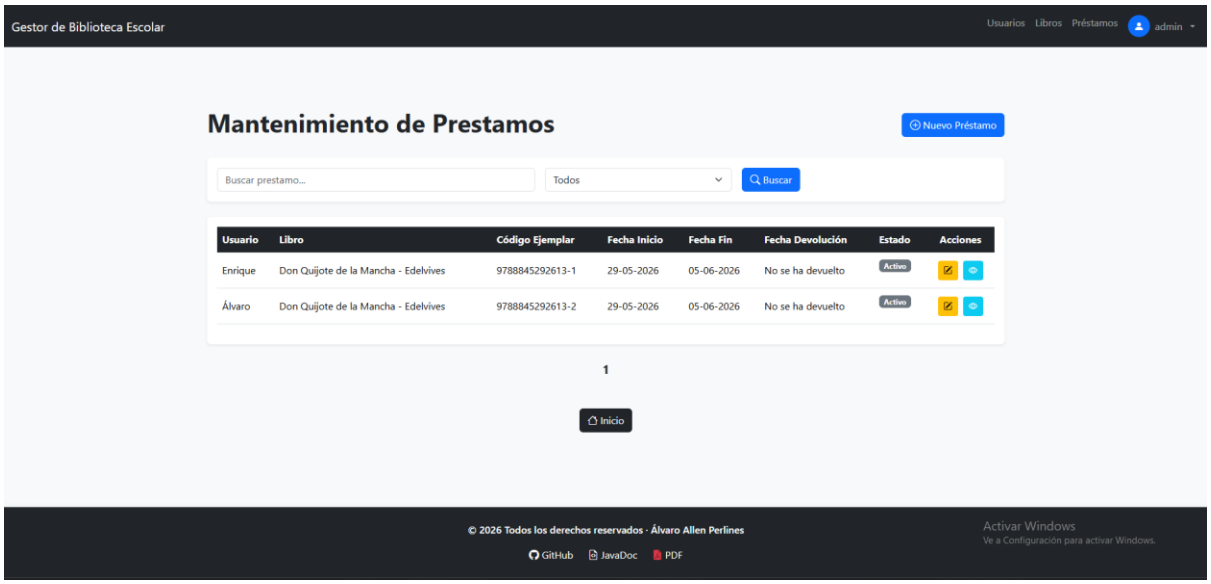


En los detalles de un libro podemos encontrar la información en un cuadro y debajo todos los ejemplares del libro. Hay que decir que se encuentran todos los ejemplares pudiendo dar de baja a aquellos que, por perdida, rotura o cualquier otro motivo ya no está disponible en la biblioteca.

Pulsando el botón de “Registrar ejemplar” se creará un nuevo ejemplar el cual guarda en el campo “Código” el ISBN del libro al que pertenece junto con el id de dicho ejemplar en la base de datos.

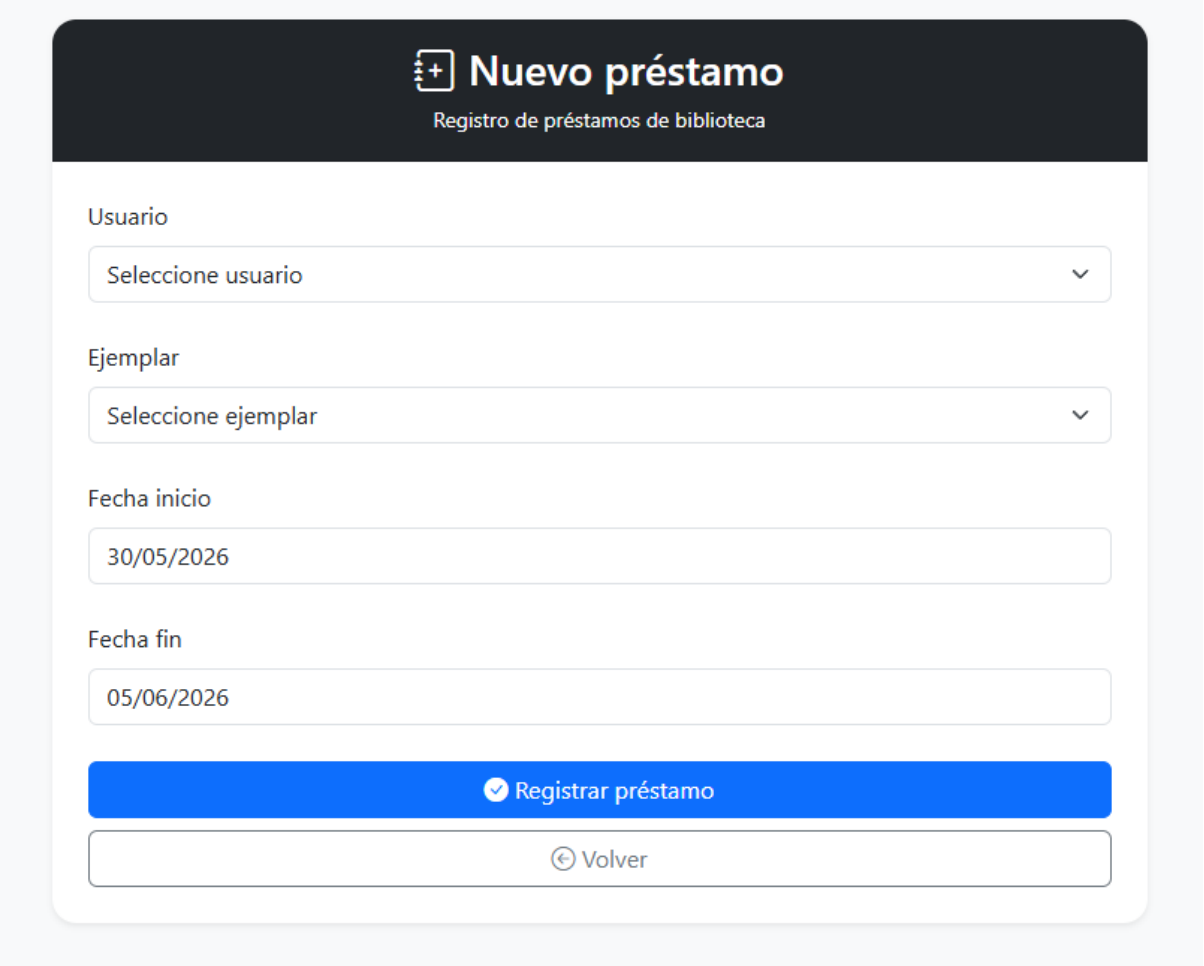
Todos aquellos ejemplares que están disponibles se pueden dar de baja o reactivar. Todos aquellos ejemplares ocupados tienen un botón de consulta de préstamo que, al igual que pasaba en detalle de usuario, nos lleva al detalle del préstamo al que pertenece.

Mantenimiento de préstamos



El mantenimiento de préstamos muestra y gestiona todos los préstamos guardados en el sistema siendo el corazón principal de la operación principal de esta aplicación. En el podemos registrar un nuevo préstamo pulsando el botón azul “Nuevo préstamo”.

Registro de préstamo



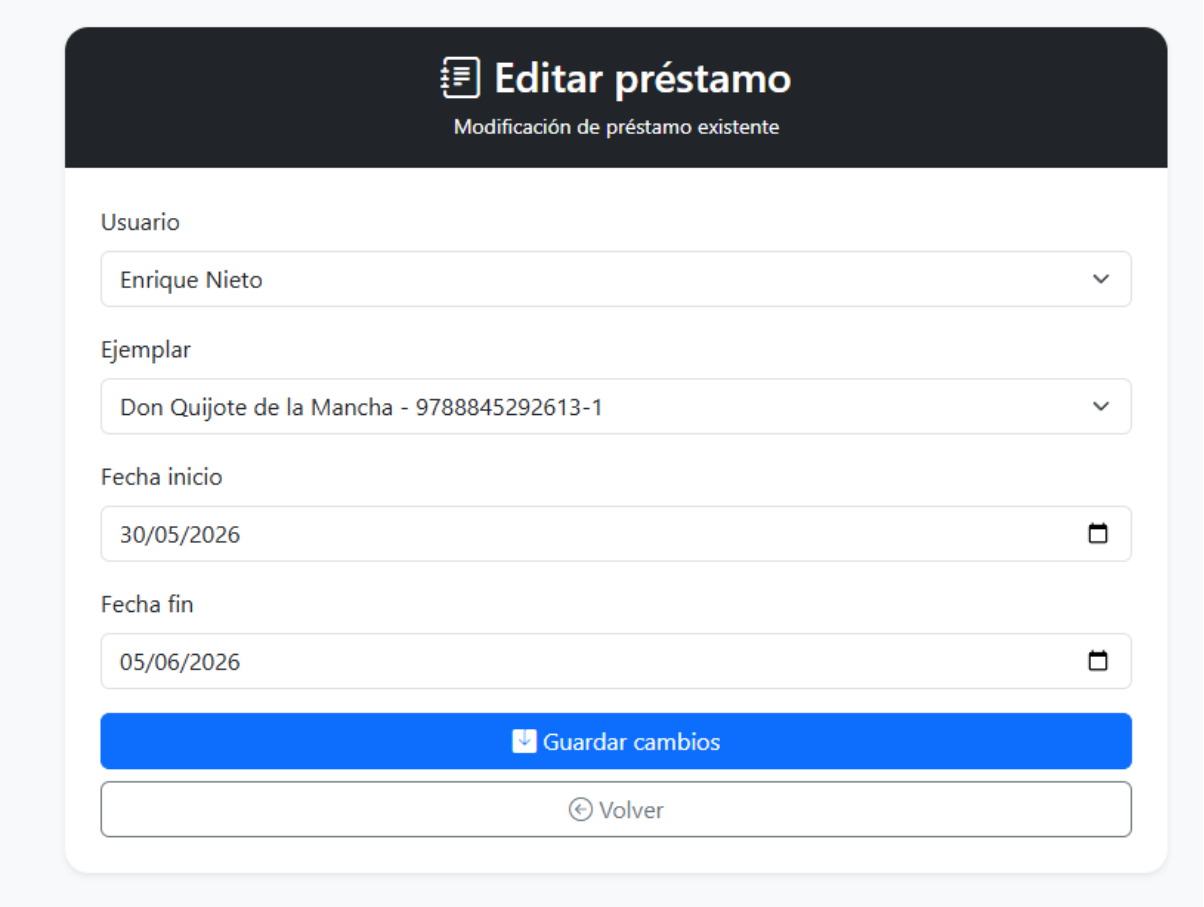
Este es el formulario de registro de préstamo y tiene mucho de qué hablar. Campo a campo estas son las características de un préstamo:

- En el select de usuario se encuentran todos los usuarios disponibles para hacer un préstamo. Para que un usuario pueda realizar un préstamo tiene que cumplir los siguientes requisitos:
    - Estar en estado activo. Si está de baja es porque ya no pertenece a la biblioteca y si está suspendido es por una falta disciplinaria.
    - Tener menos de 5 préstamos activos. Es decir, un usuario no puede tener más de cinco préstamos activos a la vez. Si quiere iniciar un nuevo préstamo en esa situación tiene que finalizar uno antes.
- En el select de ejemplar se encuentran los ejemplares disponibles del sistema. La forma de diferenciarlos es por el ISBN para saber el libro y el id del ejemplar lo que conforma el código identificativo de éste.
- Fecha de inicio y fecha de fin son dos campos fijos y no se pueden editar dado que la lógica de negocio es clara: el préstamo se inicia en el día que se realiza el préstamo y el límite de duración es hasta 5 días hábiles ya que no sería justo contar los fines de semana.

Tabla de préstamos

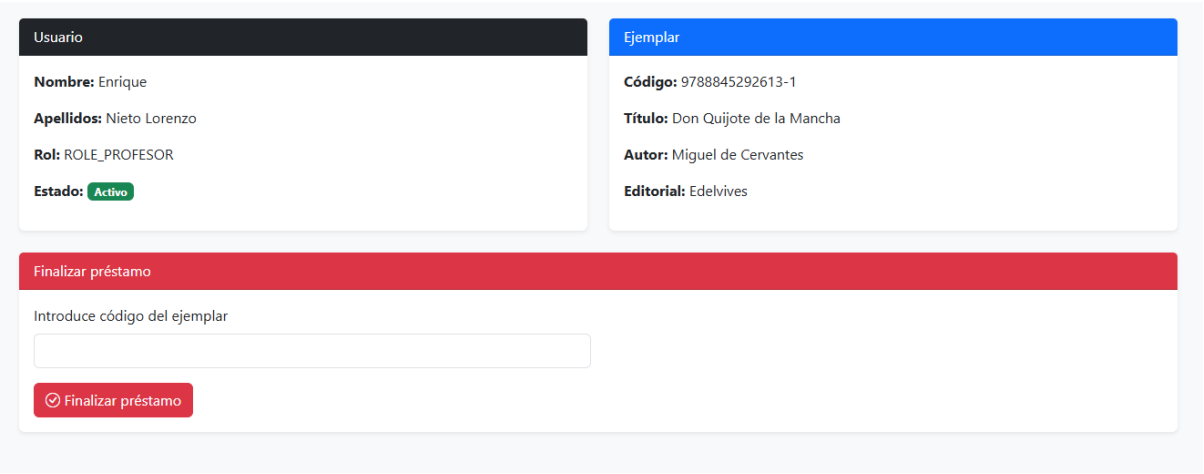
Al igual que en el resto de los mantenimientos de la aplicación, los préstamos se muestran en una tabla filtrada por búsqueda y por estado. La búsqueda puede ser por nombre del usuario o por el código del ejemplar prestado. En la tabla se muestran por columnas los siguientes datos: usuario, libro, ejemplar(código), fecha de inicio, fecha de fin, fecha de devolución y estado. Dicho estado se actualiza cada vez que se entra en el mantenimiento comparando la fecha actual con la fecha de fin para comprobar si está fuera de fecha o no.

Editar préstamo



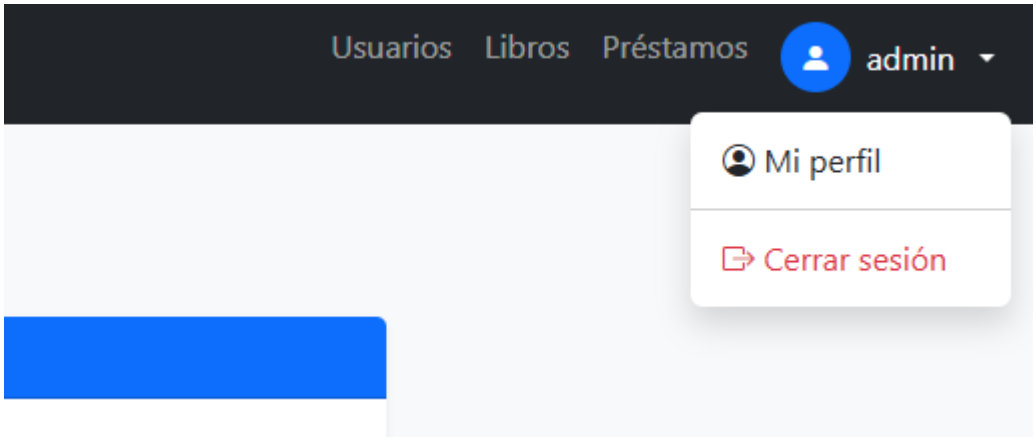
Se muestran los valores del préstamo en cada campo y en caso de no modificar un valor se queda como estaba. No se podrán seleccionar usuarios o ejemplares que no reunen las condiciones para realizar el préstamo.

Consultar préstamo



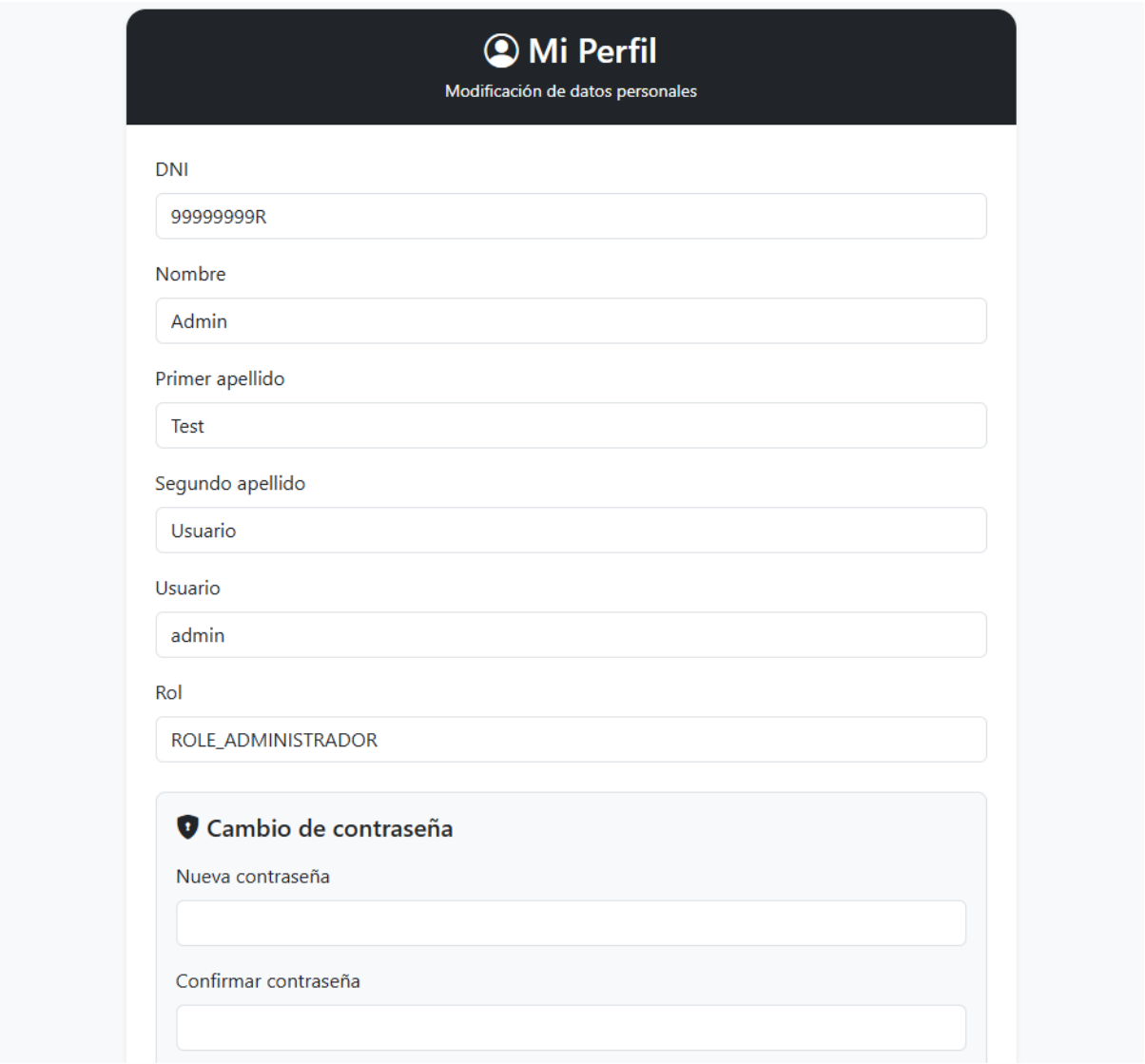
En la página de consulta de préstamo se muestran los datos del usuario y del ejemplar en dos tarjetas. Más abajo podemos encontrar el método de finalizado de préstamo. Para comprobar la identidad del ejemplar devuelto, el bibliotecario/administrador deberá introducir el código del ejemplar devuelto dentro del campo. Si el código es correcto, se finaliza el préstamo, pero si no lo es no podrá finalizarse hasta que aparezca dicho ejemplar. Durante la finalización del préstamo se comprueban las fechas de fin y de devolución para poder establecer si se ha devuelto a tiempo o no y si el usuario debe ser suspendido o no.

Perfil autenticado



Cuando un usuario está autenticado en el sistema tendrá en todo momento este desplegable en la esquina superior derecha de la página. En él podemos encontrar dos opciones: mi perfil, que nos redirige a la página de edición de datos y credenciales y cerrar sesión que simplemente cierra la sesión y redirige al inicio de la aplicación.

Editar perfil



Este es el formulario de edición del perfil autenticado en la aplicación donde puede modificar únicamente su nombre y apellidos y la contraseña para entrar en la aplicación. El cambio de contraseña está sujeto a una medida de seguridad mínima: introducirla dos veces para garantizar que se ha escrito correctamente y el usuario pueda confirmarlo.

Manual de despliegue

A continuación, se detalle el proceso, paso a paso, de cómo se despliega una aplicación Java Spring Boot MVC en el entorno de explotación Google Cloud.

Paso 1: Dockerización de la aplicación Spring Boot (contenedores)

El primer paso para que una aplicación sea "nube-nativa" es empaquetarla en un contenedor. Esto garantiza que la aplicación se ejecute exactamente igual en tu ordenador que en los servidores de Google Cloud.

Para ello, es necesario crear un archivo llamado Dockerfile (sin extensión) en la raíz de tu proyecto. El siguiente archivo contiene las instrucciones para construir la imagen de tu sistema. Se recomienda una configuración de multi-stage build para que la imagen final sea ligera y segura:

```
FROM maven:3.9.6-eclipse-temurin-21-alpine AS build
WORKDIR /app

COPY pom.xml .
COPY src ./src

RUN mvn clean package -DskipTests

FROM eclipse-temurin:21-jdk-alpine
WORKDIR /app

COPY --from=build /app/target/*.jar app.jar

EXPOSE 8080
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "app.jar"]
```

A continuación, se detalla el funcionamiento de cada instrucción:

- FROM maven:3.9.6-eclipse-temurin-21-alpine AS build: define la imagen base inicial que contiene Maven y el JDK 21. La etiqueta "AS build" nombra esta etapa para poder referenciarla más adelante.
- WORKDIR /app: establece el directorio de trabajo dentro del contenedor donde se ejecutarán los comandos.
- COPY pom.xml . Y COPY src ./src: copia los archivos de configuración de dependencias y el código fuente desde el equipo local al contenedor.

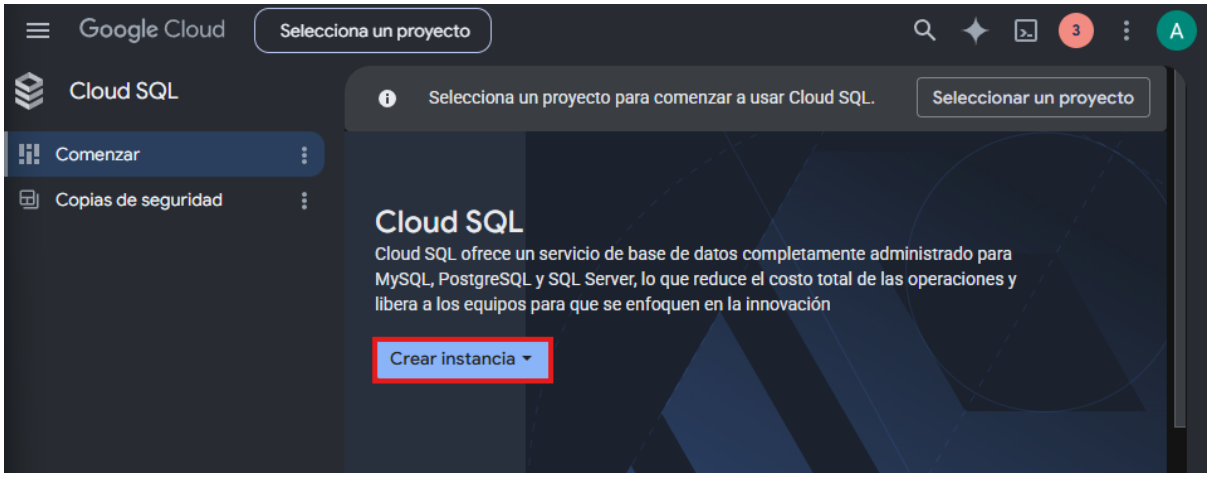
- RUN mvn clean package –DskipTests: ejecuta la compilación de maven. Genera el archivo comprimido .jar en la carpeta target. Se omiten los test para optimizar el tiempo de despliegue en la nube.
- FROM eclipse-temurin:21-jdk-alpine: inicia una segunda etapa con una iamgen mucho más pequeña que solo contiene el entorno de ejecución, no el compilador de ni Maven.
- COPY –from=build /app/target/*.jar app.jar: esta es la línea clave. Copia el archivo .jar construido en la etapa “build” y lo trae a la nueva imagen limpia.
- EXPOSE 8080: informa que el contenedor escuchará en el puerto 8080, que es el estándar de Spring Boot y el esperado por Cloud Run.
- ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "app.jar"]: define el comando definitivo que se ejecutará al iniciar el contenedor.

El concepto de Multi-stage build consiste en dividir el proceso de despliegue en dos fases:

- **Etapa de construcción:** se usan las herramientas pesadas para crear el archivo ejecutable.
- **Etapa de ejecución:** se desecha el anterior y solo se conserva el ejecutable sobre una basae mínima.
- **Ventajas:** reduce el tamaño final de la imagen, el despliegue es más rápido en Google Cloud y otorga una mayor seguridad.

Paso 2: Creación de la base de datos en GoogleCloud

Nos dirigimos a Cloud SQL y creamos una nueva instancia en el apartado de comenzar.

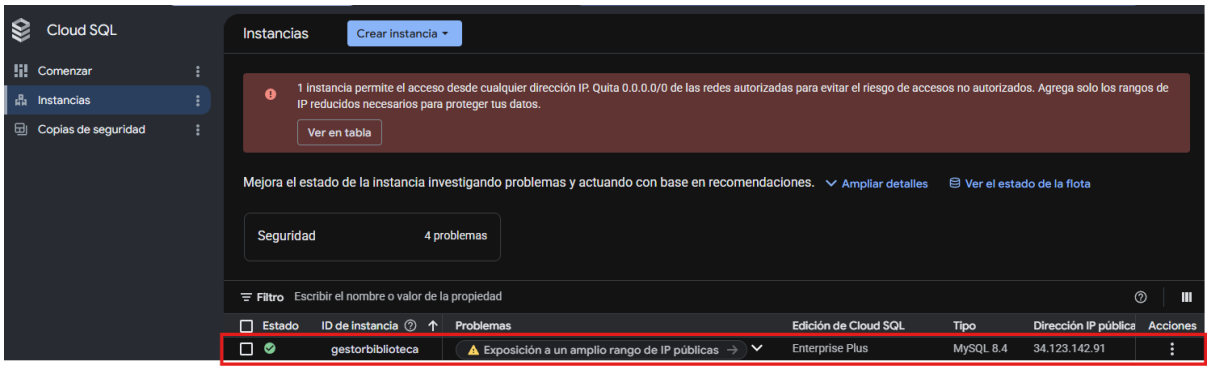


Seleccionamos MySQL y rellenamos los campos:

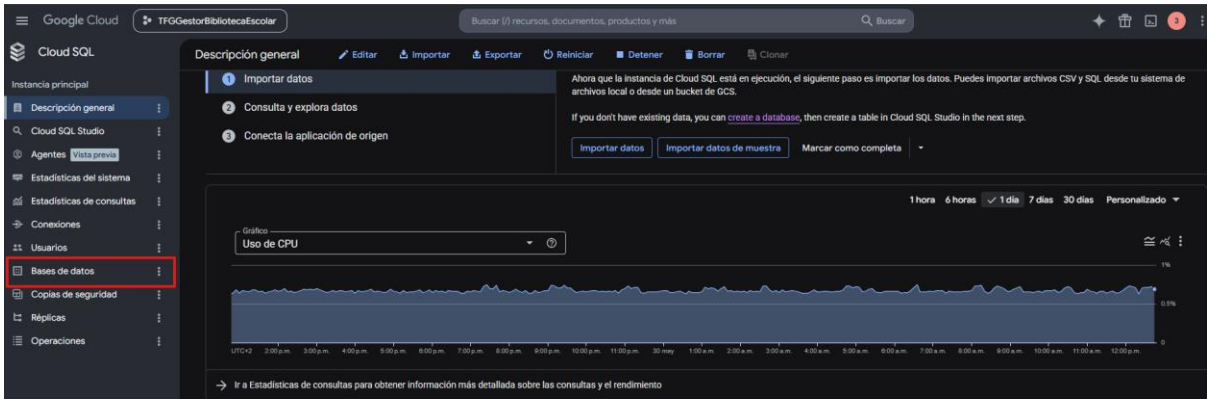
- Id de instancia va a ser el identificador de la instancia (útil en un futuro) y no podrá contener símbolos ni letras minúsculas.
- Contraseña para acceder a la hora de conectarse desde la configuración de la aplicación (guardar en un archivo de texto para usar después). Se recomienda hacer poner una contraseña complicada por seguridad.

- Elegir la zona en la que va a estar alojado el servidor de SQL. En este caso, se recomienda lo más cercano a tu país, siendo cualquier opción de Europa válida para España.

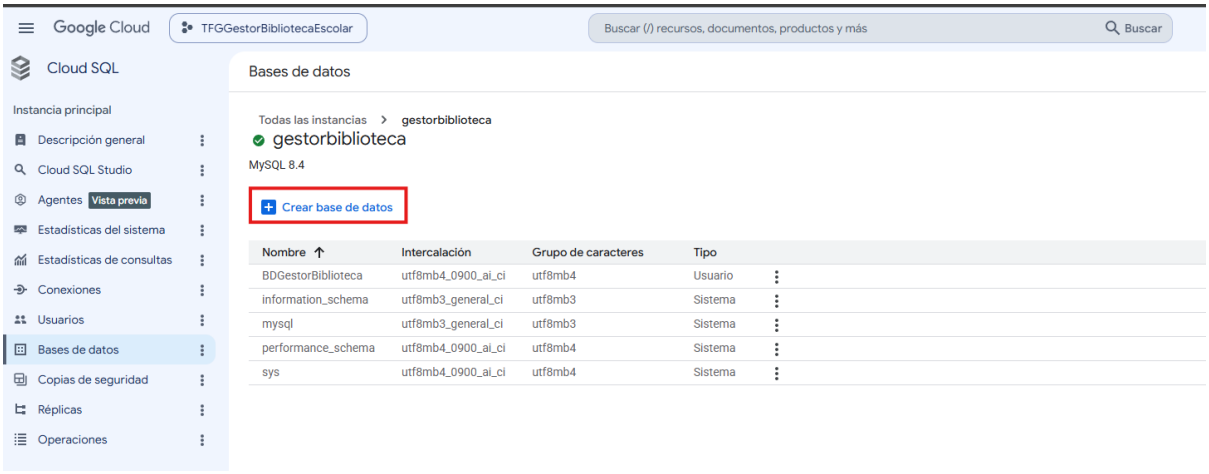
Una vez finalicemos dicho formulario, creamos la instancia. A continuación, vamos al apartado de instancias y seleccionamos la recién creada.



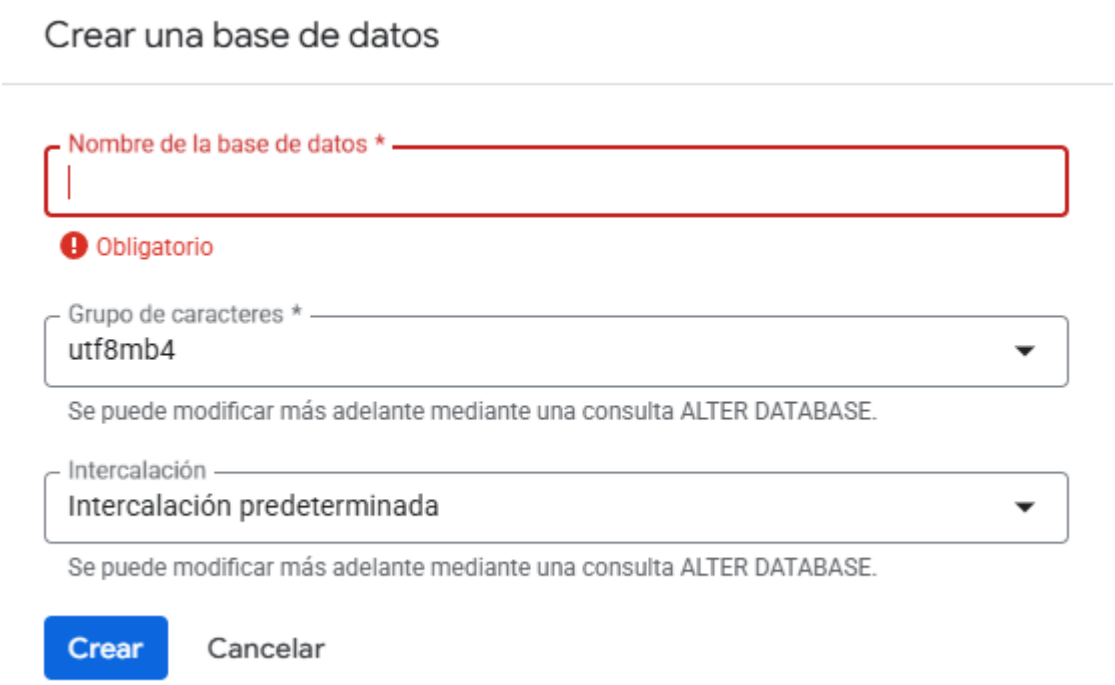
Se nos abrirá el panel de configuración y gestión de la instancia donde se encuentra el apartado de bases de datos donde crearemos la necesaria para la aplicación



Encontraremos varias bases creadas por defecto y la opción que crear una nueva.



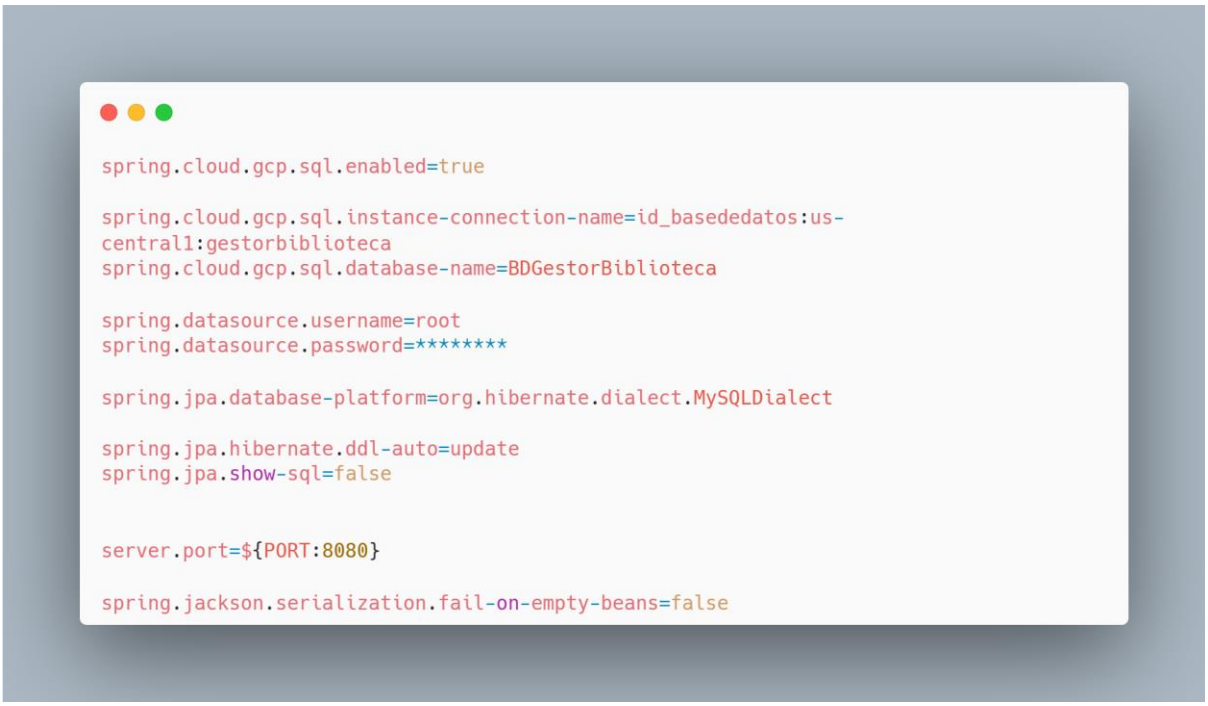
Introducimos el nombre de la base el cual será necesario en el archivo de configuración de la aplicación para realizar la conexión



Paso 3: Gestión de repositorio y configuración de perfiles (Spring Profiles)

Para que el despliegue sea existoso, la aplicación debe ser capaz de distinguir cuándo se está ejecutando en el ordenador del desarrollador y cuándo está en los servidores de Google Cloud. Para ello, se utiliza la potencia de los Spring Profiles.

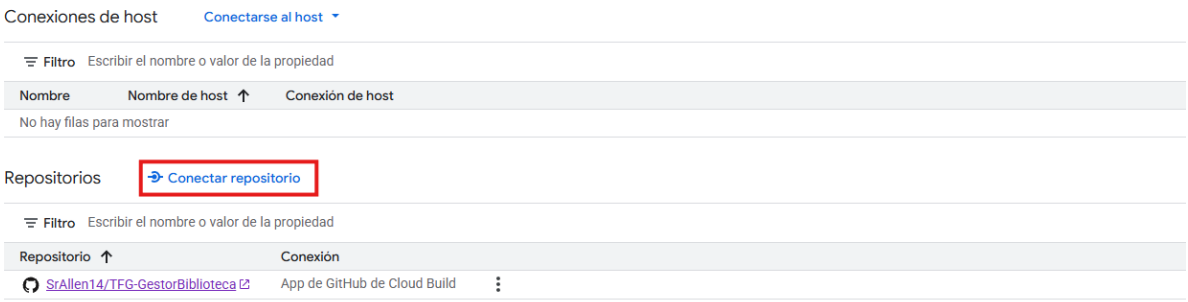
Dentro de la carpeta src/main/resources, junto con el archivo application.properties, se debe crear un archivo específico para la nube. Este archivo contendrá las credenciales de la base de datos Cloud SQL y los ajustes de Hibernate para producción.



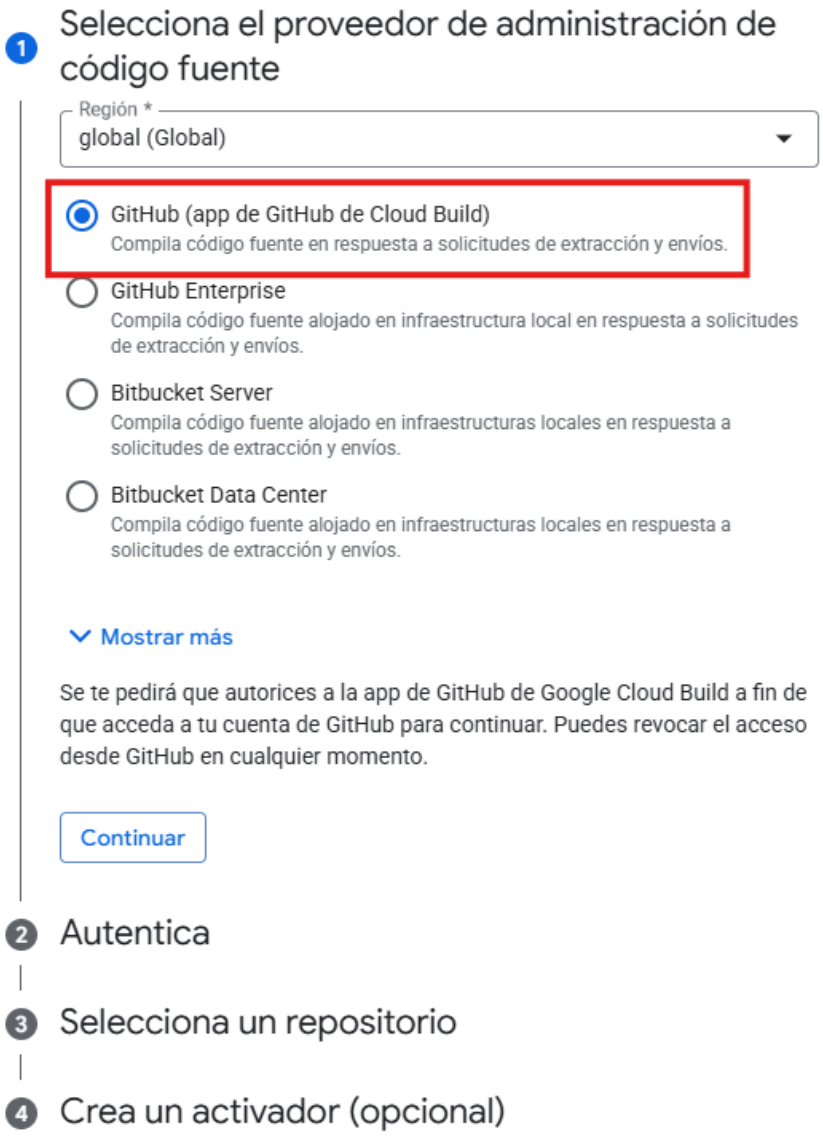
```
spring.cloud.gcp.sql.enabled=true

spring.cloud.gcp.sql.instance-connection-name=id_basededatos:us-
central1:gestorbiblioteca
spring.cloud.gcp.sql.database-name=BDGestorBiblioteca

spring.datasource.username=root
spring.datasource.password=********

spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false


server.port=${PORT:8080}

spring.jackson.serialization.fail-on-empty-beans=false
```

Una vez configurado el archivo, se procede a subir el código al repositorio remoto. Es fundamental asegurar que la rama principal (master o main) esté limpia y contenga el Dockerfile creado en el paso anterior. Realizamos un commit y un push en la rama master. Dicho push va es estar sujeto a un trigger creado en el google cloud.

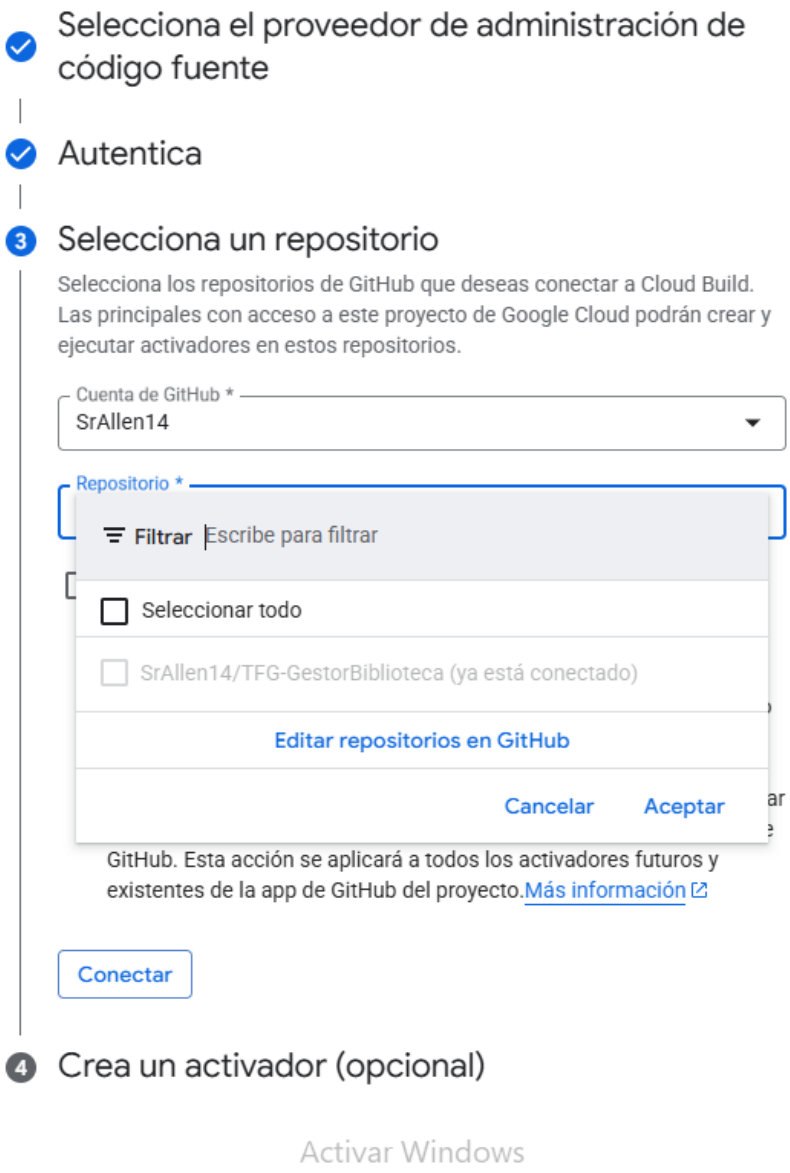
Desde el apartado de repositorios en Cloud Build debemos realizar la conexión con nuestro repositorio. Se puede escoger entre conectar la cuenta completa o escoger únicamente el respositorio del proyecto.



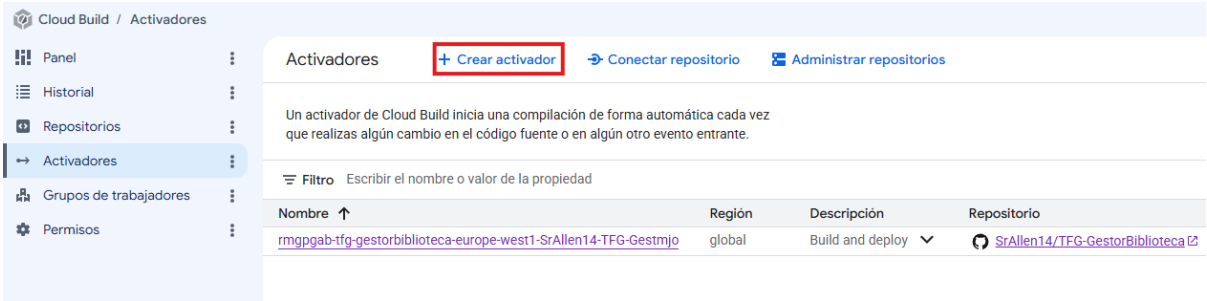
En el panel lateral que aparece se selecciona el repositorio que estamos usando online.


1 Selecciona el proveedor de administración de código fuente
Región *
global (Global)
GitHub (app de GitHub de Cloud Build)
Compila código fuente en respuesta a solicitudes de extracción y envíos.
GitHub Enterprise
Compila código fuente alojado en infraestructura local en respuesta a solicitudes de extracción y envíos.
Bitbucket Server
Compila código fuente alojado en infraestructuras locales en respuesta a solicitudes de extracción y envíos.
Bitbucket Data Center
Compila código fuente alojado en infraestructuras locales en respuesta a solicitudes de extracción y envíos.
Mostrar más
Se te pedirá que autorices a la app de GitHub de Google Cloud Build a fin de que acceda a tu cuenta de GitHub para continuar. Puedes revocar el acceso desde GitHub en cualquier momento.
Continuar
2 Autentica
3 Selecciona un repositorio
4 Crea un activador (opcional)

Acto seguido, seleccionamos el proyecto concreto que queremos subir o, como segunda opción, también podemos escoger todo y después definir cual subir.



Como se observa en la imagen superior, se puede crear un activador durante la conexión del repositorio GitHub, pero en este caso, se va a realizar desde el apartado de activadores donde se presenta mejor el formulario.



En dicho formulario estableceremos la rama en la que queremos situar el activador o "trigger". Este trigger se dispara cuando se realiza un "push" desde tu IDE al repositorio en GitHub haciendo que el desarrollo y el despliegue sea seguro.

Paso 4: configuración y despliegue en Google Cloud Run.

Una vez que el código está en GitHub y el Dockerfile está listo, el siguiente paso es configurar Cloud Run. Este servicio está encargado de recibir la imagen de Docker, ejecutarla y asignarle una dirección URL pública.

Para que el despliegue sea automático, primero debemos configurar el trigger entre Google y GitHub:

- En la consola de GCP, accedemos a Cloud Build > Activadores
- Creamos un nuevo activador vinculado a nuestro repositorio de GitHub
- Seleccionamos el evento de ejecución: "Empujar a una rama" y elegimos master.
- En "Configuración", seleccionamos Dockerfile, lo que indica a Google que debe usar las instrucciones que escribimos en el paso 1 para construir la imagen.

Con el activador listo, procedemos a configurar el servicio que mantendrá viva la aplicación:

- Vamos a Cloud Run y seleccionamos "Crear servicio".
- Elegimos la opción "implementar continuamente desde un repositorio" para que cada cambio en el código actualice la web automáticamente.
- Configuración del contenedor en el puerto 8080.
- Establecemos la variable de entorno "SPRING_PROFILES_ACTIVE" con valor "prod"

Conclusiones y Trabajo Futuro

Conclusiones

La realización de este TFG ha permitido desarrollar una aplicación web de gestión bibliotecaria utilizando tecnologías ampliamente empleadas en el desarrollo, como Spring Boot, Spring MVC, Spring Data JPA, MySQL y Thymeleaf. A mayores, el proceso de despliegue en Google Cloud ha supuesto una buena práctica de formación.

Durante el desarrollo se han aplicado conocimientos adquiridos a lo largo del grado relacionados con el análisis de requisitos, diseño de bases de datos, programación orientada a objetos, desarrollo web

y despliegue de aplicaciones. Asimismo, el proyecto ha permitido profundizar en el uso de arquitectura por capas y buenas prácticas de desarrollo, favoreciendo la creación de una aplicación estructurada mantenible y escalable.

Los objetivos planteados al inicio del proyecto han sido alcanzados satisfactoriamente, consiguiendo una aplicación funcional capaz de gestionar usuarios, ejemplares, préstamos y sanciones de forma centralizada.

Posibles mejoras a futuro

Aunque la aplicación desarrollada cumple con los requisitos establecidos, existen diversas mejoras que podrían incorporarse en futuras versiones para ampliar sus funcionalidades.

Una de las mejoras sería la implementación de un sistema de donación de libros por parte de alumnos y profesores. Esta funcionalidad permitiría incrementar el catálogo de la biblioteca, fomentar la reutilización de material y promover prácticas sostenibles dentro de la comunidad educativa.

Además, la incorporación de este sistema facilitaría la creación de mecanismos de recompensa para los usuarios más participativos. Por ejemplo, podrían ofrecerse ventajas como un mayor número de préstamos simultáneos, ampliaciones en los plazos de devolución o acceso prioritario a determinadas reservas.

Otras mejoras posibles, más a largo plazo, podrían ser la incorporación de un sistema de notificaciones para mejorar la experiencia de los usuarios, avisando de fechas de fin inminentes, nuevos libros en el catálogo, etc...

Webgrafías y Referencias

- Inicialización de Spring Boot:
  - Spring Team. (s.f.). *Installing Spring Boot*. Spring Documentation. https://docs.spring.io/spring-boot/installing.html
- Spring Boot JPA documentación:
  - Spring Team. (s.f.). *SQL Databases*. Spring Documentation. https://docs.spring.io/spring-boot/reference/data/sql.html
- Spring Security:
  - Spring Team. (s.f.). *Spring Security*. Spring Documentation. https://docs.spring.io/spring-boot/reference/web/spring-security.html
- Spring Validation:
  - Spring Team. (s.f.). *Validation*. Spring Documentation. https://docs.spring.io/spring-boot/reference/io/validation.html
- Spring initializr:
  - VMware, Inc. (s.f.). *Spring Initializr*. Spring Initializr. https://start.spring.io/
- Spring Framework (teoría y características):
  - Wikipedia Contributors. (2026, 8 de abril). *Spring Framework*. Wikipedia. https://es.wikipedia.org/wiki/Spring_Framework
- IoC (inyección de control):
  - Blancarte, O. (2016, diciembre 1). *Concepto de Inversion of Control*. Oscar Blancarte Blog. https://www.oscarblancarteblog.com/2016/12/01/concepto-inversion-of-control/
  - Wikipedia Contributors. (2025, 1 de marzo). *Inversión de control*. Wikipedia. https://es.wikipedia.org/wiki/Inversi%C3%B3n_de_control
- DI (inyección de dependencias):
  - Arquitectura Java. (s.f.). *Inyección de dependencia: un patrón clave*. Arquitectura Java. https://www.arquitecturajava.com/inyeccion-de-dependencia-un-patron-clave/
  - Adictos al Trabajo. (2013, julio 25). *Spring Container e Inyección de Dependencias*. Adictos al Trabajo. https://adictosaltrabajo.com/2013/07/25/spring-container-inyeccion-dependencias/
- Spring Boot (teoría):
  - IBM. (s.f.). *¿Qué es Spring Boot?*. IBM Think. https://www.ibm.com/es-es/think/topics/java-spring-boot
- Spring Boot MVC:

  - Centro Nacional de Actualización y Capacitación Empresarial (CNAC). (s.f.). *¿Qué es Spring MVC?*. CNAC. https://www.cnac.es/noticias/que-es-spring-mvc/
- DispatcherServlet:
  - Vino7Tech. (s.f.). *Understanding DispatcherServlet in Spring MVC*. Medium. https://medium.com/@vino7tech/understanding-dispatcherservlet-in-spring-mvc-f49e034de016
- Flujo de petición HTTP:
  - Fastly. (s.f.). *Qué es una solicitud HTTP*. Fastly Learning Center. https://www.fastly.com/es/learning/http-requests-explained
- Hibernate:
  - Wikipedia Contributors. (2024, 10 de junio). *Hibernate*. Wikipedia. https://es.wikipedia.org/wiki/Hibernate
  - NTT DATA. (s.f.). *¿Qué es Java Hibernate y por qué usarlo?*. NTT DATA. https://ifgeekthen.nttdata.com/s/post/que-es-java-hibernate-por-que-usarlo-MC5FU56AIPGBGIHNJ677RBIXUHOI?language=es
- MySQL (teoría):
  - Oracle Corporation. (s.f.). *¿Qué es MySQL?*. Oracle. https://www.oracle.com/es/mysql/what-is-mysql/
- Maven:
  - Arquitectura Java. (s.f.). *¿Qué es Maven?*. Arquitectura Java. https://www.arquitecturajava.com/que-es-maven/
- Archivo pom.xml, características:
  - MTech, J. (s.f.). *Archivos pom.xml y settings.xml en Maven: una guía detallada*. JosemTech. https://josemtech.com/archivos-pom-y-settings-xml-en-maven-una-guia-detallada/
- Anotaciones Spring Boot:
  - MVIT Innovación Tecnológica. (2020, 6 de febrero). *Guía de anotaciones de Spring Framework*. MVIT Innovación Tecnológica. https://mvitinnovaciontecnologica.wordpress.com/2020/02/06/guia-de-anotaciones-de-spring-framework/
- Servidor Tomcat y Servlets:
  - Wikipedia Contributors. (2026, 31 de enero). *Apache Tomcat*. Wikipedia. https://es.wikipedia.org/wiki/Tomcat
  - Arquitectura Java. (s.f.). *¿Qué es un Servlet?*. Arquitectura Java. https://www.arquitecturajava.com/que-es-un-servlet/
- Guía de despliegue de Google Cloud:

- Google Cloud. (s.f.). *Cloud SQL para MySQL*. Google Cloud Documentation. https://docs.cloud.google.com/sql/docs/mysql?hl=es-419
- Google Cloud. (s.f.). *Despliegue continuo desde un repositorio GitHub*. Google Cloud Documentation. https://docs.cloud.google.com/run/docs/quickstarts/deploy-continuously?hl=es-419#cloudrun_deploy_continuous_code-java

Tutoriales de Spring Boot

- CodeJava. (2021, 19 de noviembre). *Inicio de sesión y registro de usuarios con Spring Security + Thymeleaf + MySQL y Bootstrap*[Vídeo]. YouTube. https://www.youtube.com/watch?v=0wTsLRxS3gA
- CodeJava. (2021, 27 de octubre). *Desarrollo de un CRUD Completo en Spring Boot con MySQL, Thymeleaf y Bootstrap* [Vídeo]. YouTube. https://www.youtube.com/watch?v=oF3XmiHgT-I