UNIVERSIDAD ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

INFORME DE ARQUITECTURA
Módulo de préstamos deportivos
ECI-Bienestar
Equipo Ingenieros BackEnd Grupo Esmeralda

AUTORES
Roger Rodriguez y Sebastian Galvis

PROGRAMA
Ingeniería de Sistemas

ASIGNATURA
Ciclos de Vida del Desarrollo de Software (CVDS)

PROFESORES
Ing. Rodrigo Humberto Gualtero Martínez
Ing. Andrés Martín Cantor Urrego

BOGOTÁ DC, COLOMBIA
Sábado, 10 de Mayo de 2025

Módulo de Prestamos Deportivos

Este módulo permite a los miembros de la comunidad reservar y acceder al préstamo de equipos deportivos disponibles en el coliseo, facilitando tanto la solicitud como la devolución de los elementos. A su vez, los funcionarios de bienestar gestionan la disponibilidad del inventario y verifican el estado de los equipos al momento de la devolución, garantizando así un uso adecuado de los recursos institucionales.

Objetivo del modulo:

Requisitos Funcionalidades:

1. Solicitud y registro de préstamo: El estudiante podrá solicitar el préstamo de uno o más elementos deportivos desde su perfil institucional. o La plataforma permitirá consultar la disponibilidad en tiempo real y realizar la reserva según los horarios definidos por Bienestar.

2. Datos asociados al préstamo: Cada solicitud deberá registrar la siguiente información: o ID único del préstamo. o Nombre completo del usuario. o Número de identificación. o Rol dentro de la institución (Estudiante, Docente, Administrativo, Servicios Generales). o Tipo y descripción del equipo solicitado. o Fecha y hora del préstamo. o Fecha y hora programada para la devolución. o Duración total del préstamo. o Estado del equipo al momento del préstamo y al momento de la devolución (Ej: En buen estado, Dañado, Requiere mantenimiento). o Registro de devolución confirmada por el funcionario responsable. CVDS COMPANY

3. Gestión de disponibilidad de equipos: Los funcionarios de bienestar podrán habilitar, deshabilitar o limitar la disponibilidad de ciertos equipos, ya sea por mantenimiento, pérdida o rotación de uso. o El sistema debe impedir el préstamo de equipos que no estén disponibles o que hayan sido reportados como dañados.

4. Verificación y control de estado: Al momento de la devolución, el funcionario deberá verificar el estado del equipo y registrar observaciones si aplica. o El sistema permitirá marcar elementos que requieren reparación o reposición, activando alertas para el personal encargado.

5. Reportes administrativos: El sistema deberá permitir generar reportes de préstamos por estudiante, por tipo de equipo, por estado del elemento, y por rango de fechas. o Estos reportes servirán para evaluar el uso del inventario y facilitar la toma de decisiones para mejoras o adquisiciones futuras.

6. Notificaciones y recordatorios automáticos: El sistema notificará al estudiante sobre la fecha y hora límite de devolución del equipo. o También podrá enviar alertas en caso de retraso o si se reporta un elemento como no devuelto.

7. Integración con el sistema general de Bienestar: Este módulo se integrará con los otros módulos (gestión de salas, clases extracurriculares, turnos de atención), permitiendo al usuario acceder a todos los servicios desde una única plataforma.

Tecnologías a Usar:

Lenguaje:

- **Java 17**
  - Java 17 es la versión LTS (Long-Term Support) más reciente, con mejoras de rendimiento, estabilidad y soporte a largo plazo. Es una opción sólida para aplicaciones empresariales, como es este caso, y se integra bien con Spring Boot y otros frameworks asociados.

Frameworks:

- **Spring Boot (para el Backend)**
  - Spring Boot es ideal para crear aplicaciones RESTful con una rápida configuración. Facilita la integración con servicios como MongoDB y ofrece muchas funcionalidades para manejar seguridad, autenticación (Spring Security), gestión de excepciones, validación de datos y más.
- **Spring Data MongoDB**
  - Como base de datos, MongoDB es adecuado para almacenar registros de seguimiento físico, rutinas y reservas, debido a su naturaleza flexible y escalable. Spring Data facilita la integración con MongoDB y la creación de repositorios eficientes.

Servicios Adicionales:

- Azure DevOps (para CI/CD)
  - **Razón**: Azure DevOps facilitará la integración continua, la entrega continua y la gestión del ciclo de vida del proyecto. Permite automatizar las pruebas, la implementación y el monitoreo del sistema.

Autenticación y Seguridad:

- Spring Security
  - **Razón**: Spring Security es la opción estándar para manejar la autenticación y autorización en aplicaciones Spring. Con roles definidos (Estudiante, Entrenador, Administrador, etc.), se puede controlar el acceso a los diferentes módulos de manera eficiente y segura.

Planteamiento Comparativo entre Tecnologías:

Java 17 vs. Otras Tecnologías (Node.js, Python):

- Java 17 ofrece mejor rendimiento y estabilidad para aplicaciones a gran escala, especialmente cuando se manejan grandes volúmenes de datos como en este caso. Node.js es ideal para aplicaciones en tiempo real, pero Java ofrece una mayor capacidad de procesamiento y una mayor cantidad de recursos empresariales.
- Python es más flexible y tiene una comunidad de desarrollo de inteligencia artificial más activa, pero Java sigue siendo la opción más robusta y escalable para sistemas de gestión empresarial.

MongoDB vs. SQL (MySQL, PostgreSQL):

- MongoDB es una excelente opción debido a su flexibilidad para manejar datos no estructurados o semiestructurados, como los datos de progreso físico y rutinas personalizadas. Si bien SQL puede ofrecer mejores garantías de integridad referencial, MongoDB permite un desarrollo más ágil, especialmente con registros que pueden variar mucho en términos de estructura.

Spring Boot vs. Otras Opciones (Django, Express):

- Spring Boot es altamente robusto y adecuado para aplicaciones empresariales complejas que requieren integración con bases de datos NoSQL, manejo de usuarios, seguridad y escalabilidad.
- Django es más sencillo y rápido para proyectos pequeños y medianos, pero no ofrece la misma modularidad y escalabilidad que Spring Boot.
- Express.js es ideal para aplicaciones que necesitan un alto rendimiento y velocidad de desarrollo, pero no proporciona tanto de forma nativa para la seguridad y la administración de bases de datos en comparación con Spring Boot.

¿Por qué usar Springboot?
Esto es debido a los siguientes factores que nos ofrece springboot:

- **Configuración mínima**
  Spring Boot elimina la necesidad de configuraciones extensas con sus valores por defecto inteligentes, permitiendo arrancar proyectos rápidamente.
- **Integración con el ecosistema Spring**
  Está construido sobre el ecosistema Spring, que es robusto, bien probado y con una gran comunidad detrás.
- **Servidor embebido**
  Spring Boot permite correr aplicaciones como servicios independientes sin necesidad de desplegar en un servidor externo (Tomcat, Jetty, etc.).
- **Producción lista desde el inicio**
  Tiene muchas características listas para producción como métricas, monitoreo, gestión de errores, y más.
- **Auto configuración**
  Detecta los componentes y configura automáticamente lo necesario (por ejemplo, conexiones a base de datos, seguridad, etc.).
- **Facilidad de pruebas**
  Soporta pruebas integradas de forma sencilla, tanto unitarias como de integración.
- **Amplio ecosistema y comunidad**
  Tiene una comunidad enorme, excelente documentación y recursos disponibles.
- **Escalabilidad y mantenimiento**
  Es adecuado tanto para microservicios como para aplicaciones monolíticas escalables.

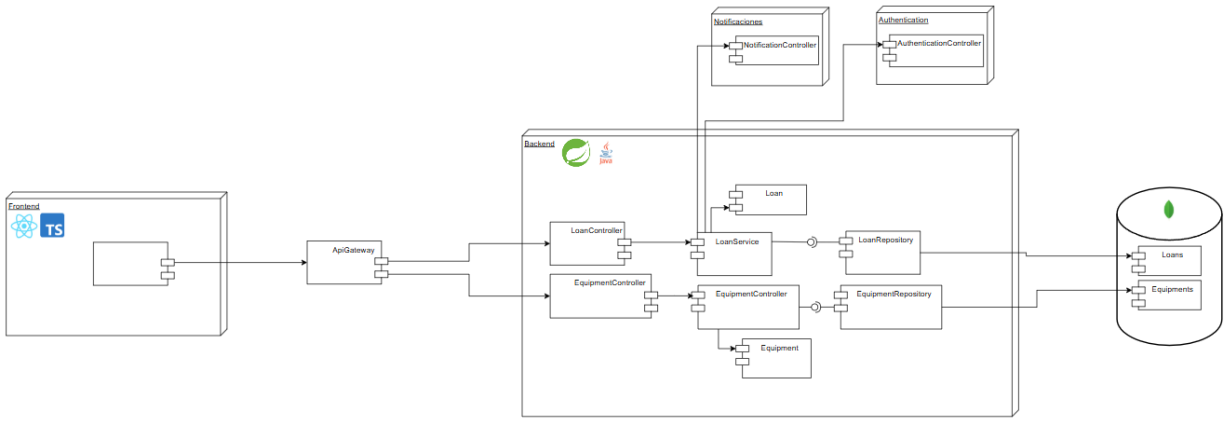
Aquí presentamos una tabla que lo compara con diferentes frameworks y nos muestra sus ventajas y desventajas frente a estos

Característica	Spring Boot	Jakarta EE (Java EE)	Micronaut	Quarkus
Configuración inicial	Mínima, con autoconfiguración	Extensa, requiere configuración manual	Muy ligera	Ligera, enfoque cloud-native
Modularidad	Muy alta, integración con módulos Spring	Modular pero más rígida	Alta, pero menor ecosistema	Alta, enfocado en microservicios
Tiempo de arranque	Rápido (aunque no el más rápido)	Lento en aplicaciones grandes	Muy rápido	Muy rápido, ideal para GraalVM
Testing	Soporte completo y maduro	Menor soporte nativo	Buen soporte	Buen soporte
Servidor embebido	Sí (Tomcat, Jetty, Undertow)	No, necesita contenedor	Sí	Sí

		externo		
Productividad del desarrollador	Muy alta, gracias a herramientas y comunidad	Media	Alta	Alta
Comunidad y soporte	Muy grande y activa	Más limitada actualmente	Más pequeña, en crecimiento	En crecimiento, con apoyo de Red Hat
Ecosistema	Enorme (Spring Security, Data, etc.)	Limitado en comparación	Aceptable	Compatible con algunas librerías Spring
Filosofía	Convención sobre configuración	Configuración explícita	Ligereza y rendimiento	Rendimiento y cloud-native

Diagramas de arquitectura del modulo:

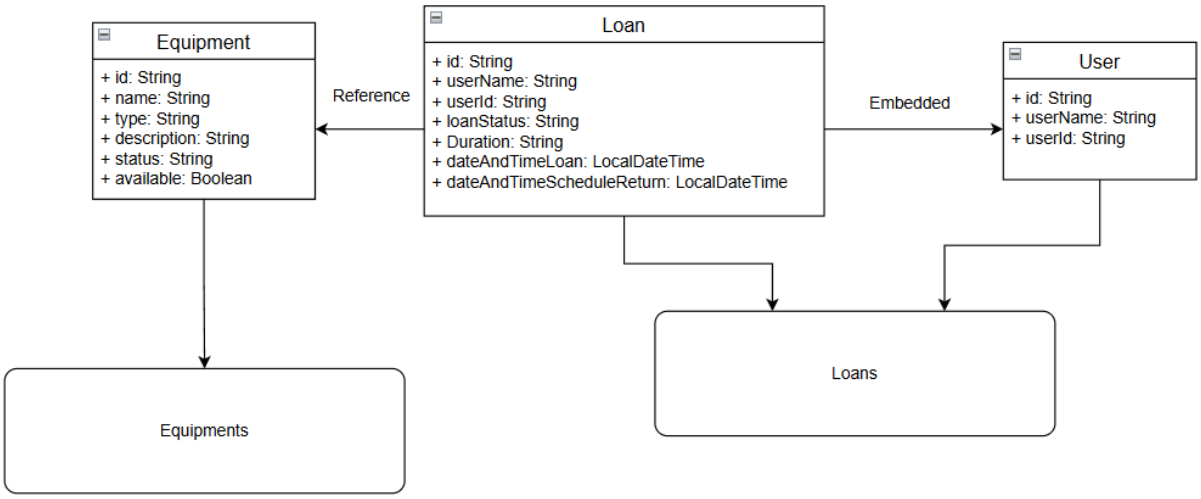
1.Diagrama de Componentes:



Se tienen las clases de equipment, user y loan las cuales representan la lógica de nuestro de negocio de préstamos, donde tenemos la información de cada objeto en equipment y se diferencian por tipos, por otro lado las loan tienen referencia a que objeto se le hizo préstamo y los diferentes datos necesarios para la realización y control del préstamo.

2.Diagrama de Datos:

A nivel general contamos con dos colecciones principales :
- **equipaments :** Donde se guarda la información de los artículos.
- **Loans :** Alli encontramos los detalles de los prestamos y en esta se encuentra embebido el usuario el cual es el encargado de realizar el préstamo , allí también se hace referencia al artucilo el articulo a reservar.



Descripción de las Colecciones y sus Campos:

1. Equipamentos (equipments): Representa a los equipos o artículos para el préstamo. Este documento tiene la información del articulo , este se vería tal que asi:

   {

   “id” : “01”,

   “name” : “Balon de futbol”,

   “type” : “Balon”,

   “description” : “Balon de futbol hecho de poliuretano material sintetico cocido, de la marca golty ”,

   “available” : true (Disponibilidad del objeto para el prestamo),

   “status” : “Nuevo” (Estado del articulo: “Nuevo” , “bueno” , “mal estado” , etc..)

   }

2. Prestamos (Loans): Representa el prestamo de un articulo deportivo hecho por un usuario y se vería algo asi:

```
{
  "_id": "loan-001",
  "equipmentId": "eq-123",
  "user": {
    "userId": "usr-456",
    "userName": "María González",
    "email": "maria@universidad.edu",
  },
  "dateAndTimeLoan": "2025-05-21T09:00:00",
  "loanStatus": "Activa",
  "dateAndTimeScheduleReturn": "2025-05-25T11:00:00",
```

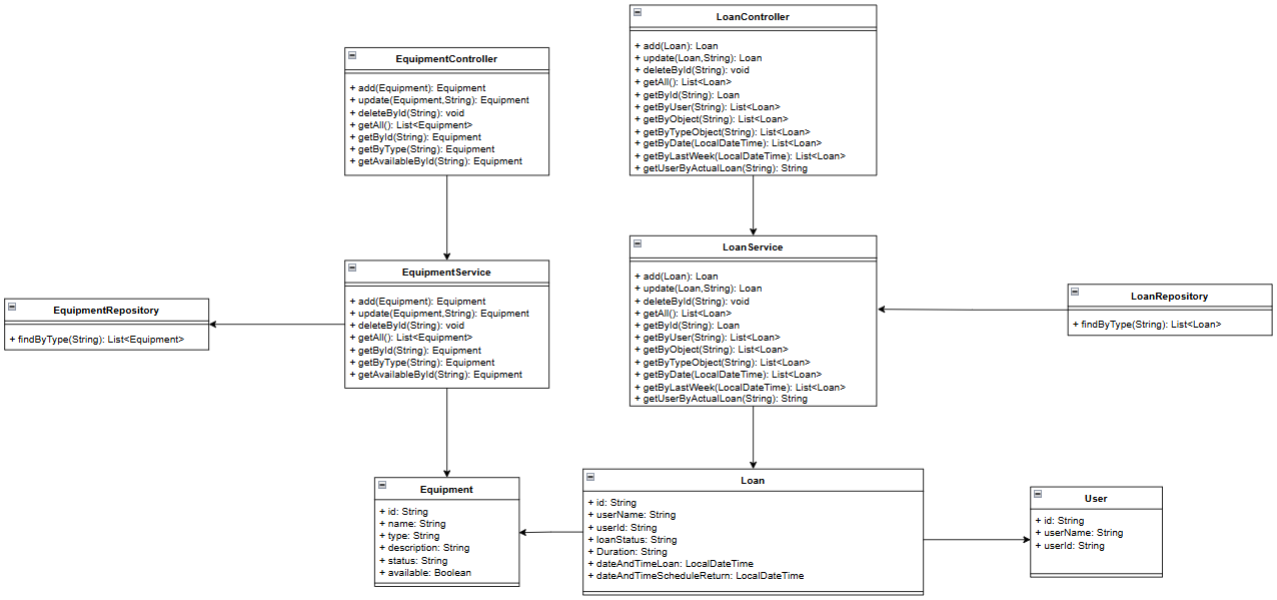
```
  "duration": "2 horas"
 },
 {
  "_id": "loan-002",
  "equipmentId": "01",
  "user": {
   "userId": "10002131",
   "userName": "Carlos Gomez",
   "email": "carlos.gomez-a@escuelaing.edu.co",
  },
  "dateAndTimeLoan": "2025-05-24T14:00:00",
  "loanStatus": "Completado",
  "dateAndTimeScheduleReturn": "2025-05-24T16:00:00",
  "duration": "2 horas",
  "returnConditions": "Buena"
 }
```


3.Diagrama de Clases:

En este diagrama se encuentran todas las clases a usar en el sistema como lo son los servicios, controladores , repositorios y entidades asi como las respectivas relaciones entre estas , a continuación se explica un poco del funcionamiento:

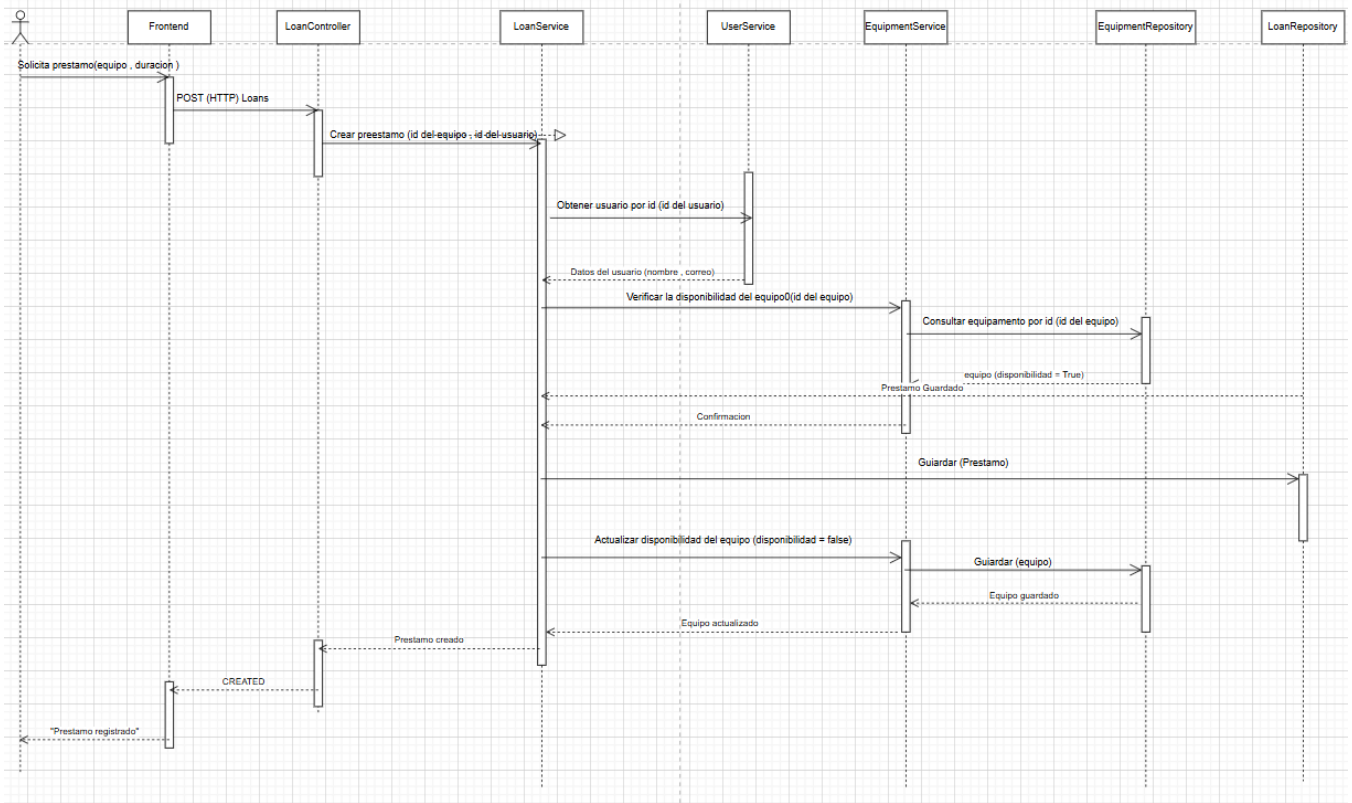
- EquipmentController : Recibe las peticiones HTTP y delega la lógica comunicándose con “EquipmentService”.
- EquipamentService : Gestiona disponibilidad y estado de los equipos , se comunica con “EquipmentRepository” para hacer operaciones de base de datos como almacenar la información .

- LoanController: Gestiona los prestamos mediante “LoanService” llamándole para registrar o consultar prestamos.
- LoanService : Maneja la lógica de creación de prestamos y devoluciones , se comunica “LoanRepository” para la persistencia de datos, asi como las consultas.
  Este también se comunica con el servicio externo de “UserService”, verifica el equipo mediante “EquipmentService”.



4.Diagrama de secuencia:

Tenemos a continuación la modelación de la secuencia que sigue el usuario para crear un préstamo que es el servicio principal que se presta en el modulo , es un modelo general de la funcionalidad principal:



Consideramos que no es necesario explicarlo ya que a nivel del mismo diagrama es solo seguir las flechas y ver las relaciones entre cada clase , lo cual permite entender fácilmente pero en resumen un usuario desde el front end solicita hacer un préstamo pero para esto primero se verifica el usuario una vez verificado se consulta la disponibilidad del articulo a prestar esta consulta nos muestra si esta disponible para el préstamo , si es asi retorna una confirmación y se guarda el presamo con su respectiva información en la base de dato, luego se actualiza el estado del equipo a no disponible para el préstamo y se guarda en la base de datos; Luego de todo lo anterior se confirma el préstamo y este se le muestra al cliente .

Base de datos

Para este módulo decidimos como equipo usar una base de datos no relacional (MongoDB) debido a los beneficios que ofrece en cuanto a flexibilidad y escalabilidad, especialmente considerando las características dinámicas del módulo de préstamos deportivos. A diferencia de las bases de datos relacionales, MongoDB nos permite trabajar con estructuras de datos más flexibles (documentos JSON), lo cual facilita la representación de préstamos que pueden variar en su estructura o contener información embebida (como datos del usuario o del artículo prestado).

Además, MongoDB se adapta mejor al crecimiento del sistema, ya que permite escalar horizontalmente sin comprometer el rendimiento. Esto es útil si en el futuro se gestionan múltiples sedes o se incrementa el volumen de préstamos. A continuación, una tabla comparativa que respalda nuestra elección:

Tabla comparativa Bases de datos			
Criterio	Base de Datos SQL	Base de Datos NoSQL	Justificación para MongoDB
Flexibilidad del esquema	Esquema fijo	Esquema dinámico	Permite modificar fácilmente los documentos sin afectar el sistema.
Escalabilidad	Vertical (limitada)	Horizontal (alta)	Ideal para sistemas que pueden crecer en usuarios y operaciones.
Modelado de datos complejos	Uso de múltiples tablas y JOIN	Documentos embebidos	Facilita consultas y evita complejidad innecesaria.
Velocidad de desarrollo	Más estructurado y riguroso	Ágil y adaptable	Rápida iteración en fases iniciales del desarrollo.
Rendimiento en consultas	Óptimo para relaciones complejas	Eficiente para consultas simples y rápidas	MongoDB es adecuado para operaciones frecuentes de lectura/escritura en tiempo real.

Manejo de Errores:

Código HTTP	**Escenario**	**Mensaje**
400 Bad Request	**Datos incompletos o inválidos**	**"Faltan campos obligatorios", "Formato incorrecto"**
401 Unauthorized	**Token no enviado o inválido**	**"Token no válido o expirado"**
404 Not Found	**Recurso no encontrado**	**"Articulo no existe"**
500 Internal Server Error	**Fallo inesperado**	**"Error interno del servidor, intente más tarde"**