



Programción Concurrente

0341403

Práctica de Programación Concurrente con Hibernate y JPA en un Sistema de Gestión de Bibliotecas

Por

Jorge Alejandro Cadrecha Del Rey

139846

DECLARACIÓN

Yo, Jorge Alejandro Cadrecha Del Rey, confirmo que el trabajo presentado en este informe es mío. Si la información procede de otras fuentes, confirmo que así se indica en el informe.

Jorge Alejandro

CONTENTS

Declaración	2
Lista de figuras	4
1 Introducción	5
1.1 Enunciado del ejercicio	5
1.2 Requisitos Técnicos.....	5
1.3 Entregables	6
1.4 Criterios de evaluación.....	7
2 Código	8
2.1 Estrutura	8
2.2 Modificaciones	8
3 Inicialización	10
3.1 Comandos.....	10
3.2 Implementaciones	10
3.3 Diseño de la base de datos	11
4 Resultados	12
5 Conclusión	15
5.1 Conclusion	15

LIST OF FIGURES

2.1	Organización del proyecto	9
3.1	Descripción de la base de datos.....	11
4.1	Inicialización del puerto	12
4.2	Docker	13
4.3	Web de lectores	13
4.4	Edición de lectores	14
4.5	Eleminación de lector.....	14

1. INTRODUCCIÓN

1.1. ENUNCIADO DEL EJERCICIO

En una biblioteca pública grande, existen miles de libros y cientos de lectores que buscan pedir prestados, devolver y renovar estos libros. Además, los bibliotecarios deben ser capaces de agregar nuevos libros al sistema, eliminar libros obsoletos o dañados, y realizar un seguimiento de los préstamos de libros. Para manejar estas tareas de forma eficiente y segura, necesitamos desarrollar un Sistema de Gestión de Bibliotecas (LMS por sus siglas en inglés) que use Hibernate y JPA para interactuar con una base de datos SQL y que pueda manejar solicitudes concurrentes de manera segura.

1.2. REQUISITOS TÉCNICOS

1. Diseñar e implementar un modelo de datos para la biblioteca. Esto debe incluir clases para libros, lectores, préstamos, y cualquier otra entidad que considere necesaria.
2. Utilice Hibernate y JPA para mapear sus clases de dominio a las tablas de la base de datos.
3. Proporcione una API que permita a los clientes (bibliotecarios y lectores) realizar las operaciones básicas de la biblioteca, como buscar libros, pedir prestados libros, devolver libros, renovar préstamos, agregar nuevos libros y eliminar libros obsoletos.(<https://www.nigmacode.com/java/crear-api-rest-con-spring/>)
4. Implemente el control de concurrencia para evitar condiciones de carrera, por ejemplo, dos lectores que intentan pedir prestado el mismo libro al mismo tiempo.
5. Implemente auditoría y control de versiones para realizar un seguimiento de quién hace qué y cuándo en el sistema.
6. Utilice una caché para mejorar el rendimiento de las operaciones comunes, como

buscar libros.

7. Utilice pruebas unitarias e integración para verificar el correcto funcionamiento de su aplicación.

1.3. ENTREGABLES

- Diseñar e implementar un modelo de datos para la biblioteca. Esto debe incluir clases para libros, lectores, préstamos, y cualquier otra entidad que considere necesaria.

- Utilice Hibernate y JPA para mapear sus clases de dominio a las tablas de la base de datos.

- Proporcione una API que permita a los clientes (bibliotecarios y lectores) realizar las operaciones básicas de la biblioteca, como buscar libros, pedir prestados libros, devolver libros, renovar préstamos, agregar nuevos libros y eliminar libros obsoletos.(<https://www.nigmacode.com/java/crear-api-rest-con-spring/>)

- Implemente el control de concurrencia para evitar condiciones de carrera, por ejemplo, dos lectores que intentan pedir prestado el mismo libro al mismo tiempo. Implemente auditoría y control de versiones para realizar un seguimiento de quién hace qué y cuándo en el sistema.

- Utilice una caché para mejorar el rendimiento de las operaciones comunes, como buscar libros.

- Utilice pruebas unitarias e integración para verificar el correcto funcionamiento de su aplicación.

1.4. CRITERIOS DE EVALUACIÓN

- Correcta implementación de la concurrencia y el manejo de datos con Hibernate y JPA.
 - Complejidad y robustez de las pruebas realizadas.
 - Claridad y calidad del código, del diagrama de la base de datos y de la documentación.
- Reflexión crítica sobre el proceso de desarrollo.
- Funcionalidad y facilidad de uso de la aplicación.

2. CÓDIGO

2.1. ESTRUCTURA

El código está estructurado en varios paquetes que contienen las clases correspondientes. Dentro del proyecto, se pueden identificar los paquetes "config", "controller", "domain", "model", "repos", "rest", "service" y "util". Además, se encuentra la clase "GestionBibliotecasApplication", que es la que debe ejecutarse para iniciar el programa. La siguiente imagen ilustra esta estructura.

2.2. MODIFICACIONES

En el proyecto, fue necesario modificar dos archivos clave para establecer la conexión con la base de datos. El primero que ajusté para su correcto funcionamiento fue el archivo "application.yml", encargado de la configuración en aplicaciones Java con Spring Boot. Este archivo se utiliza para definir propiedades de la aplicación, como configuraciones de bases de datos y perfiles de entorno, simplificando la personalización y gestión de la aplicación sin necesidad de código adicional.

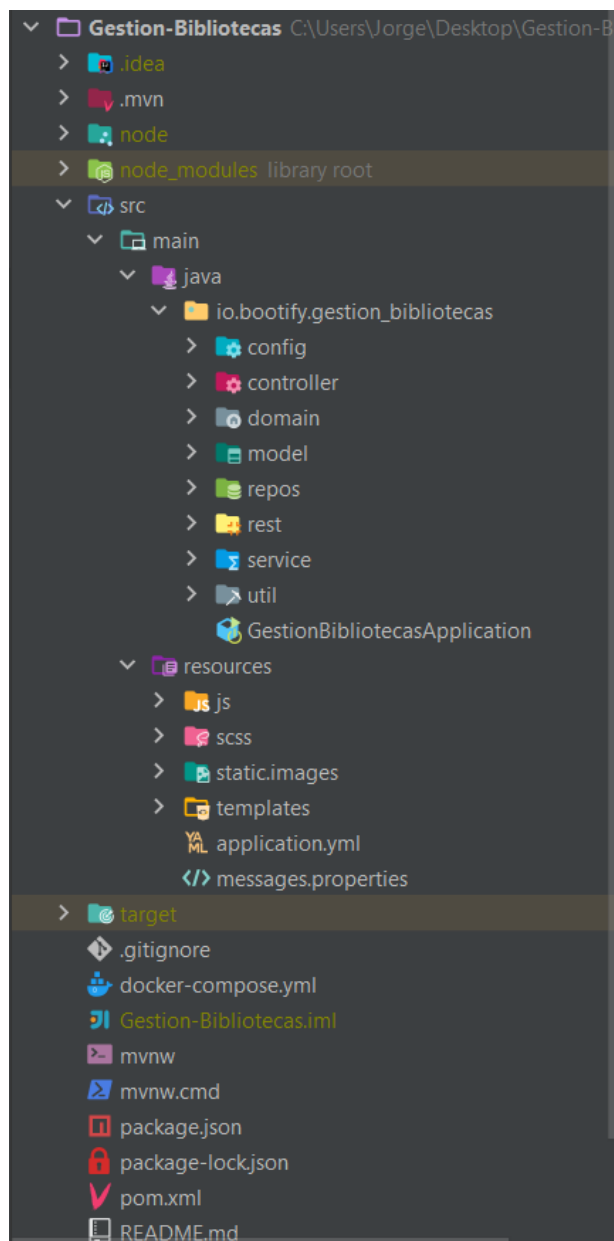


Figure 2.1: Organización del proyecto

3. INICIALIZACIÓN

Para iniciar el proyecto, lo primero que necesitamos es un entorno de desarrollo que admita la programación en Java. Posteriormente, debemos asegurarnos de tener instaladas las aplicaciones clave: PostgreSQL, encargada de la gestión de la base de datos; Docker, responsable del despliegue de la aplicación en contenedores de software. Finalmente, es necesario contar con la aplicación de Node.js para ejecutar los comandos de instalación de paquetes y para inicializar el servidor de desarrollo (DevServer).

3.1. COMANDOS

Han sido necesarios los comandos:

- `npm install`
- `npm run devserver`

3.2. IMPLEMENTACIONES

Para el funcionamiento del proyecto fueron necesarias ciertas dependencias en el `pom.xml`:

- `spring-boot-starter-web`
- `spring-boot-starter-validation`
- `spring-boot-starter-data-jpa`
- `postgresql`
- `spring-boot-starter-thymeleaf`
- `thymeleaf-layout-dialect`

- springdoc-openapi-starter-webmvc-ui
- jakarta.persistence-api
- lombok
- spring-boot-devtools
- spring-boot-docker-compose
- spring-boot-starter-test

3.3. DISEÑO DE LA BASE DE DATOS

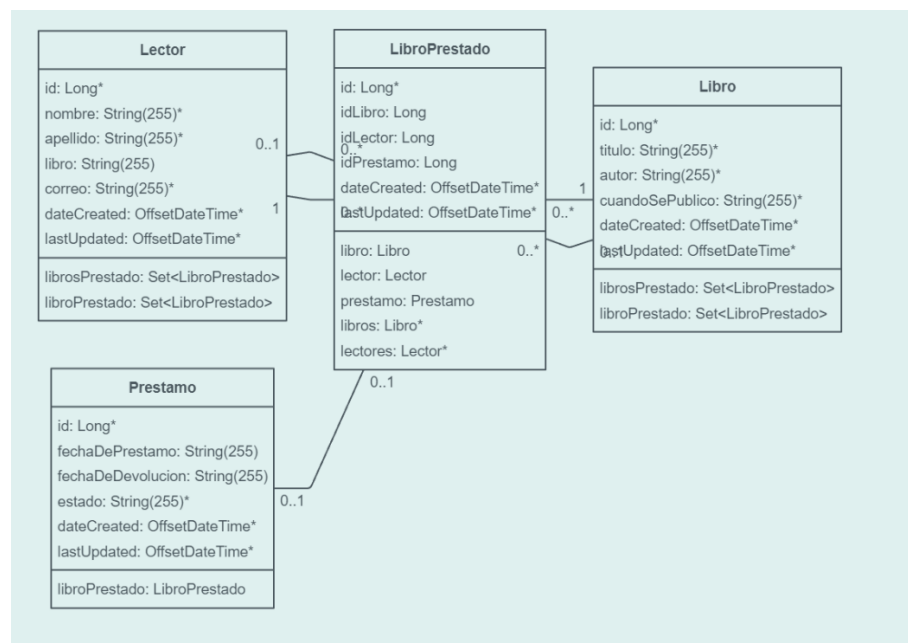
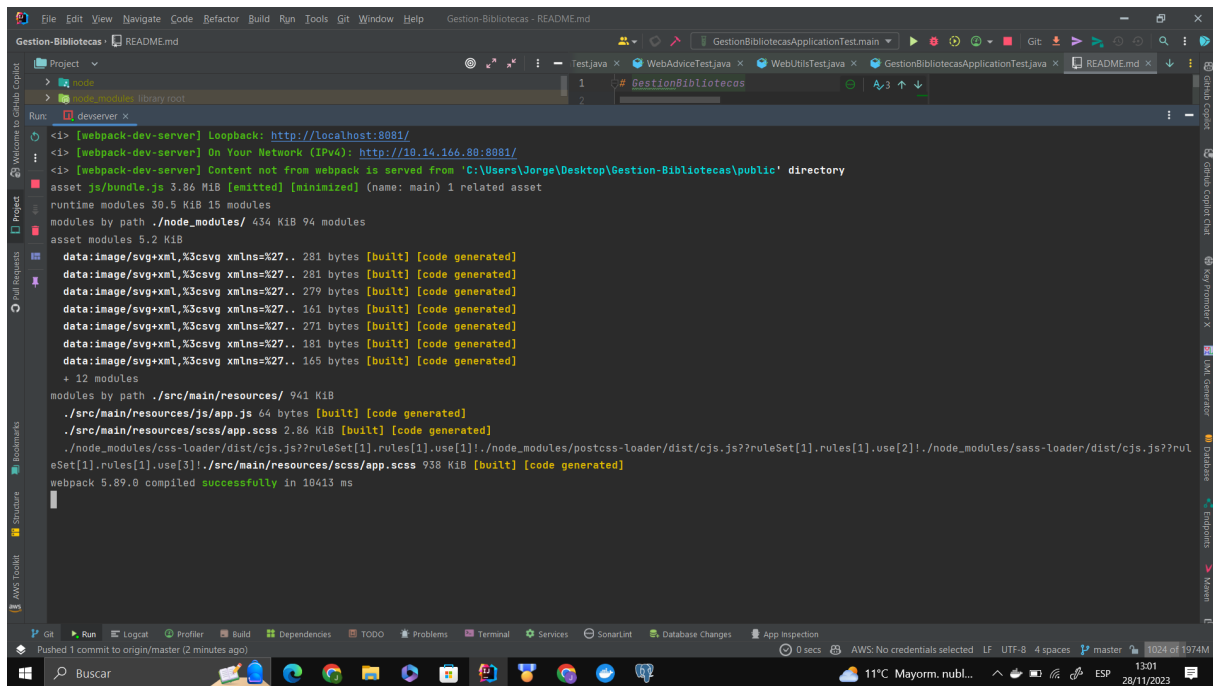


Figure 3.1: Descripción de la base de datos.

4. RESULTADOS



The screenshot shows an IDE window with the title 'Gestion-Bibliotecas - README.md'. The main editor displays the output of a Webpack Dev Server command. The output indicates that the application is running on `http://localhost:8881/` and `http://10.14.166.88:8881/`. It also shows that the content is served from the directory `'C:\Users\Jorge\Desktop\Gestion-Bibliotecas\public'`. The output lists various assets and modules, including `asset js/bundle.js 3.86 MiB [emitted] [minimized] (name: main) 1 related asset`, `runtime modules 30.5 KiB 15 modules`, `modules by path ./node_modules/ 434 KiB 94 modules`, and `asset modules 5.2 KiB`. The output also lists several data assets, such as `data:image/svg+xml,%3csvg xmlns:%27.. 281 bytes [built] [code generated]`, and modules by path `./src/main/resources/ 941 KiB`. The output concludes with `webpack 5.89.0 compiled successfully in 10413 ms`. The IDE interface includes a sidebar with 'Project', 'Run', 'Terminal', and 'Output' tabs, and a bottom status bar showing 'Pushed 1 commit to origin/master (2 minutes ago)'.

```
<i> [webpack-dev-server] Loopback: http://localhost:8881/
<i> [webpack-dev-server] On Your Network (IPv4): http://10.14.166.88:8881/
<i> [webpack-dev-server] Content not from webpack is served from 'C:\Users\Jorge\Desktop\Gestion-Bibliotecas\public' directory
asset js/bundle.js 3.86 MiB [emitted] [minimized] (name: main) 1 related asset
runtime modules 30.5 KiB 15 modules
modules by path ./node_modules/ 434 KiB 94 modules
asset modules 5.2 KiB
  data:image/svg+xml,%3csvg xmlns:%27.. 281 bytes [built] [code generated]
  data:image/svg+xml,%3csvg xmlns:%27.. 281 bytes [built] [code generated]
  data:image/svg+xml,%3csvg xmlns:%27.. 279 bytes [built] [code generated]
  data:image/svg+xml,%3csvg xmlns:%27.. 161 bytes [built] [code generated]
  data:image/svg+xml,%3csvg xmlns:%27.. 271 bytes [built] [code generated]
  data:image/svg+xml,%3csvg xmlns:%27.. 181 bytes [built] [code generated]
  data:image/svg+xml,%3csvg xmlns:%27.. 165 bytes [built] [code generated]
+ 12 modules
modules by path ./src/main/resources/ 941 KiB
  ./src/main/resources/js/app.js 64 bytes [built] [code generated]
  ./src/main/resources/scss/app.scss 2.86 KiB [built] [code generated]
  ./node_modules/css-loader/dist/cjs.js??ruleSet[1].rules[1].use[1]!./node_modules/postcss-loader/dist/cjs.js??ruleSet[1].rules[1].use[2]!./node_modules/sass-loader/dist/cjs.js??ruleSet[1].rules[1].use[3]!./src/main/resources/scss/app.scss 938 KiB [built] [code generated]
webpack 5.89.0 compiled successfully in 10413 ms
```

Figure 4.1: Inicialización del puerto

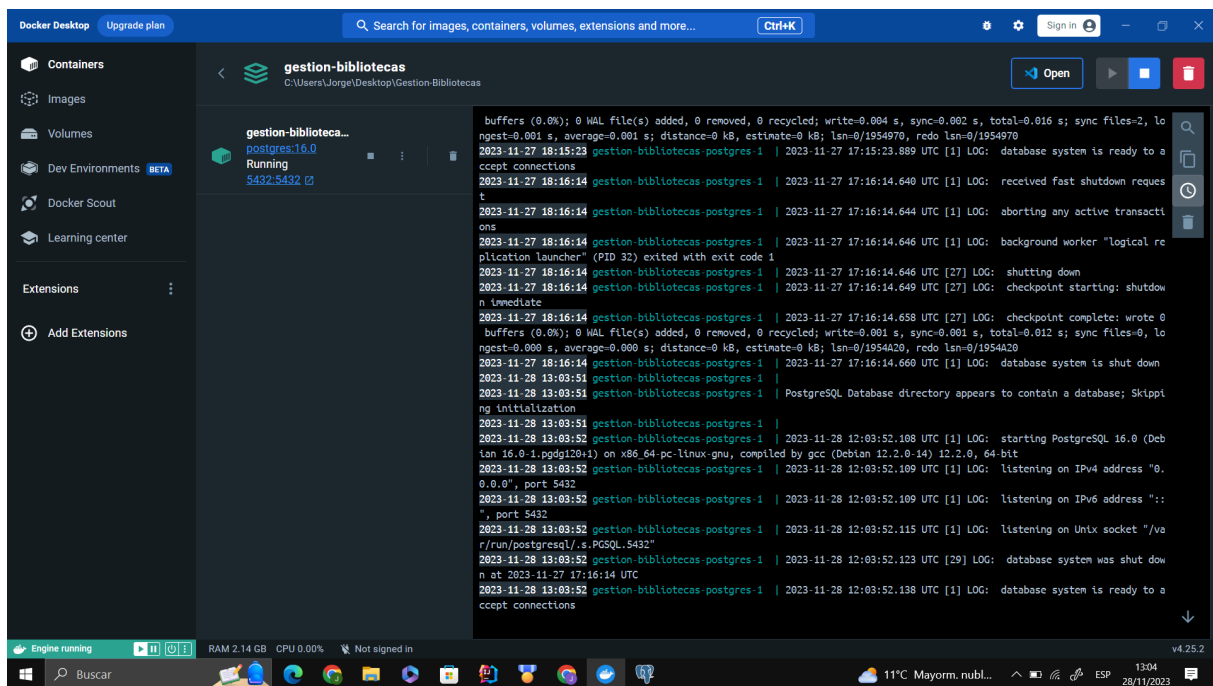


Figure 4.2: Docker

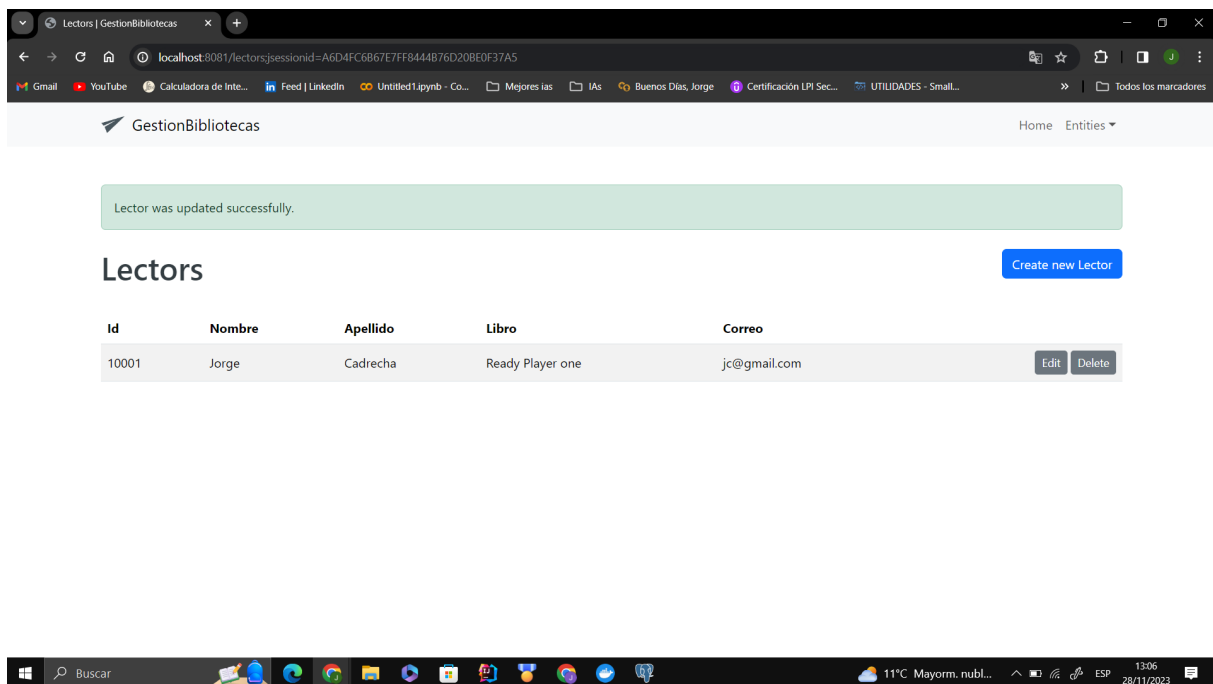


Figure 4.3: Web de lectores

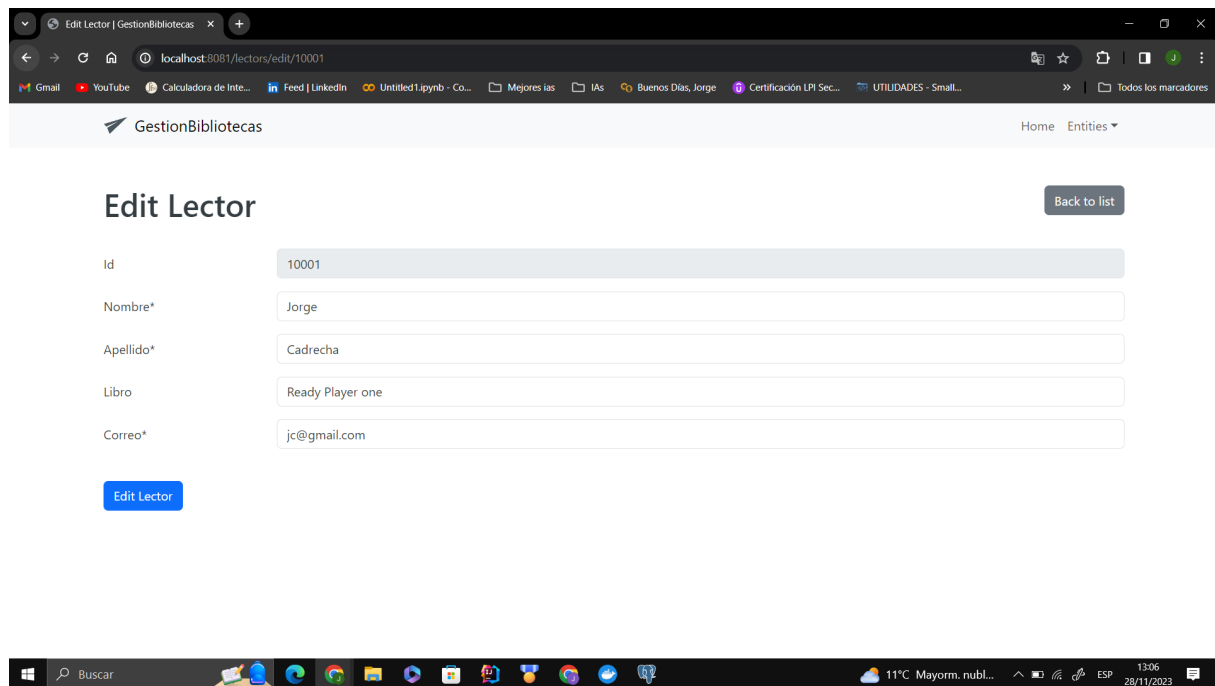


Figure 4.4: Edición de lectores

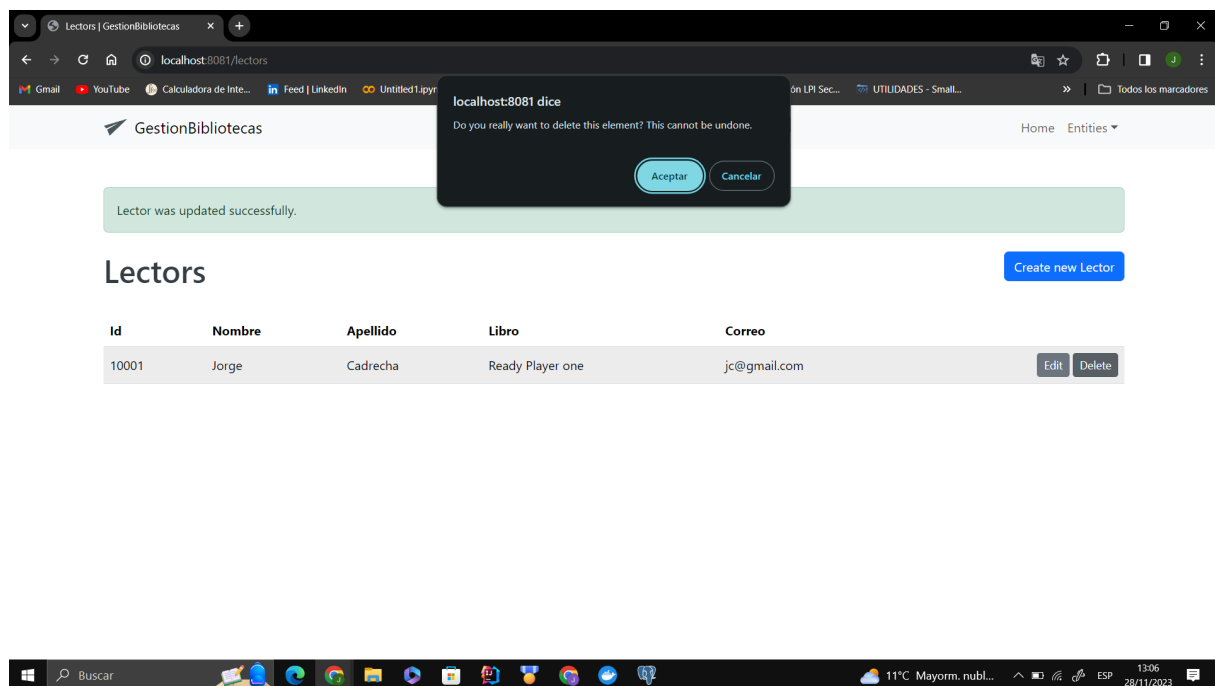


Figure 4.5: Eliminación de lector

5. CONCLUSIÓN

5.1. CONCLUSION

En resumen, he adquirido conocimientos sobre la generación de código necesario para desarrollar una aplicación de gestión de bibliotecas en línea utilizando SQL. Además, he aprendido los pasos necesarios para la inicialización del proyecto, lo cual incluye la investigación necesaria para su desarrollo. Durante este proceso, también he adquirido habilidades en el uso de aplicaciones como PostgreSQL y Docker para garantizar una correcta puesta en marcha del sistema.