



PROYECTO FIN DE CICLO

DAW

EzLib

Fase: 3: Entrega final
Swann Julien

Los documentos, elementos gráficos, vídeos, transparencias y otros recursos didácticos incluidos en este contenido pueden contener imprecisiones técnicas o errores tipográficos. Periódicamente se realizan cambios en el contenido. Fomento Ocupacional FOC SL puede realizar en cualquier momento, sin previo aviso, mejoras y/o cambios en el contenido.

Es responsabilidad del usuario el cumplimiento de todas las leyes de derechos de autor aplicables. Ningún elemento de este contenido (documentos, elementos gráficos, vídeos, transparencias y otros recursos didácticos asociados), ni parte de este contenido puede ser reproducida, almacenada o introducida en un sistema de recuperación, ni transmitida de ninguna forma ni por ningún medio (ya sea electrónico, mecánico, por fotocopia, grabación o de otra manera), ni con ningún propósito, sin la previa autorización por escrito de Fomento Ocupacional FOC SL.

Este contenido está protegido por la ley de propiedad intelectual e industrial. Pertenecen a Fomento Ocupacional FOC SL los derechos de autor y los demás derechos de propiedad intelectual e industrial sobre este contenido.

Sin perjuicio de los casos en que la ley aplicable prohíbe la exclusión de la responsabilidad por daños, Fomento Ocupacional FOC SL no se responsabiliza en ningún caso de daños indirectos, sean cuales fueren su naturaleza u origen, que se deriven o de otro modo estén relacionados con el uso de este contenido.

© 2023 Fomento Ocupacional FOC SL todos los derechos reservados.

Índice

1 Estudio inicial previo a la realización del proyecto.....	4
1.1. Clasificar las empresas del sector por sus características organizativas y el tipo de producto o servicio que ofrecen.....	4
1.2. Poner un ejemplo de estructura organizativa para una empresa del sector.....	4
1.3. Identificar las necesidades demandas que cubre el proyecto y asociarlas con las necesidades del cliente.....	5
1.4. Descripción del proyecto.....	5
1.5. Justificar el tipo de proyecto elegido para dar solución al problema.....	6
1.6. Características principales del proyecto elegido.....	6
2. Identificación de necesidades y diseño del proyecto.....	8
2.1. Estudio inicial y planificación del proyecto.....	8
2.2. Aspectos fiscales y laborales.....	13
2.3. Viabilidad económica.....	14
2.4. Modelo de solución.....	17
3. Ejecución del proyecto y pruebas.....	25
3.1. Riesgos de ejecución del proyecto.....	25
3.2. Documentación de ejecución.....	26
3.3. Incidencias.....	57
4. Pruebas y soporte.....	59
4.1. Crear documento con las pruebas a realizar.....	59
4.2. Registro de las pruebas realizadas.....	60
4.3. Evaluar que el proyecto cumple todo lo requerido.....	60

EzLib / DAW

1 Estudio inicial previo a la realización del proyecto

1.1. Clasificar las empresas del sector por sus características organizativas y el tipo de producto o servicio que ofrecen

Las empresas que ofrecen programas de gestión de prestamos para bibliotecas escolar suelen ser de dos tipos:

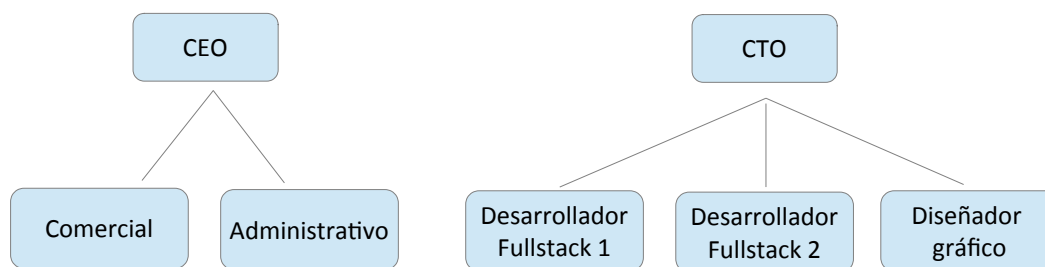
1. Empresas medianas o grandes, de 50 o más trabajadores, que proponen su producto a nivel internacional. Esas empresas no suelen vender solo programas de gestión de prestamos para bibliotecas sino que también han desarrollado y venden otros software como ERPs por ejemplo. Se puede encontrar el caso también de que este tipo de empresa proponga servicios: almacenaje en la nube o de mantenimiento de las infraestructuras IT por ejemplo.
2. Empresas pequeñas de menos de 50 empleados y que venden su producto principalmente en el mercado nacional. Puede ser incluso microempresas compuesta de dos o varios socios. Este tipo de empresa no tiene las competencias humanas ni la estructura organizativa, al principio, para vender su producto fuera de España.

Que sea un caso o el otro, se trata de empresas de desarrollo de software que crean y comercializan un programa informático que aporta una solución a una problemática y así responde a una demanda del mercado.

1.2. Poner un ejemplo de estructura organizativa para una empresa del sector

EzLib es una micro empresa española que cuenta con dos socios, ambos ingenieros de software, un técnico-comercial, un administrativo, dos desarrolladores Fullstack y un desarrollador Frontend especializado en diseño gráfico. EzLib está registrada en el Registro Mercantil cómo Sociedad Limitada y cuenta con un capital social de 10.000€.

Cada socio gestiona una area del negocio. Uno es el CEO y responsable del area comercial y administrativo mientras que el otro es el CTO y gestiona el equipo de desarrolladores. Se puede ver abajo el organigrama de la empresa.



1.3. Identificar las necesidades demandas que cubre el proyecto y asociarlas con las necesidades del cliente.

EzLib ha tenido la oportunidad que un colegio británico de la Costa del Sol, el Sunny View, le confié la responsabilidad de desarrollar e implementar un programa de gestión de los préstamos para la biblioteca del colegio.

La biblioteca del Sunny View ha crecido mucho esos últimos años. Cada año se ha comprado libros nuevos además de recibir donaciones por parte de las familias de los alumnos. En total entre las clases de primarias y de secundarias, la biblioteca cuenta ahora con más de 3.000 libros.

Hasta ahora los préstamos se gestionaban de forma manual, a través de tablas Excel donde se guardaba un registro de los libros que se habían prestado a los alumnos. Pero con el tiempo este sistema ha llegado a su límite. Se han perdidos muchos libros y hay préstamos que no se han registrado correctamente.

Esta problemática no es particular al Sunny View. La gran mayoría de los colegios en España no cuentan con un sistema informático de gestión de los préstamos. Así que la solución que se va a desarrollar se podría adaptar perfectamente a cualquier colegio que tenga la misma problemática.

Según el Ministerio de Educación y Formación Profesional, España cuenta con 11.000 centros docentes privados no universitario y 22.500 centros públicos¹. Habría que profundizar la búsqueda y determinar cuantos de esos centros disponen de una biblioteca escolar y también determinar el tamaño crítico de número de alumnos a partir del cual este programa puede ser interesante para una escuela. Pero ya podemos ver que a primera vista, hay un potencial de 30.000 centros donde podríamos comercializar este programa.

1.4. Descripción del proyecto.

El objetivo principal de este proyecto es de aportar al Sunny View school una solución a medida para la gestión diaria de los préstamos de su biblioteca.

Se trata de una aplicación web que permite gestionar los préstamos a través de un sistema de alertas

1 <https://www.educacion.gob.es/centros/buscarCentros>

que se desencadenará cuando un libro tiene que haber sido devuelto.

Más allá de esto, se trata también de aportar una solución para ver, buscar y editar digitalmente los libros de la biblioteca. Así, los alumnos podrán buscar libros según varios campos de búsqueda. La bibliotecaria podrá definir palabras claves para cada libro y que servirán después para las búsquedas (ej. "ficción", "2ºESO" etc).

Otra funcionalidad que aportará esta aplicación web es la posibilidad de sacar estadísticas que ayudarán la bibliotecaria y el equipo pedagógico a tomar decisiones. Por ejemplo se podrá saber el número de veces que un libro ha sido prestado, el alumno o la clase qué más veces ha pedido prestado.

Este solución beneficiará a todo el colegio en general pero sobre todo a la bibliotecaria. Le agilizará la gestión de los prestamos ya que no tendrá que tener un registro de esos por tablas Excel sino que se escaneará los libros a la entrada y a la salida de la biblioteca. Esto permitirá liberarle tiempo que podrá dedicar a tareas que aportan más valor a la biblioteca y al colegio.

1.5. Justificar el tipo de proyecto elegido para dar solución al problema.

Este proyecto se va a desarrollar como una aplicación web. A través de un URL, el usuario, sea el administrador o un alumno podrá acceder, con un navegador web y una conexión a internet, a la aplicación.

Tras autenticarse, el usuario tendrá ciertos privilegios según las reglas de seguridad que se hayan creado. Los alumnos podrán solamente consultar los libros y reservarlos mientras que el administrador tendrá todo el control sobre la base de datos de la biblioteca (consultar, editar, añadir, eliminar).

De esta forma, no se requiere descargar una aplicación de escritorio lo que hace que esta solución sea más ágil y permite usarla desde cualquier dispositivo.

1.6. Características principales del proyecto elegido.

Para resumir, esta aplicación web permitirá:

- Tener un registro interno de todos los libros de la biblioteca
- Hacer búsqueda avanzada según varios criterios: título, autor, categoría, etc
- Ver si un libro está disponible o no y si no lo es cuándo tiene que estar de vuelta
- Reservar un libro cuando esté de vuelta
- Añadir, actualizar, eliminar libros

- Generar un código QR que identifique el libro al añadirlo
- Lanzar una alerta cuando el libro no ha sido devuelto a tiempo
- Enviar un correo electrónico automáticamente al alumno que no ha devuelto el libro
- Ver todos los libros que han sido prestados, por quién así que su fecha de devolución
- Guardar un registro de todos los prestamos y así poder sacar estadísticas sobre ellos

2. Identificación de necesidades y diseño del proyecto

2.1. Estudio inicial y planificación del proyecto

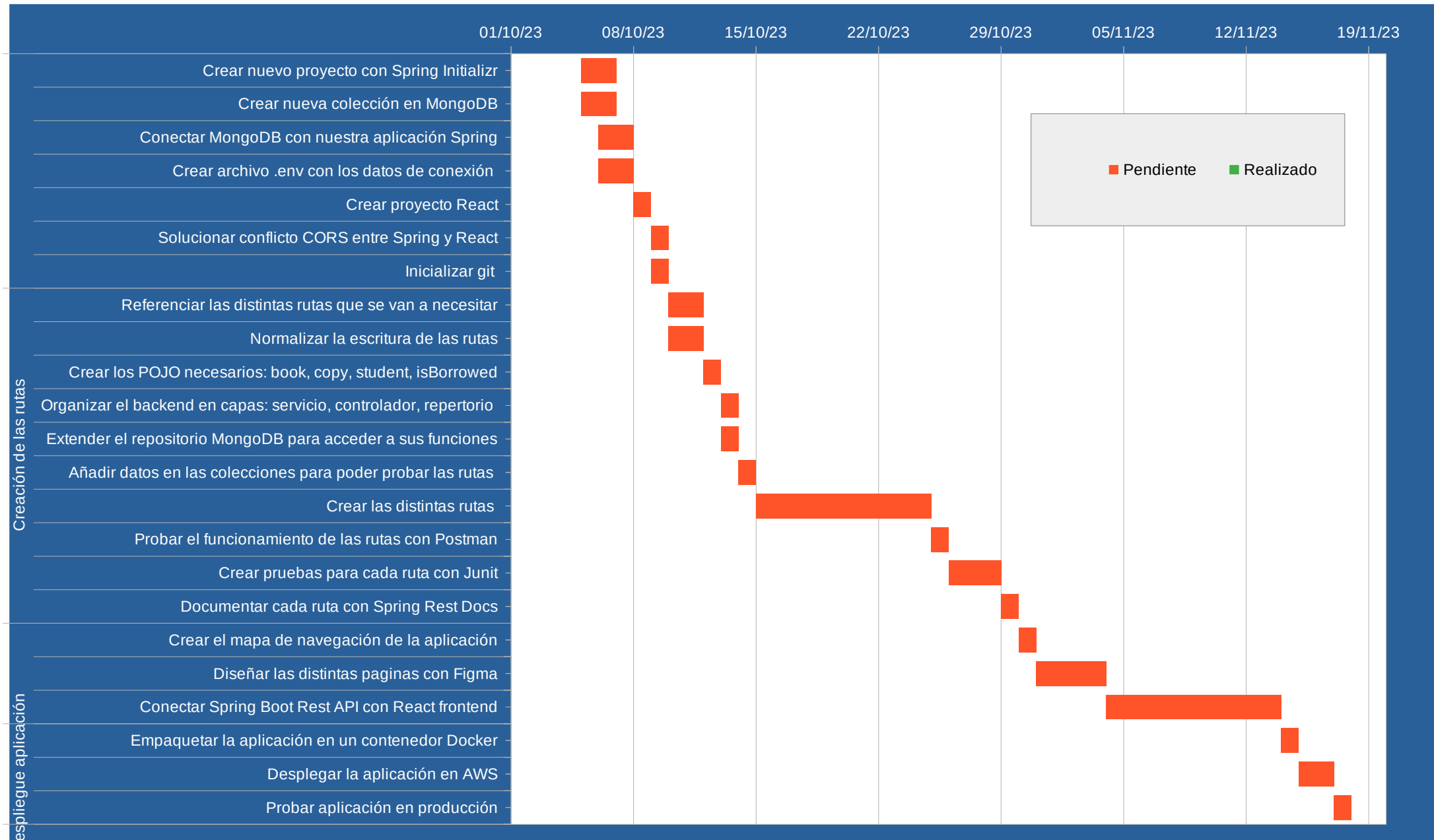
- 2.1.1. Identificar las fases del proyecto y su contenido.

Abajo se puede ver un diagrama de Gantt que muestra con todas la fases de desarrollo y de despliegue de la aplicación clasificado por orden de ejecución en el tiempo. Se adjunta también en anexo la hoja de calculo de este diagrama.

En este momento todo está por hacer o en estado “pendiente” como se puede ver en el diagrama.

Para este proyecto se han identificado cuatro fases:

- **Inicio del proyecto:** fase donde se prepara todas las herramientas requeridas para empezar a trabajar.
- **Creación de las rutas:** fase de desarrollo de la parte backend con Spring Boot. La aplicación se va a crear como micro servicios de tipo REST y es con Spring Boot que se creará las distintas rutas con su lógica.
- **Diseño y desarrollo de la interfaz grafica:** fase de desarrollo de la parte frontend y enlace a las rutas previamente creadas.
- **Despliegue de la aplicación:** fase de despliegue en un servidor y comprobación que la aplicación funciona bien en producción.



- 2.1.2. Especificar los objetivos del proyecto.
 - Crear una base de datos para almacenar los libros, los ejemplares, los alumnos así que el estado del libro.
 - Guardar un registro en la base de datos de todas las operaciones de prestamos y devoluciones para luego sacar estadísticas.
 - Crear una API de tipo REST que permita crear, leer, actualizar o eliminar datos en la base de datos.
 - Referenciar todas las rutas que se van a necesitar.
 - Desarrollar las rutas en la parte backend de la aplicación.
 - Llamar a la API de OpenLibrary para obtener información sobre un libro.
 - Modificar la información recibida por la API de OpenLibrary y guardar en la base de datos solo la que se necesita para su funcionamiento.
 - Desarrollar la posibilidad de crear un libro manualmente y añadirlo a la base de datos.
 - Llamar la API de GoQR.me para generar un código QR único para cada ejemplar que se añade a la base de datos.
 - Desarrollar una funcionalidad que permite leer un código QR para identificar el ejemplar al que pertenece en la base de datos.
 - Crear las operaciones CRUD sobre la tabla del estado de un libro. Este paso es el núcleo central que gestiona los prestamos y devoluciones.
 - Tomar en cuenta todos los mensajes de error posibles y guiar el usuario según el mensaje recibido.
 - Crear dos perfiles de usuario: Administrador y Alumno.
 - Definir los privilegios para cada tipo de usuario.
 - Crear reglas de seguridad para la conexión a la aplicación para cada tipo de usuario.
 - Desarrollar una pantalla de estadísticas a partir del registro de los prestamos: libro más leído, libro menos leído, mejor lector, clase que ha hecho más prestamos.
 - Diseñar la interfaz gráfica tomando en cuenta la parte Usuario destinada a los alumnos y la parte Administrador destinada a la bibliotecaria.
 - Crear al aplicación de forma responsive, tomando en cuenta todos los tipos de dispositivo.
 - Crear una función de búsqueda avanzada de libros según ciertos criterios.
 - Poner en marcha un sistema de alerta y de control de los libros que deberían haberse devuelto.

- 2.1.3. Especificar recursos hardware y software.

Los recursos hardware y software necesarios para desarrollar este proyecto son los siguientes:

- 2.1.3.1. Recursos hardware:
 - 3 ordenadores de mesa
 - 6 pantallas (2 por cada ordenador)
 - 1 router con conexión a internet
 - 1 servidor

- 2.1.3.2. Recursos software
 - Visual Studio Code
 - IntelliJ IDEA
 - Postman

- Chrome
 - Firefox
 - Figma
 - Photoshop
 - JDK Java 17
 - JUnit 5
 - MongoDB Atlas
 - MongoDB Compass
 - React
 - Docker
 - AWS
- 2.1.4. Especificar recursos materiales y personales.
 - **Recursos materiales**

Para desarrollar su actividad, EzLib necesita los recursos materiales siguiente:

 - una oficina de unos 80m2
 - siete ordenadores de escritorio, uno por cada puesto de trabajo
 - once pantallas: cada desarrollador así que el CTO necesitan dos de ellas
 - siete escritorios con siete sillas de oficina
 - material de oficina: bolígrafos, cuadernos, grapadoras etc.
 - una pizarra blanca grande
 - una impresora
 - un teléfono fijo
 - un teléfono móvil
 - una sala de reunión con:
 - una mesa grande
 - 8 sillas
 - un mini PC conectado a una pantalla grande
 - una webcam gran angular + micrófono
 - dos ordenadores portátiles para las visitas a clientes
 - una cocinita equipada de:
 - una maquina de café con su mueble
 - una nevera
 - un fregadero
 - vajilla básica
 - **Recursos personales**

Como comentado anteriormente, la empresa fue fundada por dos socios, cada uno ocupando un puesto a responsabilidad. Para llevar a cabo su actividad EzLib necesita contratar el personal siguiente:

 - dos desarrolladores fullstack
 - un diseñador gráfico con buenos conocimientos de las tecnologías de frontend
 - un comercial encargado de buscar nuevos clientes para las aplicaciones desarrolladas por EzLib
 - un administrativo que tendría un puesto transversal, capaz de ayudar a cualquier miembro del equipo con tareas administrativas o de soporte

- 2.1.5. Realizar una asociación de fases y recursos materiales que deben intervenir en cada fase
- 2.1.6. Realizar una asociación de fases y recursos humanos que deben intervenir en cada fase

Fases de implementación	Recursos materiales	Recursos personales
Inicio del proyecto	Oficina con todas sus comodidades 3 ordenadores 6 pantallas Intellij Idea Java 17 MongoDB Atlas MongoDB Compass	2 desarrolladores fullstack
Creación de las rutas	Oficina con todas sus comodidades 3 ordenadores 6 pantallas Intellij Idea Java 17 MongoDB Atlas MongoDB Compass Postman JUnit 5	2 desarrolladores fullstack
Diseño y desarrollo de la interfaz grafica	Oficina con todas sus comodidades 1 ordenador 2 pantallas Photoshop Figma Visual Studio Code React	1 diseñador gráfico 1 desarrollador fullstack
Despliegue de la aplicación	Oficina con todas sus comodidades 1 ordenador	1 desarrollador fullstack

	2 pantallas AWS Docker	
Comercialización de la aplicación	Oficina con todas sus comodidades 1 ordenador 1 pantalla 2 ordenadores portátiles 1 teléfono móvil	1 comercial

2.2. Aspectos fiscales y laborales

- 2.2.1 Obligaciones fiscales

EzLib como persona jurídica tiene que contribuir al sistema tributario. Esto se conoce como las obligaciones tributarias o obligaciones fiscales de las empresas. Esas obligaciones intervienen en tres momentos:

- **Iniciales:** las obligaciones que se producen como consecuencia de la constitución y puesta en funcionamiento de la empresa: alta en el Impuesto de Actividades Económicas (IAE), declaración censal, Impuesto de Transmisiones Patrimoniales y Actos Jurídicos Documentados .
- **De gestión:** las obligaciones que se producen en el desarrollo de la actividad empresarial: obligaciones contables por ejemplo.
- **De liquidación:** las obligaciones que se producen a la baja de la empresa en el sistema fiscal y que son: la baja en el censo y en el IAE, el pago del Impuesto de Transmisiones Patrimoniales y Actos Jurídicos Documentados.

También hay que tomar en cuenta la personalidad jurídica de la empresa. EzLib, al ser una Sociedad Limitada tiene la obligación de tributar al Impuesto de Sociedades y está sometida al Impuesto del Valor Añadido.

- 2.2.2 Obligaciones laborales

Cualquier empresa, con su relación con sus empleados, está sometida a respetar las siguientes obligaciones:

- Inscripción de la empresa en la Seguridad Social
- Notificar a la Autoridad Laboral pertinente que han creado un centro de trabajo
- Dar de alta los nuevos trabajadores a la Seguridad Social
- Pagar los salarios al final del mes

- Establecer una nomina al final del mes
- Realizar el abono de seguros sociales y retenciones del IRPF
- Obligación de protección de los datos de los empleados
- Contar con un plan de prevención de riesgos laborales
- Registrar los horarios diarios de trabajo de sus trabajadores
- Exponer el calendario laboral anual en cada centro de trabajo

- 2.2.3 Prevención de riesgos laborales

EzLib es una empresa de creación de software y desarrolla su actividad en la tercera planta de una oficina ubicada en un inmueble en Málaga y cuenta con 7 empleados.

Por la actividad que lleva a cabo EzLib, se identifican principalmente los riesgos siguientes:

- Riesgo para la salud de los trabajadores al estar sentado mucho tiempo
- Riesgo de incendios en el edificio
- Riesgo de sobre carga de trabajo

Con el fin de prevenir y reducir los riesgos aferentes a la producción de la empresa, se nombra a un delegado de prevención encargado entre otras cosas de vigilar y controlar que se cumple la normativa de prevención de riesgos.

También, para cada uno de los riesgos identificados se especifica medidas concretas de reducción y de eliminación de los riesgos así que medidas de protección de los empleados.

2.3. Viabilidad económica

- 2.3.1. Realizar un presupuesto económico del proyecto.

El presupuesto se ha construido según el número de hora estimadas que se necesitará para desarrollar cada fase del proyecto: backend, frontend, testing y despliegue.

Para cada una de esas fases, se ha tomado en cuenta los costes que entran en cuenta para desarrollar esta fase.

Cada coste se reparte equitativamente según el número de horas que realmente se va a usar para el proyecto. Entonces el primer paso ha sido de calcular los costes por hora de cada recursos.

Por ejemplo, IntelliJ Idea tiene un coste anual de 600€, y se sabe que se trabajan realmente 1976 horas al año, entonces este software tiene un coste por hora de $600/1976 = 0,30€$.

El número de horas laborales al año se calcula de la manera siguiente:

$(365 \text{ días al año} - 106 \text{ días de fin de semana} - 12 \text{ días festivos}) * 8 \text{ horas trabajado al día} = 1976 \text{ horas}$

El número de horas laborales al mes se calcula de la siguiente manera:

$(247 \text{ días laborales al año} / 12) * 8 \text{ horas} = 168 \text{ horas}$.

Para terminar, en este presupuesto se considera que el mobiliario y los ordenadores entran dentro de un plan de amortización anual de 5 años. Es decir que los 3 ordenadores han tenido un coste total de 6.000€ repartidos en 5 años o sea 1.200€/año.

En total este proyecto tiene un coste de 16.716,40€.

Fases	Horas dedicadas
Backend	160
Frontend	120
Testing	40
Despliegue	40

	Anual	Mensual
Días laborales	247	21
Horas laborales	1976	168

	Coste anual	Coste mensual	Coste por hora	Backend	Frontend	Testing	Despliegue	Total
Amortización ordenadores	1.200,00 €		0,61 €	97,60 €	73,20 €	24,40 €	24,40 €	219,60 €
Amortización mobiliario	1.000,00 €		0,51 €	81,60 €	61,20 €	20,40 €	20,40 €	183,60 €
Alquiler oficina		1.000,00 €	5,95 €	952,00 €	714,00 €	238,00 €	238,00 €	2.142,00 €
Electricidad		100,00 €	0,60 €	96,00 €	72,00 €	24,00 €	24,00 €	216,00 €
Intellij Idea Ultimate	600,00 €		0,30 €	48,00 €		12,00 €		60,00 €
MongoDB Dedicated		57,00 €	0,34 €	54,40 €		13,60 €		68,00 €
Creative Cloud		55,00 €	0,33 €		39,60 €	13,20 €		52,80 €
Figma profesionnal		12,00 €	0,07 €		8,40 €	2,80 €		11,20 €
Fullstack developer 1	40.000,00 €		20,24 €	3.238,40 €	2.428,80 €	809,60 €	809,60 €	7.286,40 €
Fullstack developer 2	40.000,00 €		20,24 €	3.238,40 €		809,60 €		4.048,00 €
Diseñador gráfico	30.000,00 €		15,18 €		1.821,60 €	607,20 €		2.428,80 €
			TOTAL	7.806,40 €	5.218,80 €	2.574,80 €	1.116,40 €	16.716,40 €

- 2.3.2. Identificar la financiación necesaria.

	Importe
Financiación propia	
• Aporte inicial de los socios	10.000€
• Resultado de las ventas	2.000€
Financiación externa	
• Prestamos bancarios	5.000€

- 2.3.3. Detallar posibles ayudas y subvenciones

- **A nivel estatal**

El Instituto de Crédito Oficial en su plan Empresas y Emprendedores finanza a través de unos “prestamos a los autónomos, las entidades públicas y privadas (empresas, fundaciones, ONG’s, Administración Pública), que realicen su actividad empresarial en España para desarrollar proyectos de inversión y/o las necesidades generales de la actividad”.

Esos préstamos se pueden conceder para una gran variedad de necesidades: necesidad de liquidez, adquisición de activos fijos, reforma de instalaciones, adquisición de empresa, etc.

Véase: <https://www.ico.es/ico-empresas-y-emprendedores>

- **A nivel de los ayuntamientos**

Los ayuntamientos también pueden ayudar y subvencionar a las empresas por ejemplo en tema de asesoría, de ayuda a la creación de nuevas empresas, de ayuda a negocios cuya actividad económica esta impactada por obras publicas, etc.

Un ejemplo claro de este tipo de iniciativa es la función que desempeña Promálaga que se define en su página web como “una empresa del Ayuntamiento de Málaga que trabaja en la promoción empresarial, la creación de empleo y la atracción de talento”.

En el apartado ayudas y subvenciones de su página web, podemos ver algunos ejemplos de proyectos que apoya el ayuntamiento de Málaga.

Véase <https://www.promalaga.es/malaga-emprende/#ayudas>

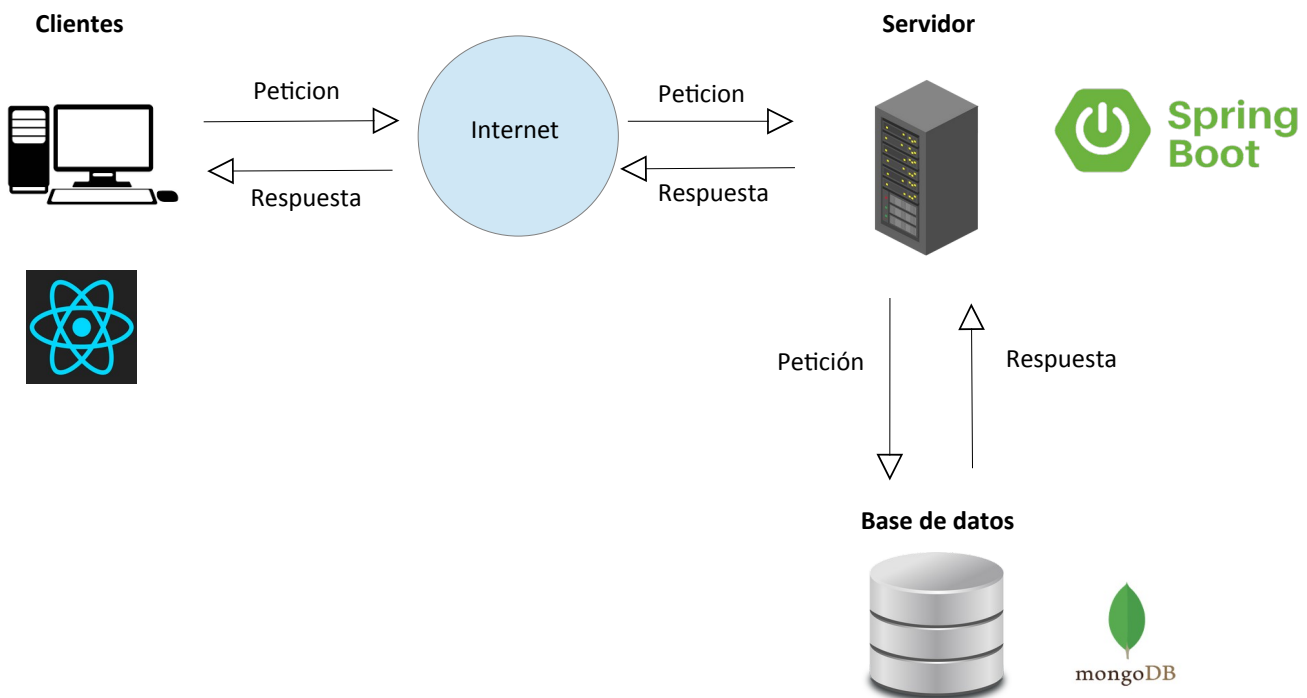
2.4. Modelo de solución

▪ 2.4.1. Modelado de la solución

✓ 2.4.1.1 Estructura general y tecnologías

Este proyecto se va a desarrollar con Spring Boot, un framework de Java, para la parte del backend. MongoDB para la base de datos. React para la parte del frontend. Concretamente se va a crear una API de tipo REST con Spring Boot, desarrollando una serie de rutas que permitirán hacer las llamadas necesarias a la base de datos. Spring Boot se encargara de la lógica y de comprobar que los datos estén correctos para cada una de esas rutas, antes de enviarlos al cliente.

Abajo se puede ver el diagrama general del proyecto con las tecnologías empleadas.



✓ 2.4.1.2 Representación gráfica de la base de datos

Antes de empezar a entrar en la lógica de la aplicación, es importante entender cuales son los datos que se va a necesitar y como se va a estructurar la base de datos.

Como comentado antes, la base de datos de este proyecto se va a construir con MongoDB, una base de datos NoSQL. Esto permite más flexibilidad a la hora de almacenar los datos y lanzar consultas.

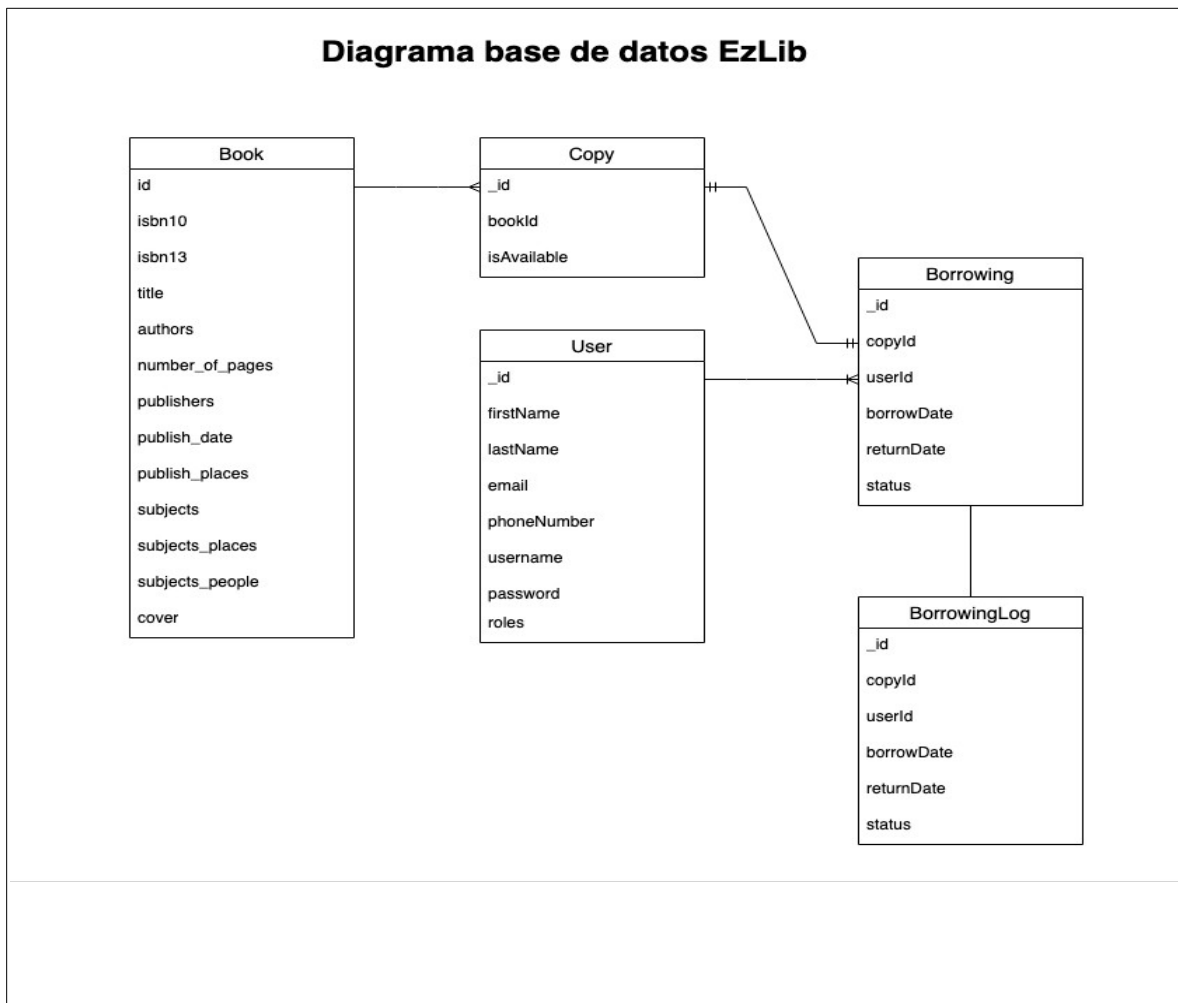
Básicamente vamos a necesitar cinco colecciones:

- **Book:** almacena todos los datos acerca de un libro (isbn, titulo, autores etc)
- **Copy:** almacena todos los ejemplares de un libro. Es el objeto físico que se prestará. Un libro puede tener 1 o más ejemplares.
- **User:** almacena toda la información sobre los usuarios. También en esta colección se

define el rol de un usuario que usaremos luego para definir los privilegios de cada uno de los usuarios.

- **Borrowing**: almacena los prestamos relacionando un usuario a un ejemplar prestado. Se almacena también la fecha del préstamo así que la fecha de devolución.
- **BorrowingLog**: almacena todas las operaciones de préstamo y de devolución de los libros. Es esta colección que se usará luego para sacar estadísticas sobre los prestamos.

Se puede ver abajo el diagrama representando la base de datos:



✓ 2.4.1.3. Presentación del diagrama de clase

◦ 2.4.1.3.1. Explicaciones generales

Toda la parte del backend de esta aplicación se va a desarrollar como una API Restful con el framework Spring Boot del lenguaje de programación Java. En otros términos, se van a crear distintas rutas que harán todas las operaciones que un usuario puede necesitar para cada operación con la base de datos. Esas rutas servirán en un segundo paso para estar consumida en los componentes que formarán la interfaz de usuario.

Para que esta aplicación sea lo más flexible y escalable posible, se va a seguir el modelo Controller – Service – Repository. Este modelo facilita también el testeo de cada funcionalidad.

Cada capa del modelo tiene las responsabilidades siguientes:

- **Controler**: expone las funcionalidades de la aplicación a través de sus rutas. Llama a un servicio para cada ruta.
- **Service**: define toda la lógica de la aplicación: qué se hace con los datos, cómo se

manipulan.

- **Repository**: almacena y llama a los datos que la capa de servicio necesita.

Para que la aplicación sea lo más flexible posible, se ha definido varios controladores:

- **BookController**: encargado de las operaciones relacionadas con los libros y sus ejemplares. Es el controlador que se usará principalmente para consultar todos los libros de la biblioteca.
- **UserController**: encargado de las operaciones relacionadas con los datos de los usuarios.
- **BorrowingController**: encargado de las operaciones relacionadas con los prestamos y el historial de los prestamos. Esta parte de la aplicación es el núcleo central que gestiona los prestamos.
- **ReportController**: encargado de enviar los datos necesarios al cliente para poder sacar estadísticas.

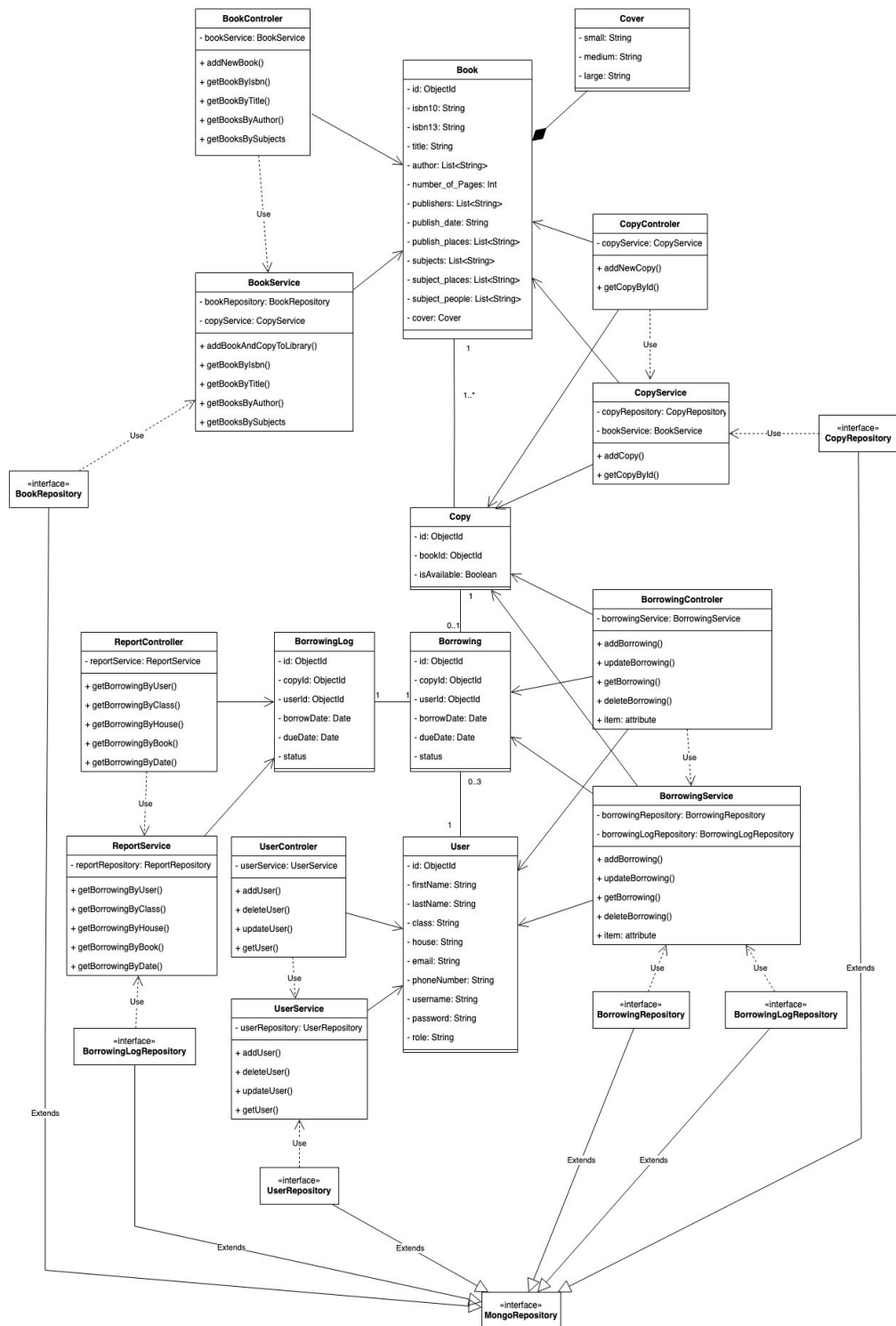
◦ 2.4.1.3.2. Limites de este modelo

Este diagrama no toma en cuenta la parte de autenticación de los usuarios. Es una funcionalidad que se implementará mas adelante.

◦ 2.4.1.3.3. Diagrama de clases

A continuación, se presenta el diagrama de clases.

Diagrama de clase EzLib



✓ 2.4.1.4. Lista de rutas

BookController:

- POST /api/books
=> añadir un libro a la biblioteca.
- GET /api/books/{isbn}
=> obtener información sobre un libro cuyo isbn aparece en parámetros.
- GET /api/books/{title}
=> obtener información sobre un libro cuyo titulo aparece en parámetros.
- GET /api/books/{author}
=> obtener información sobre los libros cuyo author aparece en parámetros.
- GET /api/books/{subject}
=> obtener información sobre los libros cuyo tema aparece en parámetros.
- PATCH /api/books/{isbn}/subject
=> actualizar los datos sobre el libro cuyo isbn aparece en parámetros.
- DELETE /api/books/{isbn}
=> eliminar el libro cuyo isbn aparece en parámetros.

CopyController:

- POST /api/copies/{isbn}
=> añadir un ejemplar al libro cuyo isbn aparece en parámetros.
- GET /api/copies/{id}
=> obtener los datos del ejemplar cuyo id aparece en parámetros.
- PATCH /api/copies/{id}/updateAvailability?isAvailable={true/false}
=> actualizar los datos sobre el ejemplar cuyo id se pasa en parámetros.
- DELETE /api/copies/{id}/
=> eliminar el ejemplar cuyo id aparece en parámetros.

BorrowingController:

- POST /api/borrowings/{userId}/{copyId}
=> crear el registro de un préstamo donde el usuario que pide el préstamo tiene el id que aparece en el primer parámetro y donde el ejemplar prestado tiene el id que aparece en el segundo parámetro.
- GET /api/borrowings
=> obtener un listado de todos los prestamos activos a la fecha.
- GET /api/borrowings/{lastName}
=> obtener un listado de todos los prestamos activos para el usuario cuyo apellido aparece en parámetros.
- PUT /api/borrowings/{id}
=> actualizar el registro del préstamo cuyo id aparece en parámetros.
- DELETE /api/borrowings/{id}
=> eliminar el registro del préstamo cuyo id aparece en parámetros.

UserController:

- POST /api/users
=> anadir un nuevo usuario a la base de datos.
- GET /api/users
=> obtener la lista completa de todos los usuarios registrados en la base de datos.
- GET /api/users/{id}
=> obtener los datos sobre el usuario cuyo id aparece en parámetros.
- PUT /api/users/{id}
=> actualizar los datos sobre el usuario cuyo id aparece en parámetros.
- DELETE /api/users/{id}
=> eliminar el usuario cuyo id aparece en parámetros.

ReportController

- GET /api/reports
=> obtener la lista completa de todos los registros de prestamos registrados en la base de datos.
- GET /api/reports/{userId}
=> obtener la lista completa de todos los registros de prestamos registrados en la base de datos para el usuario cuyo id aparece en parámetros.
- GET /api/reports/{class}
=> obtener el numero de prestamos registrados en la base de datos para la clase cuyo nombre aparece en parámetros.
- GET /api/reports/{house}
=> obtener el numero de prestamos registrados en la base de datos para la casa cuyo nombre aparece en parámetros.
- GET /api/reports/{bookId}
=> obtener el numero de prestamos registrados en la base de datos para el libro cuyo id aparece en parámetros.
- GET /api/reports/{dateRange}
=> obtener la lista completa de todos los registros de prestamos registrados en la base de datos en el rango de fechas que aparece en parámetros.

✓ 2.4.1.5 Diagrama de flujo de usuarios

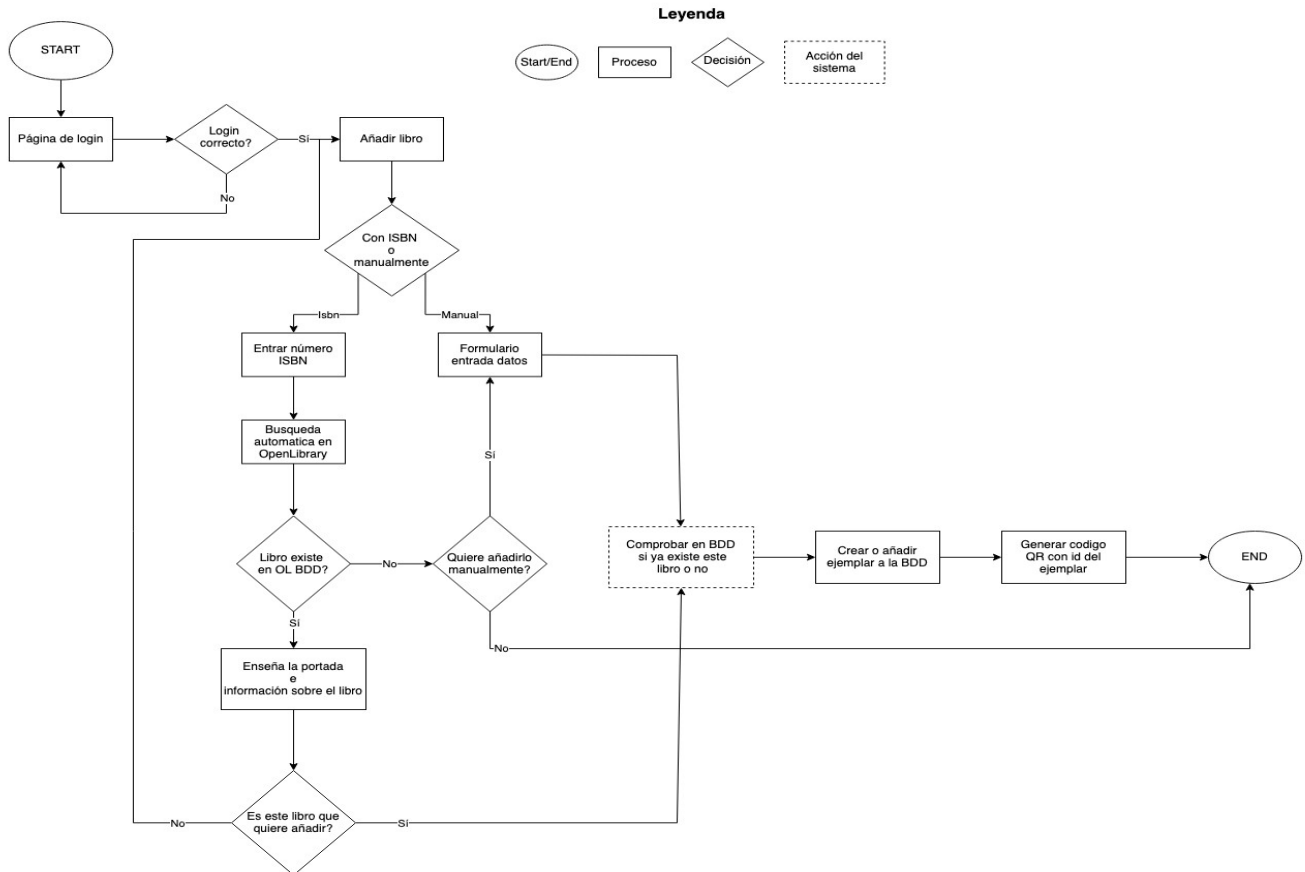
En este apartado se van a presentar dos diagramas de flujo de usuarios para añadir un libro a la biblioteca.

Un libro se puede añadir de dos formas: haciendo una búsqueda con el ISBN del libro en la API de Open Library o bien entrando todos los datos manualmente a través de un formulario.

En ambos casos, el sistema comprobará si ya está registrado o no este libro en la base de datos, creará un ejemplar del libro con un id único y generará un código QR con este id. Este código QR se imprimirá y estará físicamente pegado en el libro. Es este mismo código QR que se escaneará cuando un alumno querrá pedirle préstamo.

Diagrama de flujo de usuario

Flujo para añadir un libro a la biblioteca



- 2.4.2. Detalle de los puntos que se van a controlar para validar el proyecto.
 - Crear una base de datos para almacenar los libros, los ejemplares, los alumnos así que el estado del libro.
 - Crear una API de tipo REST que permita crear, leer, actualizar o eliminar las colecciones Libro y Ejemplar en la base de datos.
 - Llamar a la API de OpenLibrary para obtener información sobre un libro.
 - Modificar la información recibida por la API de OpenLibrary y guardar en la base de datos solo la que se necesita para su funcionamiento.
 - Desarrollar la posibilidad de crear un libro manualmente a través de un formulario y añadirlo a la base de datos.
 - Llamar la API de GoQR.me para generar un código QR único para cada ejemplar que se añade a la base de datos.
 - Desarrollar una funcionalidad que permita leer un código QR para identificar el ejemplar al que pertenece en la base de datos.
 - Tomar en cuenta todos los mensajes de error posibles y guiar al usuario según el mensaje recibido.
 - Diseñar y desarrollar una interfaz de usuario Responsive tomando en cuenta todos los tipos de dispositivo.
 - Crear una función de búsqueda avanzada de libros según ciertos criterios.

3. Ejecución del proyecto y pruebas

3.1. Riesgos de ejecución del proyecto

- 3.1.1. Identificación de riesgos del proyecto.

Identificación de riesgos	Consecuencias o impacto en el proyecto	Frecuencias	Nivel de riesgo / probabilidad que surja	Medida correctiva
Estimación inadecuada del tiempo de desarrollo.	Retrasos en el planning de las entregas. Penalización por parte del cliente.	A lo largo del proyecto. Antes de cada etapa de desarrollo.	Alto	Seguimiento semanal del desarrollo de cada fase.
Alta variación de los requerimientos por parte del cliente.	Retraso en el desarrollo. Entregas fuera de plazo. Confusión e irritación del equipo.	A lo largo del proyecto. Mensualmente.	Medio	Definir previamente todos los requisitos con el cliente. Anticipar cada aspecto del proyecto. Fijar contractual mente los requisitos de la fase en desarrollo.
Integración con sistema externo desconocido	Retraso del proyecto. Necesidad material, de software o humana no anticipada.	Durante la fase de conexión al ERP y durante la fase de despliegue e integración.	Bajo	Conocer las tecnologías empleadas por el cliente. Anticipar la conexión a su ERP.
Fechas de entrega de las distintas fases muy cortas	Retraso del proyecto y de los entregables.	Al principio del proyecto	Medio	Planificar las distintas fases con el cliente y definir fechas realizables para cada entregable.

- 3.1.2. Creación de plan de prevención de riesgos.

Identificación de riesgos	Anticipación del riesgo	Rol y responsabilidad	Acción a desempeñar
Estimación inadecuada del tiempo de desarrollo.	En fase de desarrollo, en cuanto se estima que la entrega no se va a realizar a tiempo.	Project Manager	Avisar a dirección para reevaluar con el cliente los plazos de entrega.
Alta variación de los requerimientos por parte del cliente.	En cuanto una solicitud de cambio por parte del cliente afecte a todo el proyecto y sus plazos de entrega.	Project Manager	Convocar una reunión con el cliente para reevaluar todos los plazos de entrega.
Integración con sistema externo desconocido.	En la fase inicial, fase de planificación del proyecto.	Project Manager	Prever todas las integraciones requeridas para implementar la aplicación.
Fechas de entrega de las distintas fases muy cortas.	Durante la fase de planificación con el cliente.	Project Manager	Convocar una reunión con el cliente para definir plazos de entrega realistas.

3.2. Documentación de ejecución

- 3.2.1. Indicar las necesidades (si las hubiese) en cuanto a permisos para la puesta en marcha del proyecto (por ejemplo si hay que pedir un permiso de obra), o algún tipo de permiso sobre LOPD, etc....
- 3.2.2 Ejecución del proyecto.
 - 3.2.2.1 MongoDB

El primer paso de la implementación ha sido la creación de la base de datos. Para este proyecto he usado [MongoDB Atlas](#) y la consola Compas con la cual he trabajado a diario. No he creado ninguna colección al principio, solo la base de datos vacía para probar conectarla con el backend.

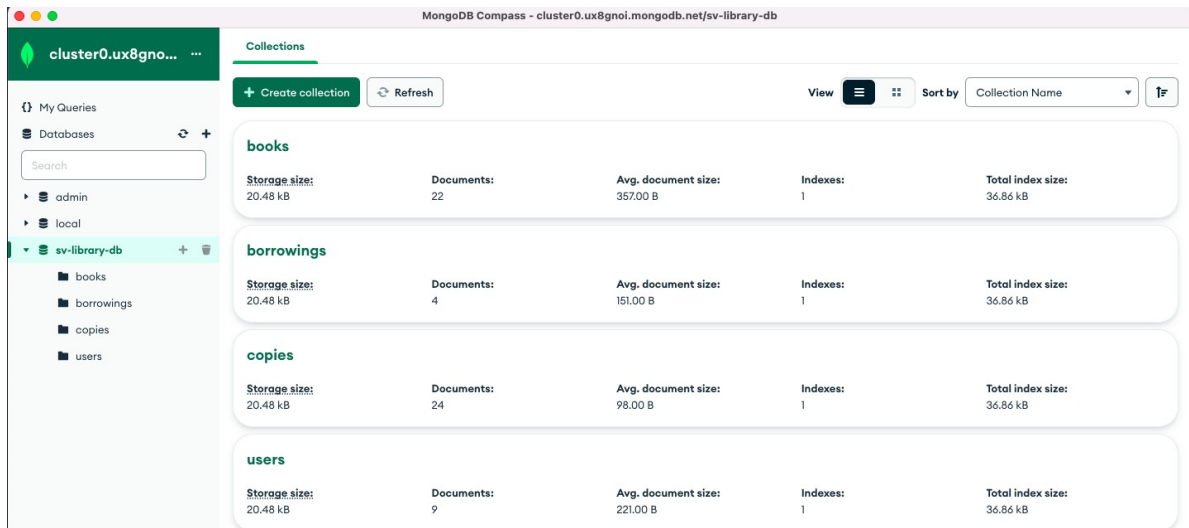


Imagen 1: Captura de pantalla de la consola Compas de MongoDB

○ 3.2.2.2 Spring Initializr

[Spring Initializr](#) es la puerta de entrada para la creación de cualquier proyecto Spring. Facilita la configuración inicial del proyecto, añadiendo automáticamente todas las dependencias necesarias al pom.xml. Para este proyecto, se ha instalado las dependencias iniciales siguientes:

- **Spring Web:** permite crear aplicaciones de tipo RESTful con todas la anotaciones necesarias: `@POST` `@GET` `@PUT` `@DELETE` `@PATCH` y también `@Controller` `@Service` `@Repository`
- **Lombok:** crea automáticamente los getters, setters y constructores a través de anotaciones, dejando el código de los beans más limpios.
- **Spring Data MongoDB:** conector a la base de datos que aporta todos los métodos necesarios para trabajar con MongoDB desde Java
- **Spring Book Dev Tools:** ofrece una funcionalidad de Live Reload para evitar tener que ejecutar la aplicación cada vez que queremos probarla.

○ 3.2.2.3 Conexión de la aplicación a la base de datos

La conexión a la base de datos se debe hacer en el archivo de configuración *application.properties* en la ruta siguiente: *src/main/resources/application.properties*.

Se requiere el nombre de la aplicación y la uri que a su vez contiene el nombre del usuario, la contraseña y el cluster. Todos esos datos se han guardado en un archivo de tipo *.env* para no exponerlos al publico durante los commits y los push al repositorio remoto de GitHub.

También para poder usar los archivos de tipo *.env* en Java Spring, se necesita añadir la dependencia *spring-dotenv* que se puede encontrar en Maven Repository.

Project

Language: ☒ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (RC2) ☐ 3.1.6 (SNAPSHOT) ☒ 3.1.5 ☐ 3.0.13 (SNAPSHOT) ☐ 3.0.12 ☐ 2.7.18 (SNAPSHOT) ☐ 2.7.17

Project Metadata

Group:

Artifact:

Name:

Description:

Package name:

Packaging: ☒ Jar ☐ War

Java: ☐ 21 ☒ 17 ☐ 11 ☐ 8

Dependencies ADD DEPENDENCIES... ⌘ + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Spring Data MongoDB NOSQL
Store data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

GENERATE ⌘ + ↵ EXPLORE CTRL + SPACE SHARE...

Imagen 2: página web de Spring Initializr con las dependencias que se han usado en este proyecto

```

1 spring.data.mongodb.database=${env.MONGO_DATABASE}
2 spring.data.mongodb.uri=mongodb+srv://${env.MONGO_USER}:${env.MONGO_PASSWORD}@${env.MONGO_CLUSTER}
3 spring.data.mongodb.auto-index-creation=true

```

Imagen 3: Archivo application.properties en src/main/resources

```

1 MONGO_DATABASE="sv-library-db"
2 MONGO_USER="Swann"
3 MONGO_PASSWORD="uywbDjETB1P89s4W"
4 MONGO_CLUSTER="cluster0.ux8gnoi.mongodb.net"

```

Imagen 4: Datos de conexión a la base de datos

```

44 <dependency>
45 <groupId>me.paulschwarz</groupId>
46 <artifactId>spring-dotenv</artifactId>
47 <version>2.3.0</version>

```

Imagen 5: Dependencia necesaria para leer los archivos .env

- 3.2.2.3 Estructura de los directorios del backend

La parte del backend se ha desarrollado con Java y el framework Spring y el gestor de proyecto Maven. La estructura de los directorios sigue la estructura clásica de cualquier proyecto Maven donde encontramos el directorio principal `/src` que contiene a su vez los directorios `/src/main` y `/src/test`.

A la raíz del proyecto tenemos el archivo de configuración del proyecto y de gestión de las dependencias `pom.xml`.

Los recursos necesitados por la aplicación se encuentran en el directorio `/src/main/resources`. El cual contiene el archivo de conexión a la base de datos `application.properties`.

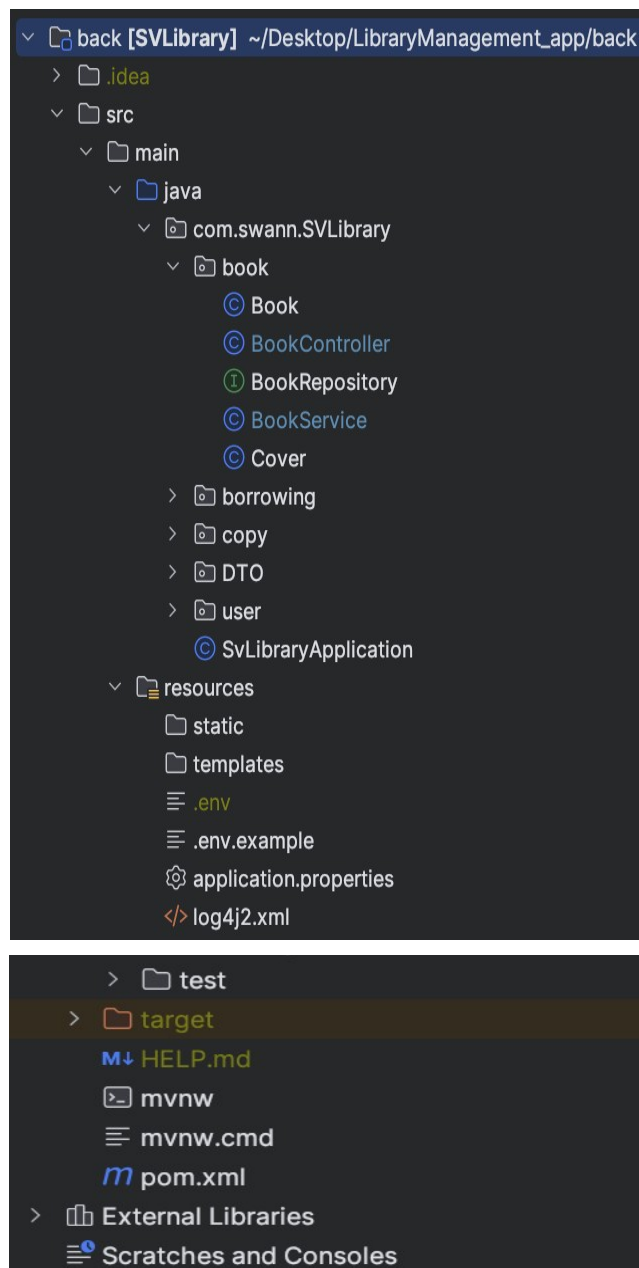


Imagen 6: Estructura de los directorios de la parte backend

Como se puede ver en la captura de pantalla anterior, las distintas clases están organizadas en:

- 4 paquetes que representan las 4 colecciones que tenemos en la base de datos (book, user, borrowing, copy)
- 1 paquete llamado DTO que contiene los distintos objetos que necesita la aplicación. Los DTO son unos contenedores que agrupan datos de distintas colecciones antes de devolverlos al usuario si lo requiere.

Para cada uno de los 4 paquetes que representan los datos tenemos:

- 1 clase **controlador** encargada de recibir las peticiones http.
- 1 clase **servicio** encargada de la lógica de negocio.
- 1 clase **repositorio** encargada de comunicar con la base de datos.
- 1 clase de tipo POJO (Plain Old Java Object) que son nada más que unas clases simples de representación de los datos.

Toda la parte del backend está basada en esta estructura de directorio y esta lógica de separación en capas: controlador, servicio, repositorio.

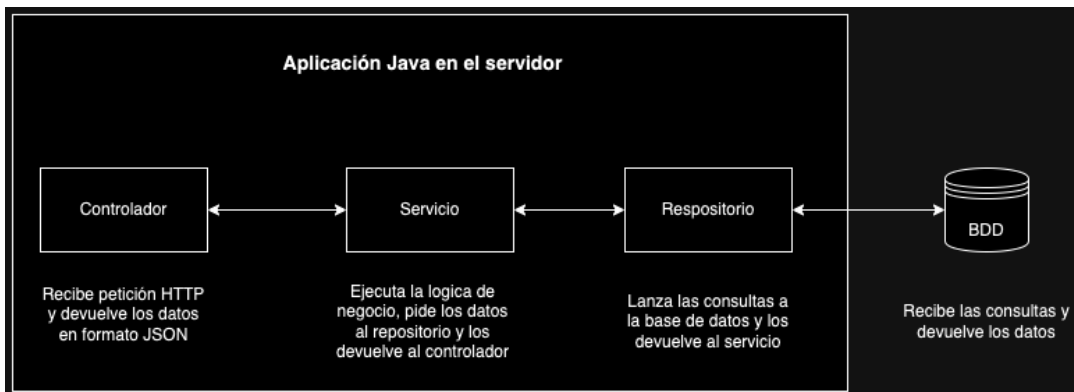


Imagen 7: Funcionamiento en capas de la parte backend

○ 3.2.2.3 Creación de las rutas de la API con Spring

Spring facilita mucho la creación de una aplicación de tipo RESTful gracias a su sistema de anotaciones para cada método HTTP: GET, POST, PUT, DELETE que se traducen en Spring como `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`.

Las rutas se especifican directamente después del método HTTP como se puede ver en la captura de pantalla siguiente.

```

    Swann Julien *
    @GetMapping("/book/get-by-id/{id}")
    public Optional<Book> getBookByIdString(@PathVariable String id){
        Optional<Book> book = bookService.findBookByIdString(id);
        if (book.isEmpty())
            throw new RuntimeException("No ID has been provided. Please send a valid ID");
        return book;
    }

    Swann Julien *
    @GetMapping("/books/dto/{id}")
    public Optional<BookDTO> getBookById(@PathVariable String id){
        Optional<BookDTO> book = bookService.findBookByCopyId(id);
        if (book.isEmpty())
            throw new RuntimeException("No ID has been provided. Please send a valid ID");
        return book;
    }

    Swann Julien
    @PostMapping("books/{isbn}/addSubject")
    public String addSubjectToExistingBook(
        @PathVariable String isbn,
        @RequestParam("subjectType") String subjectType,
        @RequestParam("subject") String subject) {
        return bookService.addSubject(isbn, subjectType, subject);
    }

```

Imagen 8: Vista de la clase BookControler con sus rutas y métodos HTTP

○ 3.2.2.4 Pruebas de las rutas de la API con Postman

Todas las rutas creadas en Spring se han probado con Postman. Postman es una herramienta que realiza peticiones HTTP y permite testear las rutas de una API. Como se puede ver en la captura de pantalla abajo, he creado una prueba para cada ruta definida en cada controlador de la aplicación.

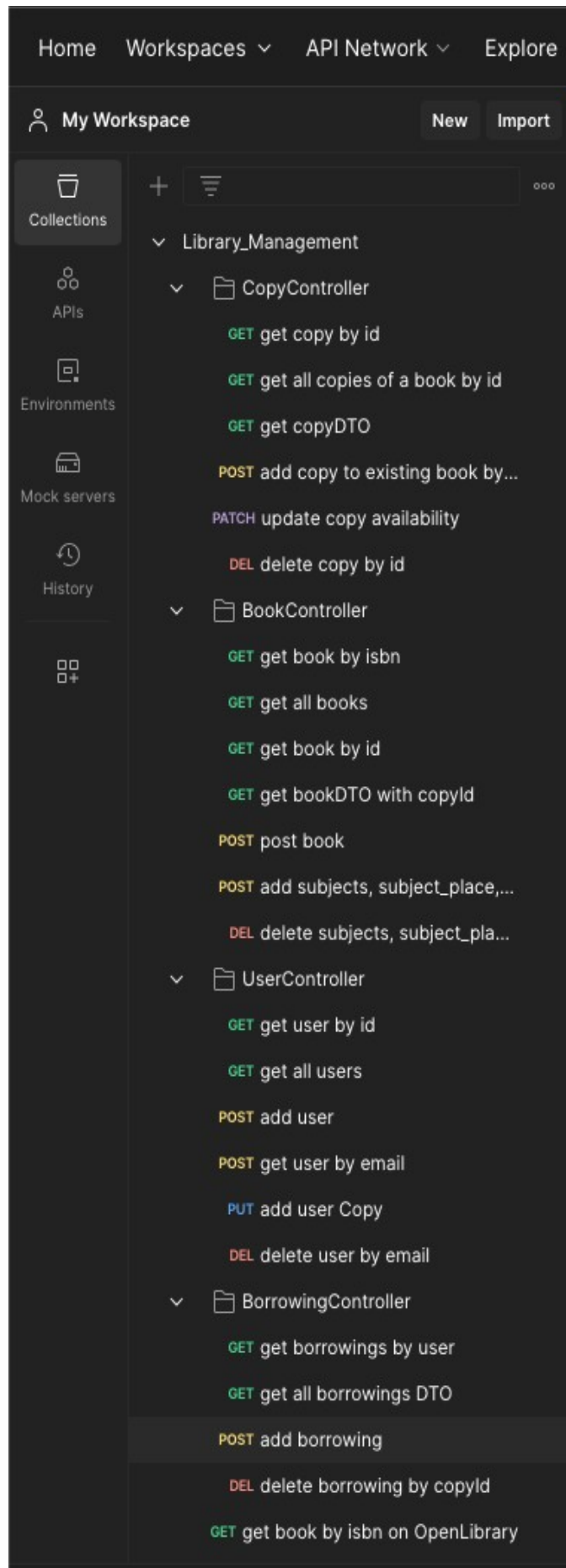


Imagen 9: Pruebas de la rutas de la API con Postman

Como ejemplo vamos a probar la ruta que permite crear un nuevo libro. Al crear un libro nuevo, el servidor nos tiene que devolver el ID de la copia del libro que se acaba de crear. Mandamos un objeto en formato JSON que contiene toda la información sobre un libro ficticio. Esta petición nos devuelve en el body el ID de la copia que se ha creado, entonces ha pasado correctamente el test.

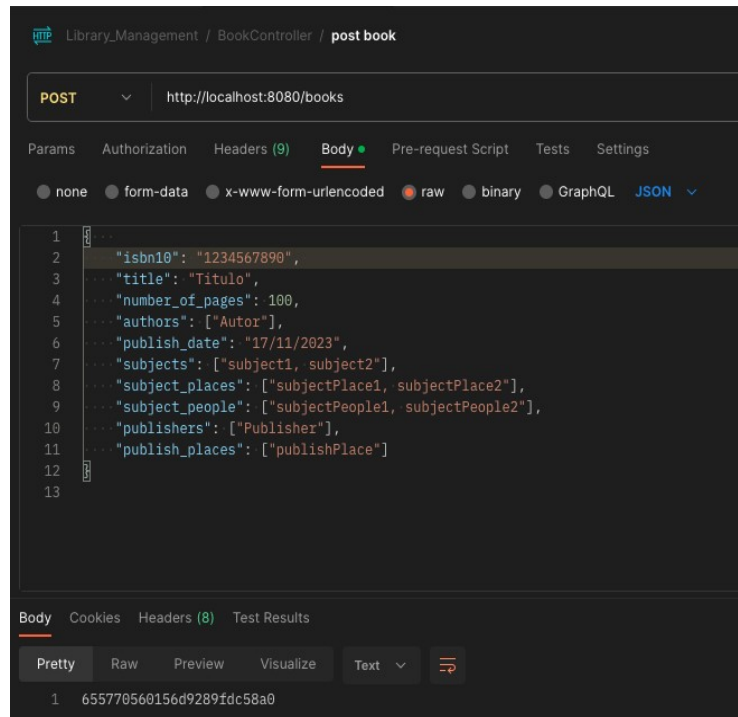


Imagen 10: Petición POST de creación de libro en Postman

Luego comprobamos en la base de datos que el libro se ha creado correctamente.

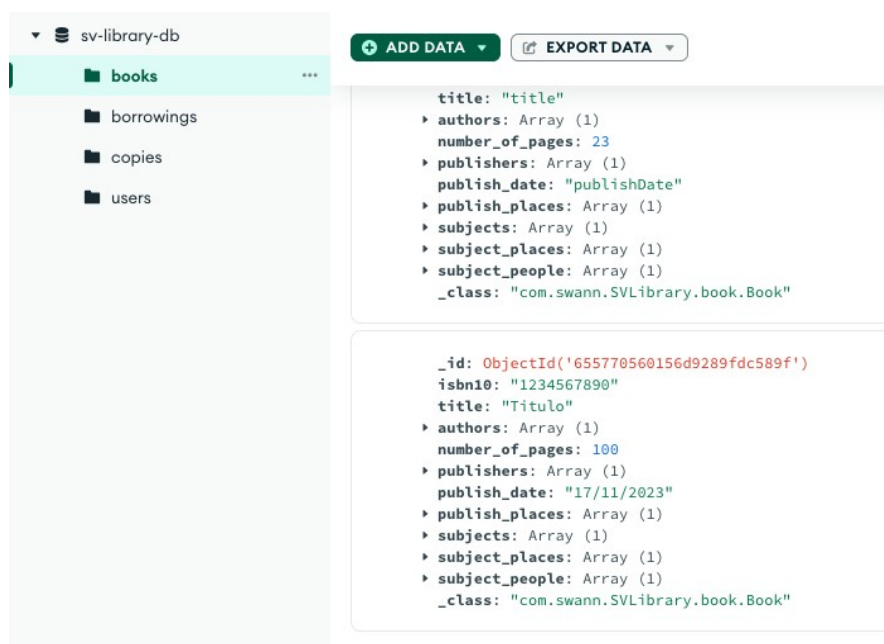


Imagen 11: Comprobación en la base de datos de la creación del libro

Si volvemos a mandar la misma petición, el servidor nos devuelve un mensaje de error porque el libro ya existe en la base de datos y hemos definido en la lógica de negocio que no se puede añadir dos o más libros con el mismo ISBN.

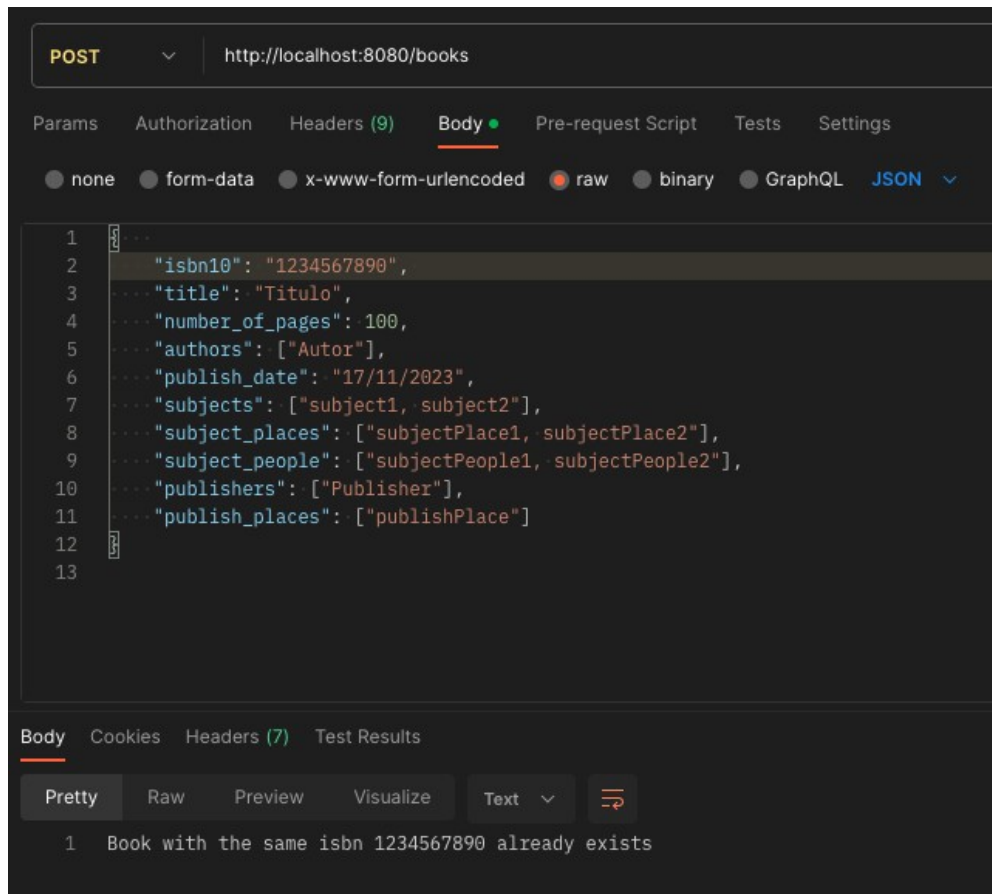


Imagen 12: Mensaje de error: no se puede crear dos libros con mismo ISBN

De esta forma, se ha probado cada ruta para ver si la API responde correctamente en todos los casos posibles.

Se puede probar la API en el enlace siguiente: <https://elements.getpostman.com/redirect?entityId=26488419-37c0429a-96c1-47a5-bfe6-b508e5e606dd&entityType=collection>

◦ 3.2.2.5 Creación de la interfaz gráfica con Next.js

Next.js es un framework de React que facilita la creación de la interfaz grafica y de las distintas rutas entre cada página. Permite también la ejecución de componentes directamente en el servidor.

Una de las ventajas de Next.js es el sistema de routing basado en directorios. Cada directorio creado dentro del directorio `/app` se convierte en una ruta. Así por ejemplo, al crear un directorio `/app/books` estamos creando una ruta que se convertirá en `http://nombre-de-dominio-de-la-aplicación/books`.

Se puede ver en la captura de pantalla abajo la estructura de los directorios de la parte frontend de la aplicación.

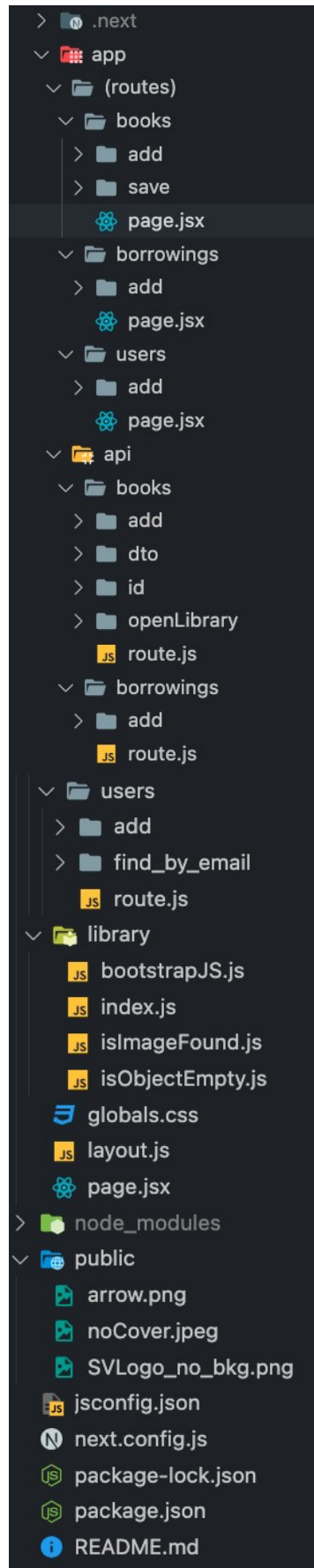


Imagen 13: Directorios aplicación Next.js

Explicación de los distintos directorios:

- **app**: contiene las distintas páginas que constituyen la aplicación. Cada directorio representa una ruta. Se puede anidar directorio lo que se traduce por un nuevo segmento de la ruta. Por ejemplo `app/books/add` se traduce por `http://nombre-de-dominio/books/add`. Cada directorio debe tener un archivo `page.jsx` que representa el componente principal de dicha ruta.
- **api**: contiene las llamadas HTTP a nuestra API. Esas llamadas se hacen directamente en el servidor.
- **library**: contiene scripts javascript de utilidad para la programación de la aplicación como funciones que se repiten en distintas páginas.
- **public**: contiene las imágenes que se usan en la aplicación

◦ 3.2.2.6 Detalle de una petición HTTP a la API desde el frontend

La mayoría de las peticiones HTTP en Next.js siguen el mismo camino: desde el navegador del cliente se hace una llamada asíncrona de tipo *fetch* al servidor. El servidor a su vez llama a nuestra API. Esta forma de proceder es la recomendada por la propia [documentación de Next.js](#).

En las capturas de pantalla siguientes, podemos ver por ejemplo que para añadir un libro a la base de datos, enviamos los datos desde el navegador del cliente a la ruta <http://localhost:3000/api/books/add>.

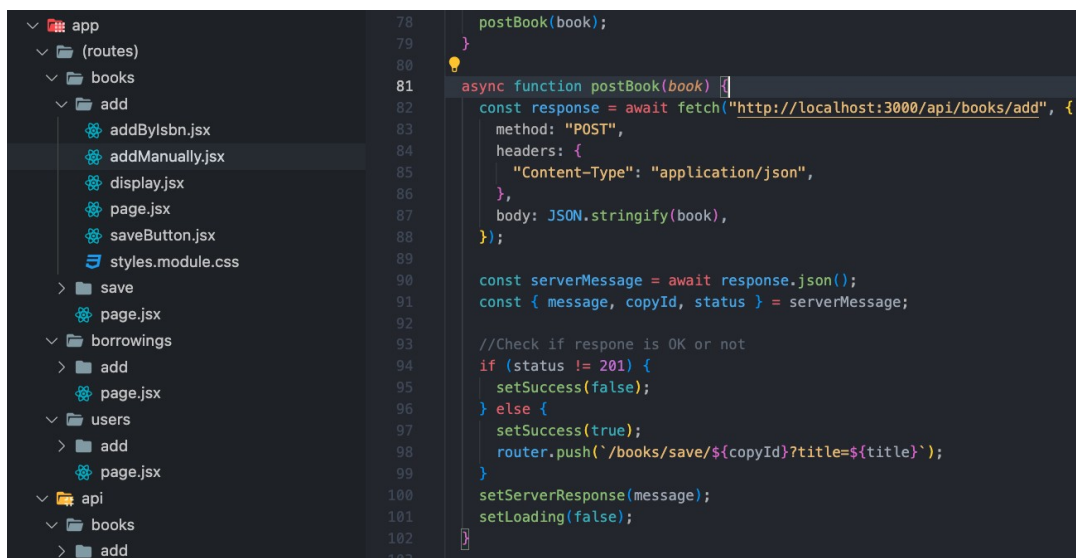
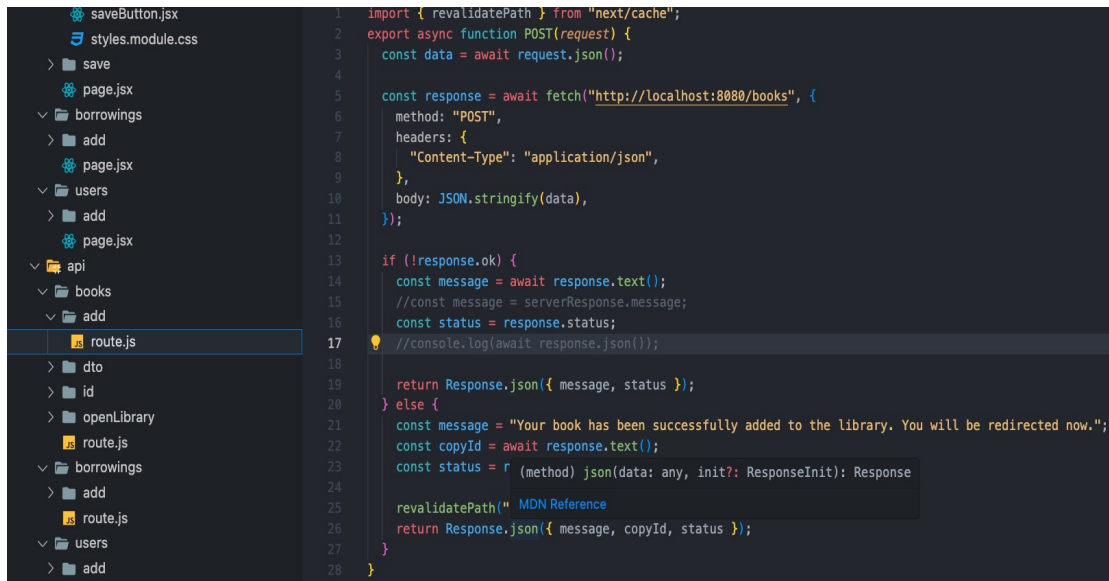


Imagen 14: Llamada a la ruta interna `api/books/add`

Luego desde esta ruta recogemos los datos en el body de petición y lanzamos una nueva petición, esta vez a nuestra API, pasando en el body de la petición los datos recogidos anteriormente.



```
1 import { revalidatePath } from "next/cache";
2 export async function POST(request) {
3   const data = await request.json();
4
5   const response = await fetch("http://localhost:8080/books", {
6     method: "POST",
7     headers: {
8       "Content-Type": "application/json",
9     },
10    body: JSON.stringify(data),
11  });
12
13  if (!response.ok) {
14    const message = await response.text();
15    //const message = serverResponse.message;
16    const status = response.status;
17    //console.log(await response.json());
18
19    return Response.json({ message, status });
20  } else {
21    const message = "Your book has been successfully added to the library. You will be redirected now.";
22    const copyId = await response.text();
23    const status = r (method) json(data: any, init?: ResponseInit): Response
24
25    revalidatePath(" MDN Reference
26    return Response.json({ message, copyId, status });
27  }
28 }
```

Imagen 15: Llamada a la API

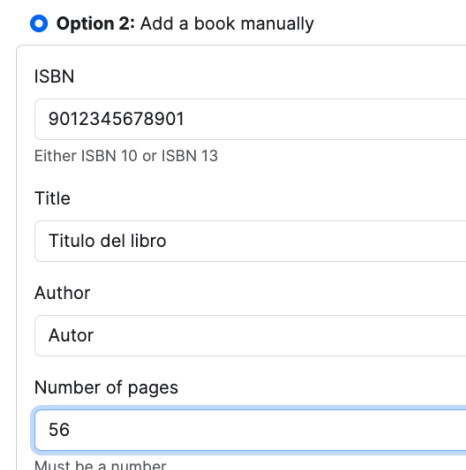
◦ 3.2.2.7 Llamada a la API Goqr.me

Si seguimos con el ejemplo anterior, después de que el usuario haya rellenado un formulario con los datos del libro a añadir a la biblioteca, si la respuesta por parte de la API es positiva, es inmediatamente redirigido a una página en la que aparece el código QR que contiene el ID de la copia.

Este mecanismo se puede ver en la captura de pantalla anterior (*Llamada a la ruta api/books/add*). Si la respuesta del servidor al añadir el libro tiene un *status code* 201 entonces el usuario está redirigido gracias al hook *useRouter* y su método *push()*.

Una vez en la nueva página */books/save*, la aplicación lee el parámetro *copyId* y el parámetro de búsqueda *title* en la URL y lanza una llamada a la API de generación de código QR [Goqr.me](https://goqr.me).

El usuario puede ahora imprimir el código QR que identifica de forma única a la copia del libro y pegarlo en él.



Option 2: Add a book manually

ISBN
9012345678901
Either ISBN 10 or ISBN 13

Title
Titulo del libro

Author
Autor

Number of pages
56
Must be a number

Imagen 16: Paso 1: Se rellenan los datos en un formulario



Imagen 17: Paso 2: usuario recibe un código QR de identificación del libro

◦ 3.2.2.8 Validación de formularios

Esta aplicación interactúa mucho con el usuario a través de formularios. Que sea para añadir un libro, un préstamo o un usuario es necesario siempre hacer una doble validación, por parte del frontend como por parte del backend.

La validación en el frontend se ha hecho gracias a un hook de React llamado react-hook-form. Básicamente react-hook coge etiquetas html que ya existen para validar un formulario como *required*, *min-length*, *max-length*, *pattern* pero añade la posibilidad de añadir un mensaje de error fácilmente.

Por lo que son de los estilos, me he basado en la clase *is-invalid* de Bootstrap que pone en rojo el borde del input y añade un símbolo de peligro.

En la captura de pantalla siguiente podemos ver la implementación de la validación de formulario para la página *addBorrowing*.

```
<form onSubmit={handleSubmit(onSubmit)} ref={formRef}>
  <div className="mb-3">
    <label htmlFor="email" className="form-label">
      Email
    </label>
    <input
      type="email"
      name="email"
      className={`form-control ${errors.email ? "is-invalid" : ""}`}
      id="email"
      {...register("email", {
        required: "You must provide an email",
        pattern: { value: /^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$/, message: "Invalid email address" },
      })}
    />
    <p className="text-danger">{errors.email && errors.email.message}</p>
    <p className={`text-danger mt-3`} >{userMessage}</p>
  </div>
</form>
```

Imagen 18: Implementación de la validación del formulario Add Borrowing

```
function onSubmit(data) {
  setLoading(true);

  const email = data.email;
  const copyId = data.copyId;
  const emailObj = {
    email: email,
  };
  getUser(emailObj);
  getBookDTO(copyId);

  // Reset form's input
  formRef.current.reset();
}
```

Si el usuario no especifica un email, le aparece un mensaje de error y si proporciona un email que no corresponda con el Regexp de validación, le aparece otro mensaje de error.

Add a new borrowing

Please provide an user email and the copy id that the user will borrow.

Email

You must provide an email

Copy id

Search

Imagen 19: Validación del email: no email proporcionado

Add a new borrowing

Please provide an user email and the copy id that the user will borrow.

Email

Invalid email address

Copy id

Search

Imagen 20: Validación del email: email incorrecto

En el backend, podemos ver un ejemplo de validación de los datos de entrada en el método `addBookAndCopyToLibrary()` en el cual se comprueba que el ISBN tenga 10 o 13 caracteres.

```

... public ObjectId addBookAndCopyToLibrary(Book book){
...     // Check if isbn 10 is 10 characters and isbn is 13 characters
...     if (book.getIsbn10() != null && book.getIsbn10().length() != 10){
...         throw new RuntimeException("ISBN 10 must be 10 characters long.");
...     } else if (book.getIsbn13() != null && book.getIsbn13().length() != 13) {
...         throw new RuntimeException("ISBN 13 must be 13 characters long.");
...     } else {
...         String isbn = book.getIsbn10() == null ? book.getIsbn13() : book.getIsbn10();
...         Optional<Book> optionalBook = findBookByIsbn(isbn);
...         // Avoid saving two books with the same isbn. Make isbn unique
...         if (optionalBook.isEmpty()){
...             Book savedBook = bookRepository.save(book);
...             Copy newCopy = new Copy(savedBook.getId(), isAvailable: true);
...             Copy savedCopy = copyRepository.save(newCopy);
...             return savedCopy.getId();
...         } else {
...             throw new RuntimeException("Book with the same isbn " + isbn + " already exists");
...         }
...     }
... }

```

Imagen 21: Validación del ISBN en la parte backend

◦ 3.2.2.9 Mensajes de error

En cada interacción con la base de datos, el usuario está informado del éxito o no de la operación y recibe un mensaje positivo o negativo personalizado según el caso.

Por ejemplo cuando se añade un nuevo usuario, si todos los campos han sido completado correctamente, se manda el formulario a la API y nos devuelve un mensaje de confirmación de creación de usuario: *New user successfully created.*

House

Picasso

Email

swann.julien@sunnyview.com

Phone number

622333444

Select a role for this user

BE CAREFUL

Admin have all rights on the database: create, update, delete either books, users and borrowings. Student on the other hand can only see books availability, that's it.

☒ Student

☐ Admin

Add user

New user successfully created.

Imagen 22: Mensaje de confirmación de creación de usuario

Pero si intentamos volver a enviar el formulario con el mismo email, la API nos devuelve un mensaje de error: *User with email: [email address] already exists and can't be added to the*

database.

House

Picasso

Email

swann.julien@sunnyview.com

Phone number

622333444

Select a role for this user

BE CAREFUL

Admin have all rights on the database: create, update, delete either books, users and borrowings.

Student on the other hand can only see books availability, that's it.

☒ Student

☐ Admin

Add user

User with email: swann.julien@sunnyview.com already exists and can't be added to the database

Imagen 23: Mensaje de error de creación de usuario

Los mensajes de error provienen de la API y se devuelven al usuario a través del body de la respuesta de la petición HTTP. Abajo podemos ver primero el mensaje de error en la API y luego como se usa este mensaje en el frontend para enseñárselo al usuario.

```
Swann Julien
public void addUser(User user) {
    Optional<User> OptionalUser = findUserByEmail(user.getEmail());
    if (OptionalUser.isPresent()){
        throw new IllegalArgumentException("User with email: " + user.getEmail() + " already exists and can't be added to the database");
    } else {
        userRepository.save(user);
    }
}
```

Imagen 24: Mensaje de error proveniente de la lógica de negocio de la API

```

app > api > users > add > JS route.js > ...
1  import { revalidatePath } from "next/cache";
2  export async function POST(request) {
3    const data = await request.json();
4
5    const response = await fetch("http://localhost:8080/users", {
6      method: "POST",
7      headers: {
8        "Content-Type": "application/json",
9      },
10     body: JSON.stringify(data),
11   });
12
13   if (!response.ok) {
14     const message = await response.text();
15     const status = response.status;
16
17     return Response.json({ message, status });
18   } else {
19     const message = "New user successfully created.";
20     const status = response.status;
21
22     revalidatePath("/users");
23     return Response.json({ message, status });
24   }
25 }

```

Imagen 25: Se captura el mensaje de error en el frontend si la respuesta no es OK

◦ 3.2.2.10 Creación de un spinner de espera cuando se cargan los datos

Un problema que tenía la aplicación es que cuando el usuario le daba a uno de los menús que llama a datos en la base de datos, había siempre un tiempo de latencia durante el cual el usuario no tenía ningún retorno. No se sabía muy bien si es que la página solicitada tenía un problema o si no se había dado bien al botón.

Para remediar a este problema, se ha creado un spinner que aparece automáticamente cuando se carga cualquier página de la aplicación. Desde Next.js 14 esta funcionalidad es muy fácil de realizar.

Dentro del archivo `layout.jsx` que está a la raíz del directorio `/app` añadimos un componente llamado `<Suspense>` que engloba todas las páginas de la aplicación. Este componente, como bien lo define la [documentación de React](#), permite cargar una función callback mientras los componentes hijos a este `<Suspense>` se cargan.

Sabiendo que el archivo `layout.jsx` que está a la raíz de `/app` es el archivo padre a todos los demás componentes de la aplicación, al englobar todos los componentes hijos con un `<Suspense>`, nuestra función callback se llamará cada vez que cualquier página de la aplicación se cargue.

Se puede ver abajo el componente `<Suspense>` que engloba todos los componentes hijos de la aplicación y que llama a una función `loading.js`. Esta función no tiene nada más que dos `<div>` para crear el estilo tanto del fondo como del spinner en sí.

```

    </li>
    <li>
      <Link href="/borrowings/add" className="dropdown-item">
        Add borrowing
      </Link>
    </li>
  </ul>
</div>
</ul>
<form className="d-flex" role="search">
  <input className="form-control me-2" type="search" placeholder="Search" aria-label="Search" />
  <button className="btn btn-outline-success" type="submit">
    Search
  </button>
</form>
</div>
</div>
</nav>
<Suspense fallback={<Loading />}>{children}</Suspense>
</body>
</html>
);
}

```

Imagen 26: Creación de un Suspense que engloba toda la aplicación

```

app > library > JS loading.js > ...
1   import styles from "../styles.module.css";
2   export default function Loading() {
3     return (
4       <div className={styles.wrapper}>
5         <div className={styles.loader}></div>
6       </div>
7     );
8   }

```

```

app > CSS styles.module.css > @keyframes spin
1   /* Thanks to David Omatayo: https://blog.logrocket.com/css-spinners/
2
3   .wrapper {
4     width: 100%;
5     height: 100vh;
6     position: absolute;
7     top: 0;
8     left: 0;
9     background-color: rgb(0, 0, 0, 0.3);
10    backdrop-filter: blur(10px);
11    display: flex;
12    justify-content: center;
13    align-items: center;
14    z-index: 99;
15  }
16
17  .loader {
18    border: 10px solid #f3f3f3;
19    border-radius: 50%;
20    border-top: 10px solid #505050;
21    width: 60px;
22    height: 60px;
23    animation: spin 2s linear infinite;
24  }
25
26  @keyframes spin {
27    0% {
28      transform: rotate(0deg);
29    }
30    100% {
31      transform: rotate(360deg);
32    }
33  }

```

Imagen 27: Estilo del spinner

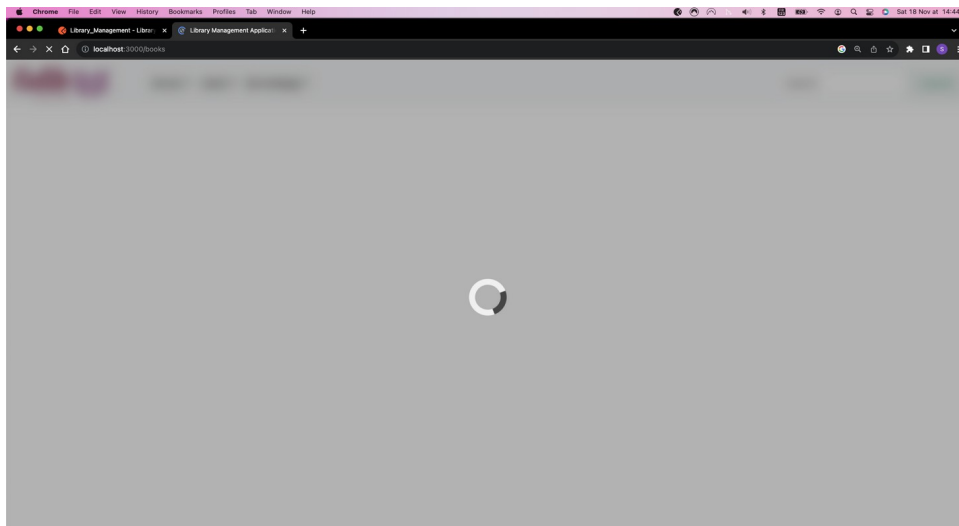


Imagen 28: Vista del spinner cuando se carga alguna pagina de la aplicación

- 3.2.3. Manuales finales
 - Manual de usuario

1. Descripción general de la aplicación

EzLib es una aplicación de gestión de libros y prestamos online. Permite no solo crear una biblioteca sino también gestionar los prestamos desde una plataforma única. Es una aplicación web y se puede consultar desde cualquier ordenador que tenga una conexión a Internet.

EzLib está conectado con la biblioteca open source colaborativa Open Library lo que le permite buscar toda la información sobre un libro con tan solo especificar un ISBN.

EzLib facilita la gestión de los prestamos al generar un código QR único para cada libro. Esta funcionalidad ofrece la posibilidad de registrar un préstamo o una devolución solo teniendo un móvil y una conexión a internet. No se necesita lector RFID o lector de código de barra. Así se puede realizar cualquier tarea de gestión en cualquier lugar de la biblioteca sin tener que invertir en aparatos costosos.

2. Página de inicio

Desde la página de inicio tenemos acceso a todas las funcionalidades en el menú. El menú es muy intuitivo y agrupa todas las funcionalidades en tres opciones:

- **Books:** para consultar y añadir libros
- **Users:** para consultar y añadir usuarios
- **Borrowings:** para consultar y crear un préstamo

Si le damos al logo de EzLib desde cualquier página de la aplicación, volvemos a la página de inicio.

Nota: La barra de búsqueda no es operativa todavía. Es una funcionalidad que se va a añadir en las próximas semanas.

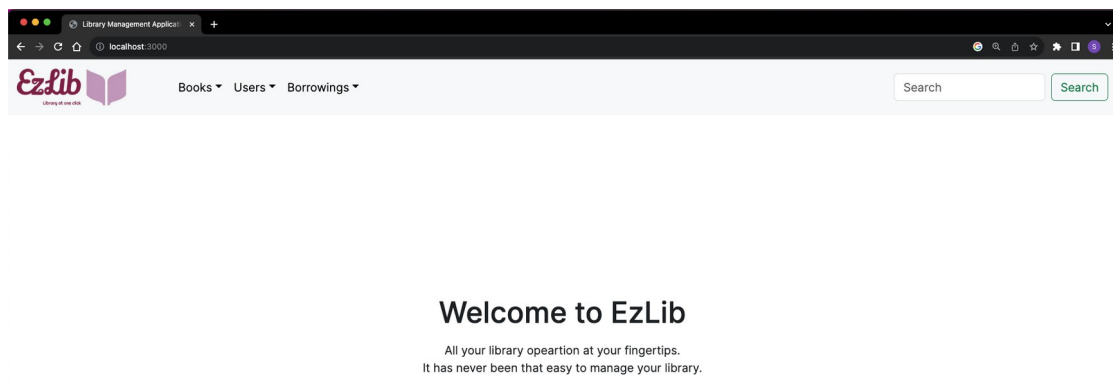
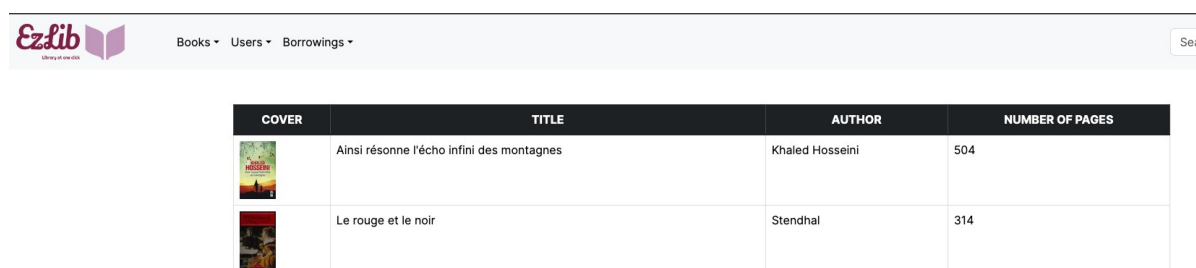


Imagen 29: Pagina de inicio de EzLib

3. Consultar los libros de la biblioteca

Se pueden consultar todos los libros de la biblioteca desde el menú *Books > See all books*. Desde este menú aparecerá todos los libros con su información presentada en una tabla.





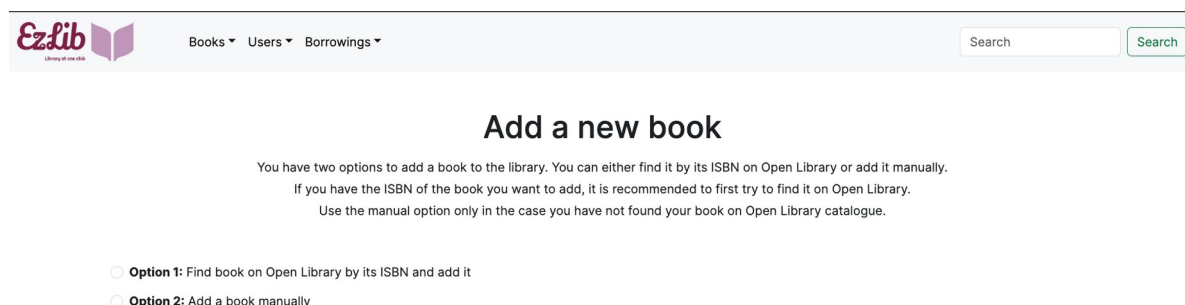
COVER	TITLE	AUTHOR	NUMBER OF PAGES
	Ainsi résonne l'écho infini des montagnes	Khaled Hosseini	504
	Le rouge et le noir	Stendhal	314

Imagen 30: Vista del menú *See all books*

4. Añadir un libro a la biblioteca

Desde el menú *Books > Add book* tenemos la posibilidad de añadir libros a la biblioteca. EzLib ofrece dos opciones para añadir un libro: haciendo una búsqueda en Open Library con el ISBN del libro o bien añadiendo todos los datos del libro manualmente. Vamos a ver cada una de esas opciones.



Add a new book

You have two options to add a book to the library. You can either find it by its ISBN on Open Library or add it manually.
If you have the ISBN of the book you want to add, it is recommended to first try to find it on Open Library.
Use the manual option only in the case you have not found your book on Open Library catalogue.

☐ **Option 1:** Find book on Open Library by its ISBN and add it

☐ **Option 2:** Add a book manually

Imagen 31: Vista del menú *Add book*

4.1 Buscar un libro en Open Library antes de añadirlo a la biblioteca


Esta opción es la que se recomienda usar por defecto. Se requiere proporcionar un ISBN y EzLib busca automáticamente en la base de datos de Open Library el libro correspondiente.

Si EzLib encuentra un libro, aparecerá en pantalla seguido de un botón preguntando al usuario si este es el libro que quiere añadir.

Al darle al botón *Add book to library*, el libro se añade automáticamente a la biblioteca con todos sus datos y la aplicación nos redirige hacia una página en la cual aparece el código QR conteniendo el ID de la copia del libro que se acaba de crear.

Este código QR se debe imprimir desde el menú del navegador y pegar en el libro.

☐ **Option 1:** Find book on Open Library by its ISBN and add it



Veinte poemas de amor y una canción desesperada
Isbn: 8492421215
Author: Pablo Neruda
Publish date: Jun 01, 2008
Publisher: Verticales

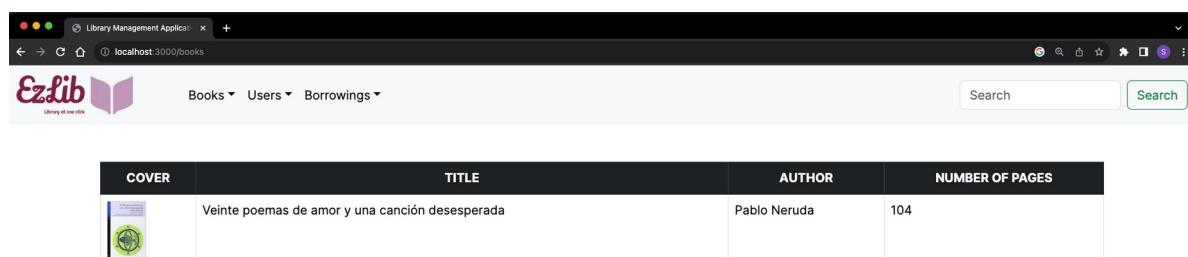
Do you want to add this book to the library?

Imagen 32: Tras entrar un numero ISBN aparece toda la información sobre el libro



Imagen 33: Código QR identificando la copia del libro creada

Se podría comprobar que el libro ha sido añadido correctamente yendo al menú *See all books* seguido de un reload de la página (F5 o ctrl + r on Windows o cmd + r on Mac).




COVER	TITLE	AUTHOR	NUMBER OF PAGES
	Veinte poemas de amor y una canción desesperada	Pablo Neruda	104

Imagen 34: El libro se ha creado correctamente

4.2 Añadir un libro manualmente

La segunda opción para añadir un libro es entrando los datos manualmente. Al darle al botón *Option 2: Add a book manually*, un formulario se despliega. Para poder enviar este formulario es obligatorio rellenar los datos siguientes:

- **ISBN:** debe tener 10 dígitos mínimo y 13 dígitos máximo. En caso contrario, saltará un mensaje de error.
- **Título:** se debe rellenar este campo
- **Autor:** se debe rellenar este campo

No es obligatorio rellenar el número de páginas pero si se rellena, debe tener solo números.

Add a new book

You have two options to add a book to the library. You can either find it by its ISBN on Open Library or add it manually.
If you have the ISBN of the book you want to add, it is recommended to first try to find it on Open Library.
Use the manual option only in the case you have not found your book on Open Library catalogue.

☐ Option 1: Find book on Open Library by its ISBN and add it

☒ Option 2: Add a book manually

ISBN

Either ISBN 10 or ISBN 13

Title

Author

Number of pages

Must be a number

Publisher

Publish date

Publish place

Subjects

Subject places

Subject people

Add book to library

Imagen 35: Formulario de creación de libro de forma manual

Una vez todos los campos rellenados y validos, se puede enviar el formulario. EzLib nos redirige entonces hacia una página en la cual aparece el código QR de la copia del libro que se acaba de crear.



Título de prueba

6558d5170156d9289fdc58c2

Go back to library

Podemos comprobar que el libro se ha creado correctamente a la biblioteca yendo al menú *See all books*.

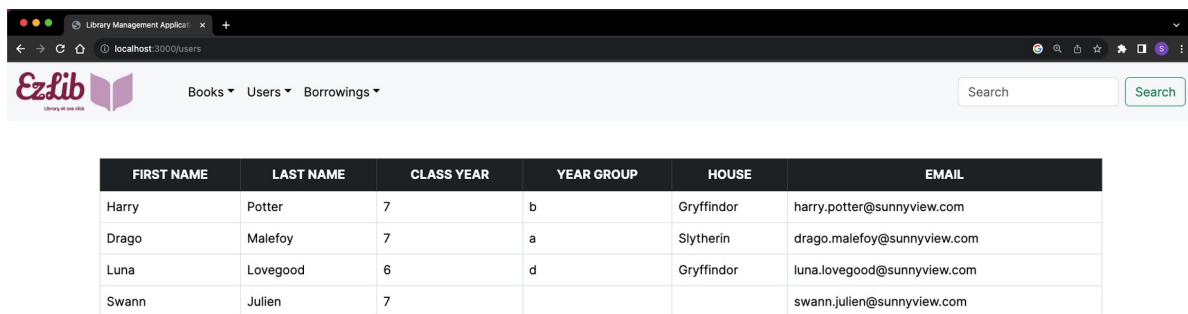


COVER	TITLE	AUTHOR	NUMBER OF PAGES
	Veinte poemas de amor y una canción desesperada	Pablo Neruda	104
	Titulo de prueba	Autor de prueba	0

Imagen 36: El libro se ha creado correctamente

5. Consultar los usuarios

Desde el menú *Users > See all users* se pueden ver todos los usuarios registrados en la base de datos. Hay que precisar que solo los usuarios registrado podrán pedir préstamo.



FIRST NAME	LAST NAME	CLASS YEAR	YEAR GROUP	HOUSE	EMAIL
Harry	Potter	7	b	Gryffindor	harry.potter@sunnyview.com
Drago	Malefoy	7	a	Slytherin	drago.malefoy@sunnyview.com
Luna	Lovegood	6	d	Gryffindor	luna.lovegood@sunnyview.com
Swann	Julien	7			swann.julien@sunnyview.com

Imagen 37: Vista del menú *See all users*

6. Añadir un usuario

Desde el menú *Users > Add users* tenemos la posibilidad de añadir usuarios. Se recuerda que solo los usuarios registrados pueden hacer préstamos.

Esta página presenta un formulario con algunos campos que son obligatorios de rellenar.

Add a new user

Please fill up all fields of the form to add a new user.

First name

Last name

Class year

Year group

House

Email

Phone number

Select a role for this user

BE CAREFUL
Admin have all rights on the database: create, update, delete either books, users and borrowings.
Student on the other hand can only see books availability, that's it.

☐ Student
☐ Admin

Imagen 38: Formulario para crear un nuevo usuario

Los campos obligatorios de rellenar son:

- **Nombre**
- **Apellido**
- **Email:** debe cumplir con un formato de email correcto: [xxx@xxx.xx](#)
- **Tipo de usuario**

La clase no es un campo obligatorio de rellenar pero si se rellena debe ser un número.

Una vez todos los campos completados y validos, si el email del usuario no está ya registrado en la base de datos, aparecerá un mensaje positivo confirmando la creación del usuario.

Select a role for this user

BE CAREFUL

Admin have all rights on the database: create, update, delete either books, users and borrowings.

Student on the other hand can only see books availability, that's it.

- ☐ Student
☐ Admin

New user successfully created.

Imagen 39: Validación de creación de usuario nuevo

En caso contrario, si el email del usuario ya está registrado, aparecerá un mensaje de error.

Select a role for this user

BE CAREFUL

Admin have all rights on the database: create, update, delete either books, users and borrowings.

Student on the other hand can only see books availability, that's it.

☐ Student

☐ Admin

Add user


User with email: prueba@prueba.com already exists and can't be added to the database

Imagen 40: Mensaje de error al crear un usuario nuevo

7. Consultar los prestamos

Se pueden ver todos los prestamos activos en el menú *Borrowings* > *See all borrowings*.

En esta página se enseña todos los prestamos activos en una tabla.



TITLE	COPY ID	FIRST NAME	LAST NAME	EMAIL	BORROW DATE	DUE DATE
Obres completes de Tísner.	655346b28f6f9c46959a327d	Olga	Olmedo Nieto	olga.on@gmail.com	2023-11-14	2023-12-14
Obres completes de Tísner.	655346b28f6f9c46959a327d	Olga	Olmedo Nieto	olga.on@gmail.com	2023-11-14	2023-12-14
Obres completes de Tísner.	655346b28f6f9c46959a327d	Olga	Olmedo Nieto	olga.on@gmail.com	2023-11-14	2023-12-14

Imagen 41: Vista del menú *See all borrowings*

8. Crear un prestamo

En el menú *Borrowings* > *Add borrowing* se puede crear un prestamos a través de un formulario en el que se pide el email del usuario que pide el libro y el id de la copia que se va a prestar. En un futuro tendremos también la opción de leer el código de barra del libro directamente desde un móvil.

Los dos campos son obligatorios y el email debe coincidir con un formato determinado de tipo xxx@xxxx.xx o xxx.xxx@xxx.xx.

Cuando se registra un préstamo, EzLib hace dos cosas:

1. comprueba que el usuario existe haciendo una búsqueda con el email proporcionado
2. comprueba que la copia está disponible. Si no cumple uno o los dos requisitos, EzLib devuelve un mensaje de error.

Add a new borrowing

Please provide an user email and the copy id that the user will borrow.

Email

Copy id

[Search](#)

Luna Lovegood
luna.lovegood@sunnyview.com
6 d
Gryffindor

Is going to borrow

[Yes, that's correct!](#)

Veinte poemas de amor y una canción desesperada
Pablo Neruda

Copy 6558dad20156d9289fdc58cc is not available

Imagen 42: Mensaje de error al crear un nuevo préstamo

En caso positivo, EzLib nos pide confirmación que tal usuario está pidiendo préstamo de tal libro. Al darle al botón *Yes, that's correct* se crea el préstamo en la base de datos. Se puede comprobar que se ha creado correctamente en el menu *Borrowing > See all borrowings*

Add a new borrowing

Please provide an user email and the copy id that the user will borrow.

Email

Copy id

[Search](#)

Luna Lovegood
luna.lovegood@sunnyview.com
6 d
Gryffindor

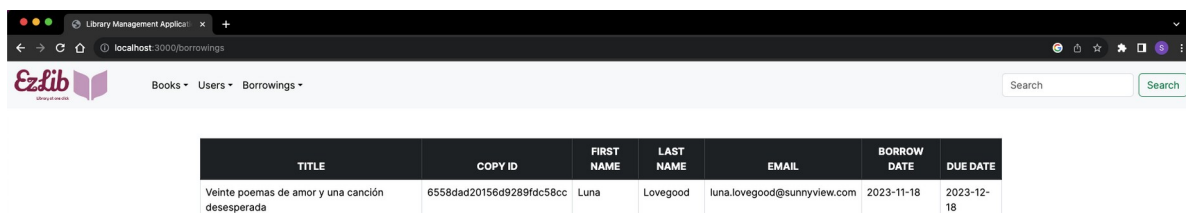
Is going to borrow

[Yes, that's correct!](#)

Veinte poemas de amor y una canción desesperada
Pablo Neruda

Borrowing successfully created

Imagen 43: Mensaje de validación de creación de préstamo



TITLE	COPY ID	FIRST NAME	LAST NAME	EMAIL	BORROW DATE	DUE DATE
Veinte poemas de amor y una canción desesperada	6558dad20156d9289fdc58cc	Luna	Lovegood	luna.lovegood@sunnyview.com	2023-11-18	2023-12-18

Imagen 44: Comprobación que el préstamo se ha creado desde el menú *See all borrowings*

- Manual de instalación

1. Aviso preliminar

EzLib sigue en proceso de desarrollo por lo cual la aplicación no está desplegada todavía. Para poder probar la aplicación en su ordenador habrá que tener instalada algunas herramientas.

2. Requisitos minimos

- Procesador de 64 bits con 2GHz o más
- 4GB de RAM
- Al menos 5GB de espacio libre en el disco duro

3. Instalación de los softwares requeridos

3.1. Instalación de Java 17 LTS

Ir a la [página de Oracle](#) y descargar la versión 17 de Java Development Kit. Tras el JDK instalado comprobar la versión de Java que tenemos instalada en nuestro sistema. Para eso, en Windows abrir el símbolo del sistema pulsando las teclas Windows + R y en la ventana que se abre, escribir el comando `cmd`. En Mac, ir a /Aplicaciones/Utilidades y hacer doble clic en Terminal.

Una vez en el terminal, lanzar el comando `java --version` para comprobar que tenemos Java instalado en nuestro sistema.

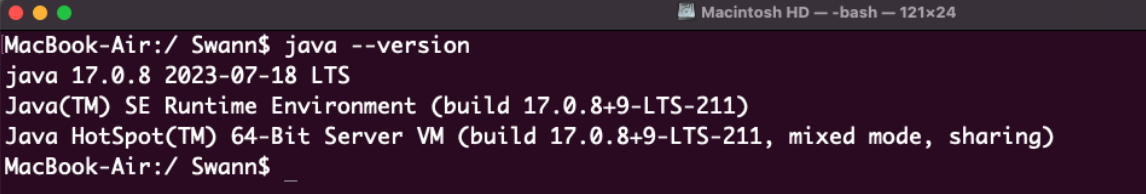
A screenshot of a macOS Terminal window. The title bar at the top reads 'Macintosh HD -- -bash -- 121x24'. The prompt is 'MacBook-Air:/ Swann\$'. The command 'java --version' has been entered, and the output is displayed on several lines: 'java 17.0.8 2023-07-18 LTS', 'Java(TM) SE Runtime Environment (build 17.0.8+9-LTS-211)', 'Java HotSpot(TM) 64-Bit Server VM (build 17.0.8+9-LTS-211, mixed mode, sharing)', and the prompt 'MacBook-Air:/ Swann\$ _'.

Imagen 45: Resultado del comando `java --version`

3.2 Instalar Node.js y NPM (Node Package Manager)

Ir a [la página oficial de Node.js](#) y descargar la última versión LTS correspondiente a su sistema operativo.

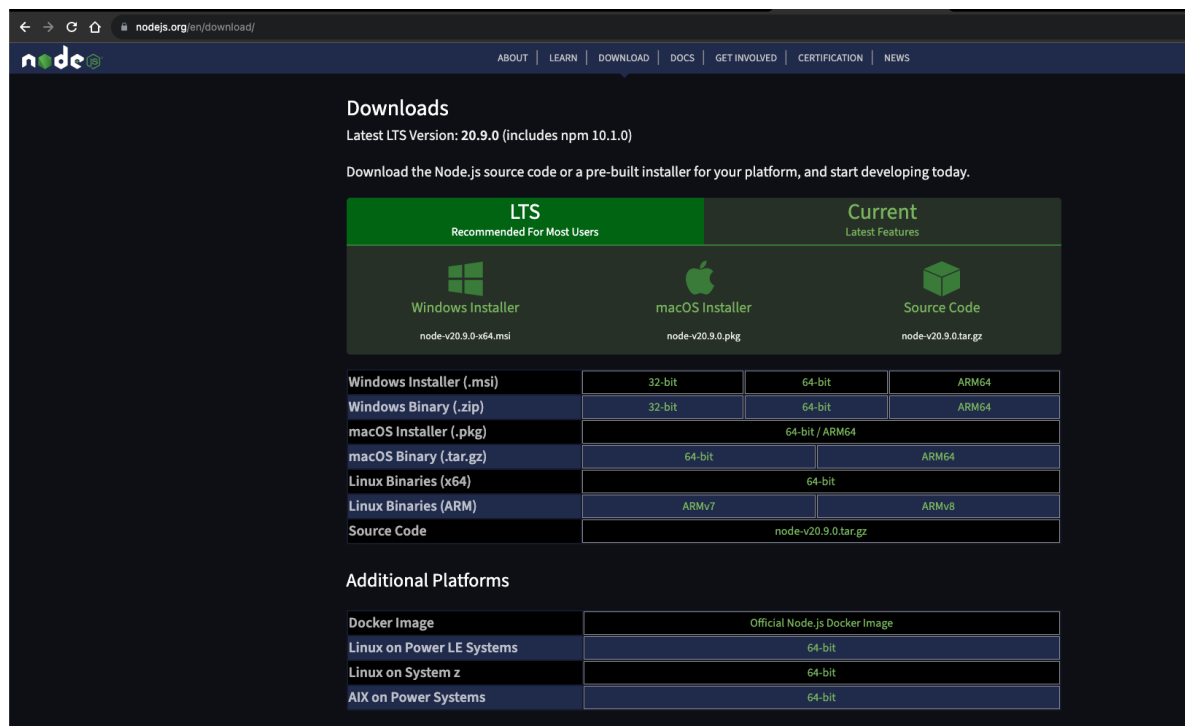


Imagen 46: Pagina de descarga de Node.js

Ejecutar el instalador que se acaba de descargar.

Una vez instalado, abrir el terminal y lanzar el comando `node -v` para comprobar que se ha instalado correctamente Node.js y luego lanzar el comando `npm -v` para comprobar que se ha instalado también npm.

```
MacBook-Air:/ Swann$ node -v
v20.5.1
MacBook-Air:/ Swann$ npm -v
9.8.1
MacBook-Air:/ Swann$
```

Imagen 47: Comandos de comprobación de instalación de Node.js y npm

3.3 Instalar Visual Studio Code

[Descargar Visual Studio Code](#) en la página oficial y lanzar el instalador.

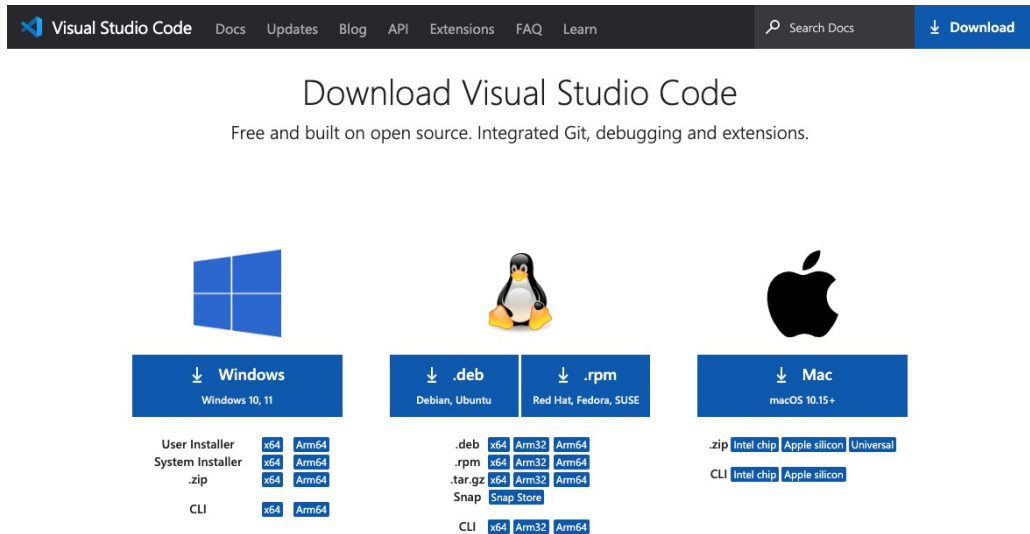


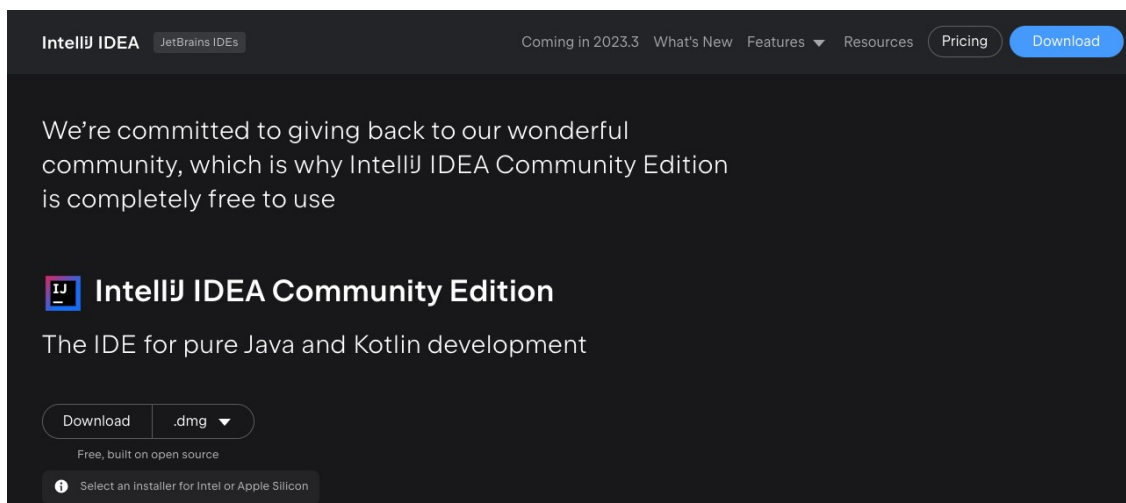
Imagen 48: Pagina de descarga de Visual Studio Code

3.3 Instalar un IDE especializado en Java tipo Eclipse o IntelliJ IDEA

Aquí se puede usar el IDE que más le guste. Existe numerosas opciones en el mercado, Eclipse, IntelliJ IDEA, NetBeans.

En este manual se explicará como ejecutar la API en un servidor local desde IntelliJ.

Descargar la versión Community Edition en [la página oficial de IntelliJ](#). Esta versión es gratis y es más que suficiente para lo que necesitamos.



4. Descargar el proyecto

El directorio raíz del proyecto contiene dos directorios y un archivo pdf:

```
.
├── PI_SwannJulien/
│   ├── back
│   ├── front
│   └── PI_SwannJulien.pdf
```

- **PI_SwannJulien/back:** contiene todo el código para la parte del backend de la aplicación. Este directorio se abrirá en IntelliJ.
- **PI_SwannJulien/front:** contiene todo el código para la parte del frontend de la aplicación. Este directorio se abrirá en Visual Studio Code
- **PI_SwannJulien/PI_SwannJulien.pdf:** corresponde a la presentación del proyecto.

5. Ejecutar la aplicación en un servidor local

5.1. Ejecutar el backend

Abrir IntelliJ y abrir el directorio *back* del proyecto. La primera vez que se abre la aplicación en IntelliJ puede tardar un poco en descargar todas las dependencias. Si las dependencias no se actualizan automáticamente, hacer clic derecho en el archivo *pom.xml* y darle a Maven > Reload project. En Eclipse hacer ALT + F5.

Una vez las dependencias descargadas, ir a */src/main/java/SVLibraryApplication* y hacer un clic derecho sobre la clase seguido de *Run*.

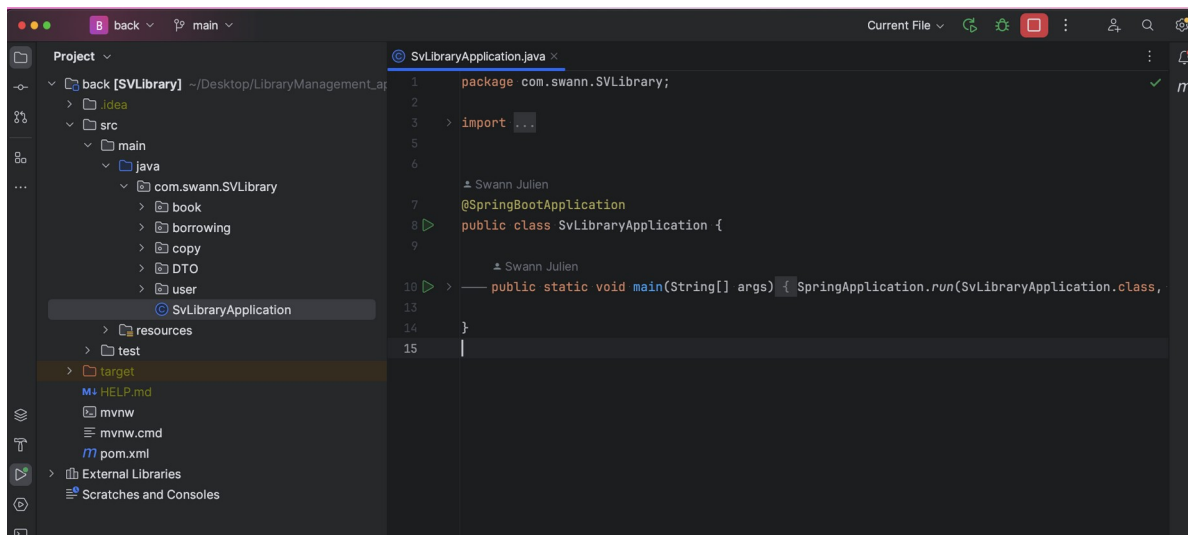
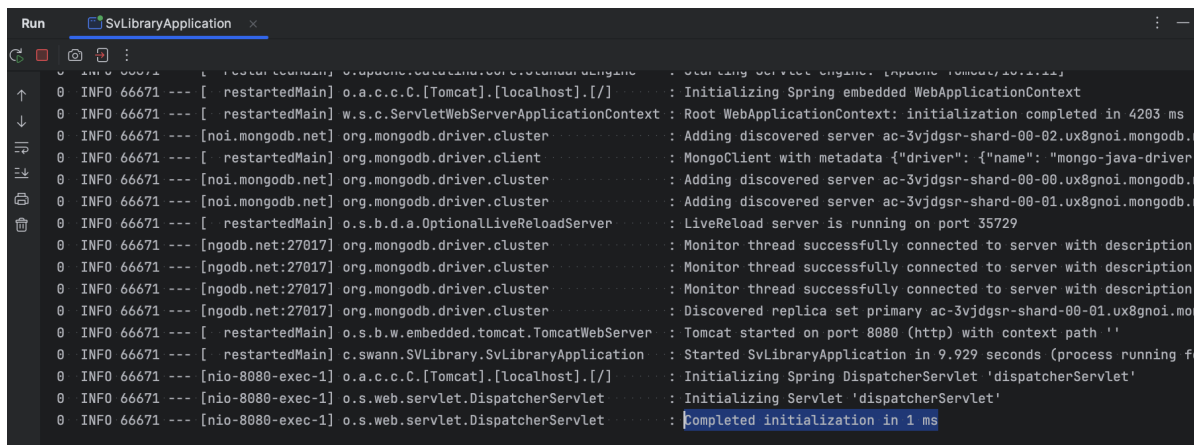


Imagen 49: Clase Main de la API que hay que ejecutar

En la consola del IDE se puede ver cuando la aplicación termina de arrancar con el mensaje *Completed initialization in x ms*.



```
Run SvLibraryApplication x
0 INFO 66671 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
0 INFO 66671 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4283 ms
0 INFO 66671 --- [noi.mongodb.net] org.mongodb.driver.cluster : Adding discovered server ac-3vjdgssr-shard-00-02.ux8gnoi.mongodb.
0 INFO 66671 --- [ restartedMain] org.mongodb.driver.client : MongoClient with metadata {"driver": {"name": "mongo-java-driver
0 INFO 66671 --- [noi.mongodb.net] org.mongodb.driver.cluster : Adding discovered server ac-3vjdgssr-shard-00-00.ux8gnoi.mongodb.
0 INFO 66671 --- [noi.mongodb.net] org.mongodb.driver.cluster : Adding discovered server ac-3vjdgssr-shard-00-01.ux8gnoi.mongodb.
0 INFO 66671 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
0 INFO 66671 --- [ngodb.net:27017] org.mongodb.driver.cluster : Monitor thread successfully connected to server with description
0 INFO 66671 --- [ngodb.net:27017] org.mongodb.driver.cluster : Monitor thread successfully connected to server with description
0 INFO 66671 --- [ngodb.net:27017] org.mongodb.driver.cluster : Monitor thread successfully connected to server with description
0 INFO 66671 --- [ngodb.net:27017] org.mongodb.driver.cluster : Discovered replica set primary ac-3vjdgssr-shard-00-01.ux8gnoi.mo
0 INFO 66671 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
0 INFO 66671 --- [ restartedMain] c.swann.SVLibrary.SVLibraryApplication : Started SvLibraryApplication in 9.929 seconds (process running f
0 INFO 66671 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
0 INFO 66671 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
0 INFO 66671 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```

Imagen 50: Mensaje de finalización de ejecución de la aplicación en el servidor local

Para comprobar que la aplicación ha arrancado, abrir una nueva página en el navegador web y lanzar el servidor local con <http://localhost:8080/>. Tiene que aparecer el mensaje de error siguiente, lo que es totalmente normal.

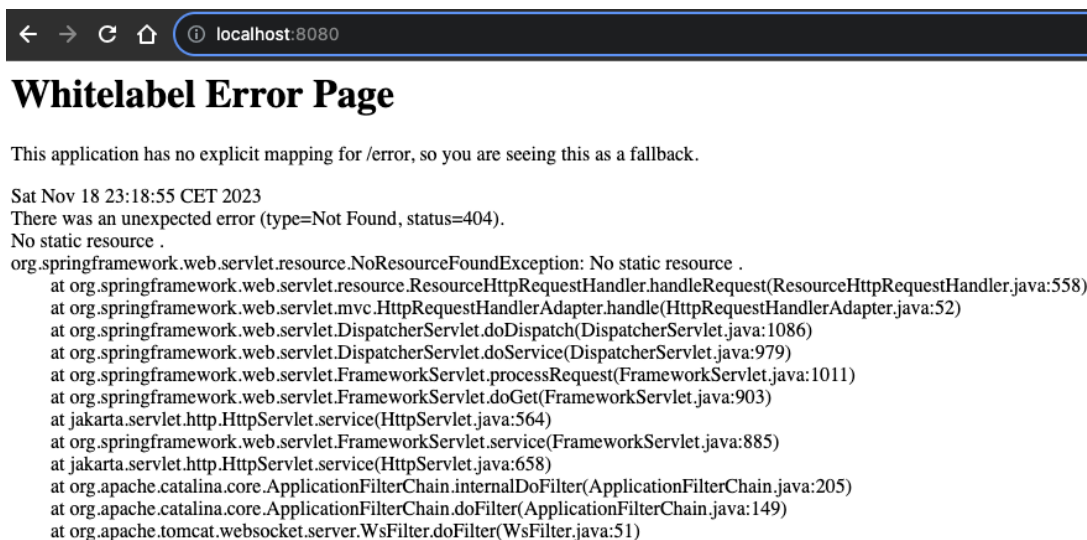


Imagen 51: Comprobación en el navegador que el servidor local funciona

5.2. Ejecutar el frontend

Abrir Visual Studio Code y abrir el directorio `/front`. Ir al menú *Terminal* > new Terminal y lanzar el comando siguiente `npm run build` seguido de `npm run start`.

Abrir una nueva página en el navegador y lanzar el servidor local <http://localhost:3000>.

Tendríamos que ver ahora la página de inicio de EzLib.

Los dos servidores locales tienen que correr a la vez: <http://localhost:8080/> y <http://localhost:3000>.

3.3. Incidencias

3.3.1. Definir un protocolo para resolución de incidencias:

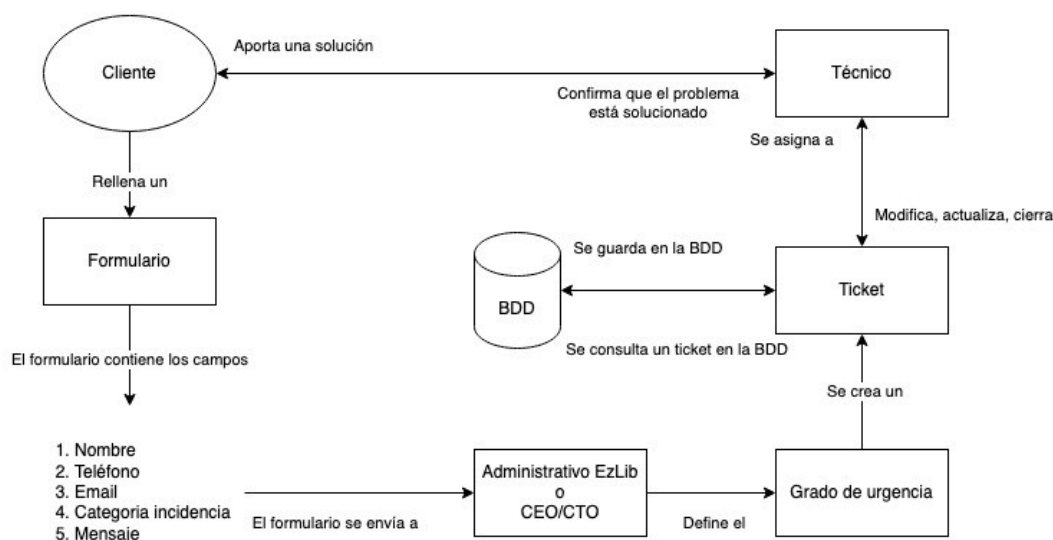
Al tener de momento solo un cliente para su solución de gestión de biblioteca, EzLib gestionará en interno las incidencias que puedan surgir. Sin embargo, EzLib redactó un protocolo de gestión de las incidencias para responder a la mayor brevedad a las solicitudes del Sunny View. Este protocolo también valdrá para sus futuros clientes.

Las distintas etapas del protocolo de gestión de incidencias son las siguientes:

1. El cliente encuentra un problema con la aplicación
2. El cliente rellena un formulario online indicando la incidencia encontrada
3. EzLib recibe el formulario. Si se recibe durante la semana, el administrativo se encarga de gestionarlo. Si se recibe durante un fin de semana o un festivo, lo gestiona uno de los dos socios.
4. Se define el grado de urgencia de la incidencia. Este grado de urgencia define también el día en el que se debe contestar al cliente.
5. Se crea un ticket y se le asigna a un técnico para su resolución
6. El ticket se guarda en un base de datos interna. El técnico actualiza y cierra el ticket cuando se termina la incidencia.

Este protocolo está resumido en el diagrama siguiente:

Protocolo gestión de incidencias



3.3.1.1. Recopilación de información, posible solución, registro

La tabla siguiente recopila las distintas incidencias que se puede encontrar el usuario con la aplicación EzLib y las soluciones que se pueden aportar.

Información de la incidencia	Posible solución	Registro
No se abre la página web de la aplicación. No se puede acceder a la aplicación.	<ol style="list-style-type: none">1. Compruebe que su ordenador está correctamente conectado a internet.2. Compruebe que la url de la aplicación es la correcta.3. Si los dos puntos anteriores son correctos, contacte con el servicio técnico	<p>2 horas máximo para resolver la incidencia.</p> <p>Se abre y se guarda un ticket en la base de datos.</p>
No consigo conectarme con mis credenciales.	<ol style="list-style-type: none">1. Compruebe que su nombre de usuario y contraseña son correctos.2. Reinicialice tu contraseña3. Si no se reconoce el nombre de usuario, contacte con el servicio técnico.	<p>2 horas máximo para resolver la incidencia.</p> <p>Se abre y se guarda un ticket en la base de datos.</p>
No consigo añadir un libro a la biblioteca. Me da un mensaje de error.	<ol style="list-style-type: none">1. Compruebe que el número isbn es el correcto.2. Si el problema sigue, contacte el servicio técnico indicando el mensaje de error.	<p>24h para resolver la incidencia</p> <p>Se abre y se guarda un ticket en la base de datos.</p>
No consigo escanear un código QR.	<ol style="list-style-type: none">1. Compruebe que el código está en buen estado.2. Compruebe que lo está escaneando desde la aplicación móvil de EzLib y no desde otra aplicación.3. Intente imprimir un nuevo código QR buscando el ejemplar correspondiente en la biblioteca.4. Contacte el servicio técnico si los puntos anteriores no permiten arreglar el problema.	<p>24h para resolver la incidencia</p> <p>Se abre y se guarda un ticket en la base de datos.</p>

4. Pruebas y soporte

4.1. Crear documento con las pruebas a realizar

1. Resumen de los métodos a probar

En el estado actual de la aplicación, se usan los siguientes métodos:

Clase	Método	Uso
/src/main/java/SVLibrary/book/ BookService	findAll()	Buscar y retornar todos los libros de la base de datos.
/src/main/java/SVLibrary/book/ BookService	addBookAndCopyToLibrary()	Añadir un libro y una copia del libro en la base de datos.
/src/main/java/SVLibrary/user/ UserService	findAll()	Buscar y retornar todos los usuarios de la base de datos.
/src/main/java/SVLibrary/user/ UserService	addUser()	Añadir un usuario en la base de datos.
/src/main/java/SVLibrary/borrowing/ BorrowingService	findAllBorrowings()	Buscar y retornar todos los usuarios de la base de datos.
/src/main/java/SVLibrary/borrowing/ BorrowingService	addBorrowing()	Añadir un préstamo en la base de datos.

Se ha realizado pruebas unitarias para cada uno de esos métodos con JUnit 4 y Mockito para simular las llamadas a la base de datos.

Todas las pruebas unitarias se pueden ver en el directorio */src/test* del backend.

Clase	Método	Parametros de entrada	Resultado esperado	Resultado
/src/test/java/SVLibrary/book/ BookService	testFindAllBooks()	Ninguno	Retornar los libros mockeados	OK
/src/test/java/SVLibrary/book/ BookService	testFindAllBooksNoData()	Ninguno	No retornar datos sin error	OK
/src/test/java/SVLibrary/book/ BookService	testAddBookAndCopyToLibrary_Success()	Objeto libro con un número ISBN valido.	El ID de la copia creada	KO, no se puede mockear la creación del ID por MongoDB

/src/test/java/ SVLibrary/book/ BookService	testAddBookAndCopyToLibrary_InvalidIsbn10()	Objeto libro con un número ISBN no valido.	IllegalArgumentException	OK
/src/test/java/ SVLibrary/book/ BookService	testAddBookAndCopyToLibrary_DuplicateIsbn()	Objeto libro con un número ISBN existente en la base de datos mockeada.	RuntimeException	OK
/src/test/java/ SVLibrary/user/ UserService	testFindAllUsers()	Ninguno	Retornar los usuarios mockeados	OK
/src/test/java/ SVLibrary/user/ UserService	testAddUser()	Objeto usuario valido	Pasar el test sin error	OK
/src/test/java/ SVLibrary/user/ UserService	testAddUserWithExistingEmail()	Objeto usuario con un email existente en la base de datos	IllegalArgumentException	OK
/src/test/java/ SVLibrary/ borrowing/ BorrowingService	testFindAllBorrowings()	Ninguno	Retornar los prestamos mockeados	OK
/src/test/java/ SVLibrary/ borrowing/ BorrowingService	testAddBorrowing()			No se ha escrito este test. Está por hacer.

4.2. Registro de las pruebas realizadas

Ver la parte sobre Postman y las pruebas de todas las rutas.

4.3. Evaluar que el proyecto cumple todo lo requerido.

Revisar punto a punto lo indicado en el punto 2.2. y comprobar que el proyecto lo cumple todo.

Objetivos	% conseguido	Observaciones
Crear una base de datos para almacenar los libros, los ejemplares, los alumnos así que el estado del libro.	100%	
Crear una API de tipo REST que permita crear, leer,	100%	

actualizar o eliminar las colecciones Libro y Ejemplar en la base de datos.		
Llamar a la API de OpenLibrary para obtener información sobre un libro.	100%	
Modificar la información recibida por la API de OpenLibrary y guardar en la base de datos solo la que se necesita para su funcionamiento.	100%	
Desarrollar la posibilidad de crear un libro manualmente a través de un formulario y añadirlo a la base de datos.	100%	
Llamar la API de GoQR.me para generar un código QR único para cada ejemplar que se añade a la base de datos.	100%	
Desarrollar una funcionalidad que permita leer un código QR para identificar el ejemplar al que pertenece en la base de datos.	0%	Faltará por desarrollar la parte lectura del código QR con el móvil.
Tomar en cuenta todos los mensajes de error posibles y guiar al usuario según el mensaje recibido.	100%	
Diseñar y desarrollar una interfaz usuario Responsive tomando en cuenta todos los tipos de dispositivo.	50%	No se ha completado totalmente pero con la librería Bootstrap ya instalada, no costará mucho tiempo en desarrollar.
Crear una función de búsqueda avanzada de libros según ciertos criterios.	0%	Está totalmente por hacer