FISBook

LEE, DESCUBRE Y DESCARGA

MICROSERVICIO DE DESCARGAS Y LECTURAS

Fundamentos de Ingeniería del Software y Sistemas Cloud

Kristina Lacasta López

Yasmina Moreira Bawazeer

ÍNDICE

1. Descripción del proyecto

La aplicación **Fisbook** está concebida para ofrecer una experiencia integral y fluida en la gestión de una biblioteca digital, brindando a los usuarios un acceso cómodo y organizado a una vasta colección de libros electrónicos. La plataforma no solo facilita la exploración y lectura, sino que también promueve una interacción enriquecedora entre los usuarios, permitiéndoles personalizar su experiencia de lectura. Entre sus características más destacadas se incluyen:

1. **Gestión de Libros**: los usuarios pueden navegar por una amplia biblioteca de títulos, cada uno con detalles completos sobre su contenido, autor, sinopsis y más. Además, pueden descargar libros en diversos formatos como ePub y PDF, adaptándose a sus preferencias de lectura.

2. **Listas de Lectura**: la plataforma permite a los usuarios crear listas de libros personalizadas, clasificándolos por géneros, temáticas o preferencias personales. Esta funcionalidad facilita la organización de las lecturas y ayuda a los usuarios a seguir su progreso.

3. **Reseñas y Calificaciones**: fomentando una comunidad activa, los usuarios tienen la posibilidad de dejar reseñas y calificar los libros que leen. A su vez, pueden interactuar con las reseñas y valoraciones de otros, enriqueciendo el proceso de descubrimiento y recomendación de nuevos títulos.

Para soportar estas funcionalidades, **Fisbook** adopta una arquitectura basada en **microservicios**, lo que le confiere modularidad, escalabilidad y flexibilidad. Cada uno de los microservicios se especializa en una función específica, optimizando la eficiencia del sistema, facilitando la integración de nuevas características y asegurando su fácil mantenimiento. Los cinco microservicios principales de la plataforma son:

- **Usuarios***: su principal función es la creación, autenticación y gestión de perfiles de los usuarios, garantizando que cada persona tenga su propia cuenta personalizada y segura. Además de permitir el inicio de sesión y la autenticación, ofrece opciones de personalización del perfil, como la actualización de la información personal,

- **Listas de Lectura**: el microservicio de *Listas de Lectura* proporciona a los usuarios la capacidad de organizar y personalizar su experiencia literaria mediante la creación y gestión de listas de libros. Estas listas pueden estar basadas en géneros, autores, temáticas específicas o cualquier otra preferencia personal. Los usuarios pueden añadir libros a sus listas, editarlas, eliminarlas o compartirlas con otros, lo que fomenta la interacción y el intercambio de recomendaciones.

- **Catálogo**: el microservicio de Catálogo es el encargado de administrar el conjunto de libros disponibles en la plataforma, brindando acceso a una vasta colección organizada de títulos. Este servicio se encarga de la búsqueda y filtrado eficiente de libros, permitiendo que los usuarios encuentren rápidamente lo que buscan según criterios como autor, género, popularidad o incluso las últimas actualizaciones.

- **Reseñas**: facilita la creación, visualización y administración de reseñas y calificaciones de libros, lo que permite a los usuarios compartir sus opiniones

***Microservicio implementado por la pareja**

sobre los libros que han leído y ver las valoraciones de otros. Este sistema de reseñas fomenta la participación activa de la comunidad, creando un espacio para el debate y el intercambio de recomendaciones literarias.

- **Descargas y Lecturas***: este microservicio se encarga de gestionar las descargas de libros y el seguimiento de las actividades de lectura de los usuarios, proporcionando un control efectivo del uso de la plataforma. Los usuarios tienen la opción de descargar libros para su lectura en dispositivos electrónicos como eBook o en sus propios dispositivos, o bien leer directamente dentro de la aplicación. Esto permite una experiencia de lectura flexible, ya sea en línea o sin conexión, adaptándose a las necesidades y preferencias individuales de cada usuario.

Este enfoque basado en microservicios asegura que cada componente pueda escalar de manera independiente, permitiendo una mejora continua y adaptaciones ágiles a las necesidades cambiantes de los usuarios y de la propia plataforma. Además, la modularidad de los microservicios facilita el mantenimiento y las actualizaciones, manteniendo **Fisbook** siempre a la vanguardia de la tecnología y las expectativas de los usuarios.

Se puede obtener más información sobre el Customer Agreement de Fisbook en el **Anexo 3.Customer Agreement (CA).**

1.1 Características Microservicio Básico

Este microservicio ha sido diseñado para gestionar las descargas de libros y las lecturas en la aplicación **FISBook**, cumpliendo con todas las características necesarias para ser considerado un microservicio básico, así como varias de las características exigidas para un microservicio avanzado. A continuación, se detallan las características implementadas. Toda la evidencia de la implementación está disponible en el repositorio de GitHub **'fisbookBE-downloads'.**

Github: https://github.com/FIS-Book/fisbookBE-downloads.git

- **API RESTful con métodos GET, POST, PUT y DELETE (/routes).** El microservicio ofrece una API REST que permite realizar las siguientes operaciones:
    - **Descargas**: consultar (GET), crear (POST), y eliminar (DELETE) registros de descargas. No se consideró necesario incluir un método PUT, ya que no tiene sentido actualizar un registro de descarga existente. Los registros de descargas son transacciones únicas, y cualquier cambio implicaría la creación de un nuevo registro en lugar de modificar uno ya existente.
    - **Lecturas**: consultar (GET), crear (POST), actualizar (PUT) y eliminar (DELETE) lecturas asociadas a los usuarios.
- **Mecanismo de Autenticación (/authentication).** Implementamos autenticación mediante JWT (JSON Web Tokens), asegurando que solo los usuarios autenticados puedan acceder a los endpoints protegidos, como los relacionados con la gestión de descargas y lecturas.

- **Frontend (/src/feature/users).** La lógica de descargas y lecturas está diseñada para ser consumida por un frontend. Este frontend está integrado con otras partes de la aplicación FISBook, permitiendo una experiencia de usuario completa. Se puede encontrar el frontend en el repositorio de github **'**frontend-fisbook'.

Frontend: https://github.com/FIS-Book/frontend-fisbook.git

- **Despliegue en la Nube.** El microservicio y su frontend está preparado para su despliegue en la nube con Azure. Su integración con Docker permite un despliegue sencillo y escalable en un entorno cloud.

Base URL: http://57.152.88.187

- **Base URL y Versionado.** La API está accesible a través de una URL versionada que sigue las buenas prácticas de API REST. Esto facilita el mantenimiento y la evolución de las funcionalidades del microservicio.

URL Versionada: http://57.152.88.187/api/v1/read-and-download

- **Documentación de la API.** La API está documentada con Swagger, proporcionando una interfaz interactiva para visualizar y probar los endpoints. Además, se cuenta con un README detallado y documentación complementaria.

Swagger: http://57.152.88.187/api/v1/read-and-download/api-docs/#/

- **Persistencia en MongoDB (db.js).** Utilizamos MongoDB Atlas como base de datos NoSQL para almacenar información sobre las descargas y lecturas. Esto asegura un manejo eficiente y escalable de los datos.
- **Validación de Datos (/models).** Los datos de las descargas y lecturas se validan antes de almacenarlos en la base de datos mediante el uso de mongoose.
- **Gestión de Código Fuente e Integración Continua (/.github/workflows).** El código del proyecto está alojado en un repositorio de GitHub, siguiendo GitHub Flow. Además, utilizamos GitHub Actions para realizar pruebas e integración continua automáticamente en cada commit, garantizando la calidad del código.
- **Dockerización (Dockerfile).** El microservicio está empaquetado en un contenedor Docker, facilitando su despliegue en diferentes entornos y asegurando la portabilidad.
- **Pruebas de Componente (/tests).** Se han implementado pruebas unitarias y de integración con Jest y Supertest, cubriendo tanto los casos de éxito como los escenarios de fallo. Esto garantiza el correcto funcionamiento de los endpoints del microservicio.

- **Pruebas de Integración con la Base de Datos (/tests/integration).** Se han realizado pruebas de integración para validar la interacción entre la API y la base de datos MongoDB, asegurando la consistencia y el correcto manejo de los datos.

1.2 Características Microservicio Avanzado

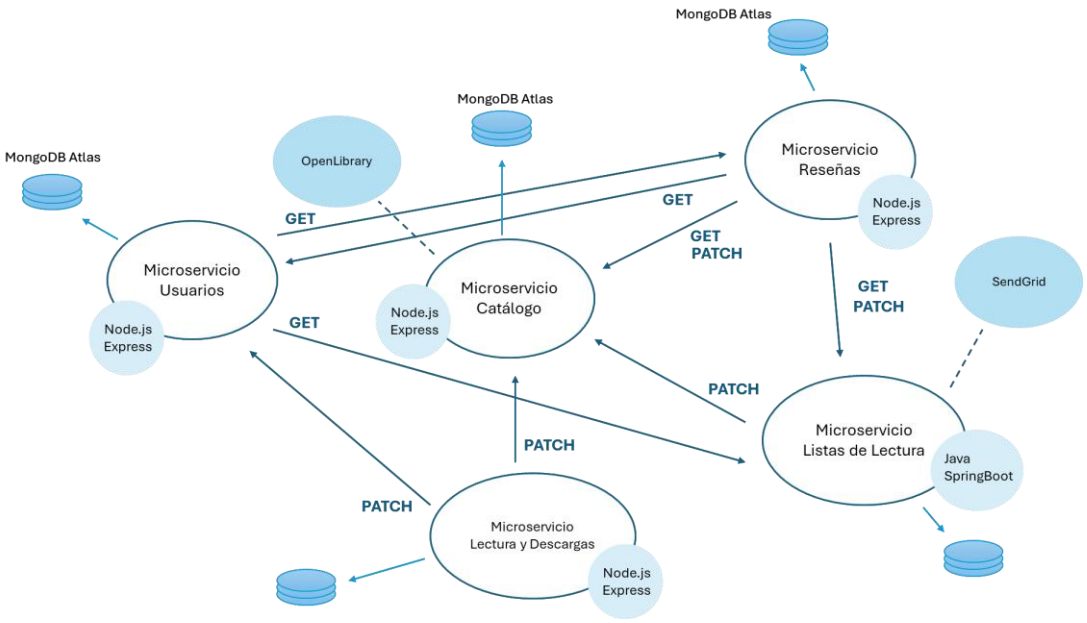
Además de cumplir con las características de un microservicio básico, se han incorporado funcionalidades avanzadas para mejorar su rendimiento y capacidades**:**

- **Autenticación JWT Avanzada (/authentication).** Se ha implementado un sistema de autenticación robusto basado en JWT, que protege las rutas y garantiza que solo los usuarios autorizados puedan realizar operaciones en la API.
- **Frontend con Rutas y Navegación.** Aunque el microservicio no incluye un frontend autónomo, está diseñado para integrarse con el frontend principal de FISBook, proporcionando las funcionalidades necesarias para gestionar las descargas y lecturas con navegación y rutas dinámicas.

Dado que no hemos tenido tiempo de realizar una característica más del microservicio Avanzado, nos presentamos al nivel hasta 5 puntos.

2. Arquitectura e interacción de microservicios

En la siguiente figura se ilustran las comunicaciones entre los distintos microservicios definidos. Como se puede observar, estos microservicios interactúan entre sí para garantizar la prestación de diversos servicios de manera coordinada y eficiente. La arquitectura está diseñada para optimizar la colaboración entre los componentes, asegurando una experiencia fluida y funcional para los usuarios.



El microservicio de ***Descargas y Lecturas*** es una pieza importante que permite a los usuarios descargar libros y realizar lecturas en línea directamente desde la plataforma. A continuación, se detalla su relación con los demás microservicios:

- **Microservicio de Usuarios.** El microservicio de Descargas y Lecturas depende estrechamente del microservicio de Usuarios para garantizar un acceso seguro y personalizado a las funcionalidades de descargas. La relación entre ambos se manifiesta de la siguiente manera:
    - **Autenticación de Usuarios:** los usuarios deben estar autenticados mediante el microservicio de Usuarios para poder descargar libros o acceder a lecturas. La autenticación se gestiona mediante JWT, que valida las solicitudes a las rutas protegidas del microservicio de Descargas.
    - **Asociación de Descargas a Usuarios:** cada descarga o lectura está asociada al ID del usuario que la realiza, lo que permite llevar un registro individualizado de las actividades de cada usuario, como el historial de descargas o los libros que ha leído en línea.
    - **Validación de Permisos:** antes de permitir una descarga, el microservicio de Descargas consulta al microservicio de Usuarios para verificar si el usuario tiene acceso al recurso solicitado, según su rol o permisos.
- **Microservicio de Catálogo de Libros.** El microservicio de Descargas y Lecturas establece una comunicación dinámica con el microservicio de Catálogo de Libros

para garantizar una gestión eficiente y actualizada de los recursos disponibles en la plataforma. La relación entre ambos microservicios se basa en los siguientes puntos clave:

- **Actualización de descargas:** cada vez que un usuario descarga un libro desde la plataforma, el microservicio de Descargas y Lecturas notifica al microservicio de Catálogo de Libros para incrementar el contador de descargas del título correspondiente. Esto asegura que el catálogo refleje en tiempo real la popularidad y el uso de los libros, mejorando las métricas internas y la experiencia de los usuarios al ver estadísticas actualizadas.

3. Descripción de la API REST

3.1. Base URL

La API del microservicio Descargas y Lecturas está documentada y accesible a través de Swagger, lo que facilita su comprensión e integración por parte de los desarrolladores. Además, el microservicio está desplegado y disponible para su uso en la siguiente URL base.

Base URL: http://57.152.88.187/api/v1/read-and-download/api-docs/#/

Desde esta dirección, es posible explorar la documentación detallada de los endpoints, visualizar las solicitudes y respuestas esperadas, y realizar pruebas directamente en el entorno de desarrollo o producción. Esta configuración asegura que la interacción con el microservicio sea clara y accesible.

3.2. Tecnologías y Herramientas Usadas

Para la implementación de esta arquitectura se han implementado las siguientes tecnologías:

- **Node.js**: plataforma de desarrollo utilizada como base del microservicio.
- **Express.js**: framework para la creación de APIs REST.
- **MongoDB + Mongoose**: base de datos NoSQL utilizada junto con un ORM para modelado de datos.
- **Swagger**: herramienta para la documentación interactiva de la API.
- **Docker**: utilizado para empaquetar y desplegar el microservicio en contenedores.
- **Jest + Supertest**: frameworks para pruebas unitarias y de integración.
- **dotenv**: para la gestión de variables de entorno.
- **CORS**: configuración para manejo seguro de solicitudes entre dominios.

3.3. Estructura el proyecto

La estructura interna del proyecto queda de la siguiente forma:

- **authentication/:** contiene la lógica de autenticación y la configuración de JWT para gestionar accesos seguros.
- **bin/:** contiene la configuración para iniciar el servidor, como el archivo www.
- **models/:** define los esquemas de los datos utilizados en MongoDB, como el modelo user.js.

- **routes/:** maneja las rutas de la API REST, incluyendo rutas base (index.js) y específicas de usuarios (users.js).
- **tests/:** directorio de pruebas automatizadas con Jest, organizadas por módulos (user.test.js y auth.test.js).
- **app.js:** archivo principal que configura la aplicación Express, conecta rutas y middleware.
- **Dockerfile:** contiene las instrucciones para crear y ejecutar el contenedor Docker.
- **db.js:** conexión a la base de datos MongoDB usando mongoose. Establece la conexión con MongoDB Atlas a través de la URI configurada en process.env.MONGO_URI_DOWLOADS. También maneja los errores de conexión y confirma la conexión exitosa a la base de datos.
- **package.json:** especifica las dependencias del proyecto y scripts de ejecución.

3.4. Endpoints para la gestión de descargas

Obtener todas las descargas

- **GET /api/v1/downloads**
- **Descripción:** Devuelve una lista de todas las descargas registradas.
- **Cuerpo de la solicitud:** No requiere.
- **Respuesta:**
  - 200 OK: Lista de descargas devuelta exitosamente.
  - 500 Internal Server Error: Error al recuperar las descargas.

Obtener una descarga por ID

- **GET /api/v1/downloads/:id**
- **Descripción:** Obtiene una descarga específica basada en su ID.
- **Parámetro de ruta:**
  - id: ID único de la descarga.
- **Respuesta:**
  - 200 OK: Descarga encontrada.
  - 404 Not Found: La descarga con el ID especificado no existe.
  - 500 Internal Server Error: Error al procesar la solicitud.

Registrar una nueva descarga

- **POST /api/v1/downloads**
- **Descripción:** Crea un nuevo registro de descarga.
- **Cuerpo de la solicitud:**
    - bookId: ID del libro descargado.
    - userId: ID del usuario que realiza la descarga.
- **Respuesta:**
    - 201 Created: Descarga registrada exitosamente.
    - 400 Bad Request: Datos inválidos o faltantes.
    - 500 Internal Server Error: Error al registrar la descarga.

Eliminar una descarga

- **DELETE /api/v1/downloads/:id**
- **Descripción:** Elimina una descarga específica por su ID.
- **Parámetro de ruta:**
    - id: ID único de la descarga a eliminar.
- **Respuesta:**
    - 200 OK: Descarga eliminada exitosamente.
    - 404 Not Found: La descarga con el ID especificado no existe.
    - 500 Internal Server Error: Error al eliminar la descarga.

Contar el número total de descargas

- **GET /api/v1/downloads/count**
- **Descripción:** Devuelve el número total de descargas registradas en el sistema.
- **Respuesta:**
    - 200 OK: Número total de descargas devuelto.
    - 500 Internal Server Error: Error al procesar la solicitud.

Contar descargas de un usuario específico

- **GET /api/v1/downloads/user/:userId/count**

- **Descripción:** Devuelve el número total de descargas realizadas por un usuario específico.
- **Parámetro de ruta:**
    - userId: ID del usuario.
- **Respuesta:**
    - 200 OK: Número total de descargas del usuario devuelto.
    - 404 Not Found: El usuario no existe o no tiene descargas registradas.
    - 500 Internal Server Error: Error al procesar la solicitud.

3.5. Endpoints para la gestión de lecturas

Obtener todas las lecturas

- **GET /api/v1/onlineReadings**
- **Descripción:** Devuelve una lista de todas las lecturas registradas.
- **Cuerpo de la solicitud:** No requiere.
- **Respuesta:**
    - 200 OK: Lista de lecturas devuelta exitosamente.
    - 500 Internal Server Error: Error al recuperar las lecturas.

Registrar una nueva lectura

- **POST /api/v1/onlineReadings**
- **Descripción:** Registra una nueva lectura para un usuario.
- **Cuerpo de la solicitud:**
    - bookId: ID del libro leído.
    - userId: ID del usuario que realiza la lectura.
- **Respuesta:**
    - 201 Created: Lectura registrada exitosamente.
    - 400 Bad Request: Datos inválidos o faltantes.
    - 500 Internal Server Error: Error al registrar la lectura.

Obtener una lectura por ID

- **GET /api/v1/onlineReadings/:id**
- **Descripción:** Devuelve los detalles de una lectura específica basada en su ID.
- **Parámetro de ruta:**
  - id: ID único de la lectura.
- **Respuesta:**
  - 200 OK: Lectura encontrada.
  - 404 Not Found: La lectura con el ID especificado no existe.
  - 500 Internal Server Error: Error al procesar la solicitud.

Eliminar una lectura

- **DELETE /api/v1/onlineReadings/:id**
- **Descripción:** Elimina una lectura específica por su ID.
- **Parámetro de ruta:**
  - id: ID único de la lectura a eliminar.
- **Respuesta:**
  - 200 OK: Lectura eliminada exitosamente.
  - 404 Not Found: La lectura con el ID especificado no existe.
  - 500 Internal Server Error: Error al eliminar la lectura.

Actualizar el estado de una lectura

- **PUT /api/v1/onlineReadings/:id**
- **Descripción:** Actualiza la información de una lectura específica.
- **Parámetro de ruta:**
  - id: ID único de la lectura.
- **Cuerpo de la solicitud:**
  - progress: Progreso del usuario en la lectura (por ejemplo, porcentaje completado).
- **Respuesta:**
  - 200 OK: Lectura actualizada exitosamente.
  - 400 Bad Request: Datos inválidos o faltantes.
  - 404 Not Found: La lectura con el ID especificado no existe.

- 500 Internal Server Error: Error al procesar la solicitud.

3.6. Modelo de datos

Los modelos de datos del microservicio de *Descargas y lecturas* están definidos en los archivos ***onlineReadings.js*** *y* ***downloads.js*** dentro de la carpeta ***models***. Este esquema se utiliza para representar los datos de los usuarios en la base de datos de MongoDB. A continuación, se describen los campos y restricciones aplicadas a cada atributo del modelo.

Tabla 1: Esquema de Datos de Descargas

Campo	Tipo de Dato	Descripción	Ejemplo
userId	String	ID del usuario que realizó la descarga.	"605c72ef153207001f786e2"
isbn	String	ISBN del libro descargado.	"9781234567890"
titulo	String	Título del libro descargado.	"El Gran Libro"
autor	String	Autor del libro descargado.	"Juan Pérez"
idioma	String	Idioma del libro descargado.	"es"
fecha	String	Fecha de la descarga en formato 'YYYY-MM-DD'.	"2025-01-07"
formato	String	Formato del archivo descargado (PDF, EPUB).	"PDF"

Tabla 2: Restricciones por Atributo y Validaciones Implementadas en Descargas

Campo	Restricciones	Validaciones Implementadas
userId	Requerido (required: true)	Valor debe ser un ObjectId válido de MongoDB.
isbn	Requerido (required: true)	El valor debe ser un ISBN válido (formato ISBN-10 o ISBN-13).
titulo	Requerido (required: true)	Longitud mínima: 3 caracteres, máxima: 121 caracteres.
autor	Requerido (required: true)	El valor no puede ser un string vacío.
idioma	Requerido (required: true)	El valor debe ser uno de los siguientes: 'en', 'es', 'fr', 'de', 'it', 'pt'.
fecha	Fecha por defecto al momento de creación (default: Date.now)	Validación de que el valor sea una fecha válida
formato	Requerido (required: true)	El valor debe ser uno de los siguientes: 'PDF', 'EPUB'

Tabla 3: Esquema de Datos de Lecturas

Campo	Tipo de Dato	Descripción	Ejemplo
userId	String	ID del usuario que realizó la descarga.	"605c72ef153207001f786e2"
isbn	String	ISBN del libro descargado.	"9781234567890"
titulo	String	Título del libro descargado.	"El Gran Libro"
autor	String	Autor del libro descargado.	"Juan Pérez"
idioma	String	Idioma del libro descargado.	"es"
fecha	String	Fecha de la descarga en formato 'YYYY-MM-DD'.	"2025-01-07"
formato	String	Formato del archivo descargado (PDF, EPUB).	"PDF"

Tabla 4: Restricciones por Atributo y Validaciones Implementadas en Lecturas

Campo	Restricciones	Validaciones Implementadas
userId	Requerido (required: true)	Valor debe ser un ObjectId válido de MongoDB.
isbn	Requerido (required: true)	El valor debe ser un ISBN válido (formato ISBN-10 o ISBN-13).
titulo	Requerido (required: true)	Longitud mínima: 3 caracteres, máxima: 121 caracteres.
autor	Requerido (required: true)	El valor no puede ser un string vacío.
idioma	Requerido (required: true)	El valor debe ser uno de los siguientes: 'en', 'es', 'fr', 'de', 'it', 'pt'.
fecha	Fecha por defecto al momento de creación (default: Date.now)	Validación de que el valor sea una fecha válida
formato	Requerido (required: true)	El valor solo puede ser 'PDF'.

3.7. Configuración

La configuración del proyecto se gestiona mediante un archivo .env que contiene las variables de entorno necesarias para la conexión a servicios externos y la personalización de la aplicación. Utilizando un archivo de configuración basado en variables de entorno, se mantiene la seguridad y la flexibilidad al permitir que los valores sensibles, como credenciales y configuraciones específicas del entorno de despliegue, no se incluyan directamente en el código fuente.

A continuación, se detallan algunas de las variables de entorno utilizadas en este microservicio:

MONGO_URI_DOWNLOADS:

- Descripción: URI de conexión a la base de datos MongoDB donde se almacenan los datos de los usuarios.
- Ejemplo: mongodb+srv://<username>:<password>@cluster0.mongodb.net/fisbook-users
- Esta variable se utiliza para establecer la conexión con la base de datos principal de la aplicación, en la cual se almacenan los datos de los usuarios.

MONGO_URI_DOWNLOADS_TESTS:

- Descripción: URI de conexión a una base de datos MongoDB separada, utilizada para realizar pruebas.
- Ejemplo: mongodb+srv://<username>:<password>@cluster0.mongodb.net/test?retryWrites=true&w=majority&appName=test
- Esta variable está dirigida al entorno de pruebas, asegurando que las operaciones de test no afecten a la base de datos de producción.

JWT_SECRET:

- Descripción: Clave secreta utilizada para firmar y verificar los JSON Web Tokens (JWT), que se utilizan para la autenticación y autorización de los usuarios.
- Ejemplo: your_jwt_secret_key
- Este valor debe ser único y confidencial, ya que es utilizado para verificar la autenticidad de los tokens emitidos durante la autenticación de usuarios.

PORT:

- Descripción: Puerto en el que la aplicación se ejecutará durante el desarrollo.
- Ejemplo: 3000
- Permite especificar el puerto en el que el servidor de desarrollo escuchará las solicitudes. Esto es útil cuando se tienen varios servicios corriendo localmente.

BASE_URL:

- Descripción: URL base para realizar solicitudes a otros microservicios.
- Ejemplo: http://api.fisbook.com
- Esta variable es clave cuando la aplicación necesita comunicarse con otros servicios a través de la red. La BASE_URL define la ubicación base de las APIs que serán consumidas por el microservicio.

4. Despliegue en la nube

El microservicio de ***Descargas y Lecturas*** ha sido desplegado en un entorno de producción en la nube utilizando **Azure** como plataforma cloud. La arquitectura del sistema está basada en una estructura de **microservicios**, lo que permite la escalabilidad y flexibilidad. Los microservicios y el frontend están gestionados a través de un **cluster de Kubernetes**, lo que facilita la orquestación, el escalado y el despliegue continuo de los servicios.

Pasos realizados para el despliegue:

1. **Creación de los servicios de microservicios**:
    - Se han creado contenedores Docker para cada microservicio, lo que permite ejecutar los servicios en entornos aislados y portátiles.
    - Los microservicios son responsables de funcionalidades específicas como la autenticación de usuarios, la gestión de catálogos de libros, la gestión de listas de lectura, la gestión de reseñas y las descargas/lecturas.
2. **Configuración del cluster de Kubernetes**:
    - Se ha configurado un cluster de Kubernetes en Azure para orquestar los microservicios.
    - Se asegura la correcta distribución y escalabilidad de los microservicios según la carga de trabajo y la demanda.
3. **Implementación del Ingress Controller**:
    - Se ha implementado un **Ingress** para enrutar el tráfico de red a los microservicios adecuados dentro del cluster de Kubernetes.
    - El Ingress permite la exposición de los servicios de manera segura, gestionando el tráfico HTTP/HTTPS entrante hacia los endpoints adecuados.
4. **Despliegue continuo y automatización**:
    - Se ha configurado un pipeline de **CI/CD** en Azure DevOps para facilitar el despliegue continuo del código. Esto permite realizar cambios rápidamente y desplegar nuevas versiones de los microservicios sin interrumpir el servicio.
5. **Monitoreo y mantenimiento**:
    - Se ha implementado una solución de monitoreo para supervisar el rendimiento y la salud de los microservicios desplegados en la nube, garantizando una alta disponibilidad y tiempos de respuesta óptimos para los usuarios.

Este enfoque de microservicios con Kubernetes y Azure asegura que la aplicación pueda escalar de forma eficiente y se mantenga disponible incluso bajo altas cargas de tráfico.

5. Testing

En este apartado explicamos el enfoque global de testing utilizado en el proyecto, donde abordamos tanto las pruebas de componentes como las de integración, con el objetivo de asegurar la calidad y el funcionamiento del sistema en su conjunto. Además, hemos implementado un proceso de CI/CD mediante **GitHub Actions** para automatizar las pruebas, garantizando que los tests se ejecuten de forma continua y eficiente en cada actualización del código.

5.1. Testing de Componentes

El testing de componentes se centra en la verificación individual de las unidades funcionales dentro de la aplicación para garantizar que cada componente se comporta como se espera de manera aislada. Esto implica probar funciones, métodos y módulos por separado, simulando diferentes condiciones de entrada y asegurando que los resultados sean los correctos.

5.1.1. Descargas

1. **GET /api/v1/read-and-download/downloads**

- **Objetivo:** Validar la funcionalidad de obtención de todas las descargas.
- **Pruebas:**
    - **Caso 1:** Si existen descargas en la base de datos, se debe retornar un código 200 con la lista de descargas.
    - **Caso 2:** Si no existen descargas, la respuesta debe ser un código 200 con un array vacío.
    - **Caso 3:** Si ocurre un error inesperado en la base de datos, se debe retornar un error 500 con el mensaje correspondiente.

2. **GET /api/v1/read-and-download/downloads/:id**

- **Objetivo:** Validar la funcionalidad de obtener una descarga específica por ID.
- **Pruebas:**
    - **Caso 1:** Si la descarga existe, debe retornar un código 200 con los detalles de la descarga.
    - **Caso 2:** Si la descarga no se encuentra, debe retornar un error 404 con el mensaje "Descarga no encontrada".
    - **Caso 3:** Si ocurre un error inesperado, se debe retornar un error 500 con el mensaje correspondiente.

3. **POST /api/v1/read-and-download/downloads**

- **Objetivo**: Validar la creación de una nueva descarga.
- **Pruebas**:

- **Caso 1:** Si la descarga se crea correctamente, debe retornar un código 201 con el objeto de la descarga.
- **Caso 2:** Si faltan datos obligatorios en la solicitud, la respuesta debe ser un error 400 con el mensaje "Faltan datos obligatorios".

4. **DELETE /api/v1/read-and-download/downloads/:id**

- **Objetivo**: Validar la eliminación de una descarga por ID.
- **Pruebas**:
  - **Caso 1:** Si la eliminación es exitosa, debe retornar un código 200 con el mensaje "Descarga eliminada".
  - **Caso 2:** Si no se encuentra la descarga, debe retornar un error 404 con el mensaje "Descarga no encontrada".
  - **Caso 3:** Si ocurre un error en la eliminación, debe retornar un error 500.

5. **GET /api/v1/read-and-download/downloads/count/:isbn**

- Objetivo: Validar la funcionalidad de obtener y actualizar el conteo de descargas de un libro.
- Pruebas:
  - **Caso 1:** Si no se proporciona un token, debe retornar un error 401 con el mensaje adecuado.
  - **Caso 2:** Si no se encuentran descargas para el ISBN, debe retornar un error 404 con el mensaje adecuado.
  - **Caso 3:** Si el conteo se actualiza correctamente, debe retornar un código 200 con el nuevo número de descargas.
  - **Caso 4:** Si ocurre un error interno en el servidor, debe retornar un error 500 con el mensaje correspondiente.

5.1.2. Lecturas

1. **GET /api/v1/read-and-download/onlineReadings**

- Objetivo: Validar la funcionalidad de obtener todas las lecturas en línea.
- Pruebas:
  - **Caso 1**: Si se obtienen correctamente las lecturas, debe retornar un código 200 OK.
  - **Caso 2:** Si ocurre un error en la base de datos, debe retornar un error 500 con el mensaje correspondiente.

2. **GET /api/v1/read-and-download/onlineReadings/:id**

- Objetivo: Validar la funcionalidad de obtener una lectura específica por ID.
- Pruebas:

- **Caso 1:** Si la lectura existe, debe retornar un código 200 OK con los detalles de la lectura.
- **Caso 2:** Si no se encuentra la lectura, debe retornar un error 404 con el mensaje adecuado.
- **Caso 3:** Si ocurre un error en la base de datos, debe retornar un error 500.

3. **POST /api/v1/read-and-download/onlineReadings**

- Objetivo: Validar la creación de una nueva lectura en línea.
- Pruebas:
  - **Caso 1:** Si la lectura se crea correctamente, debe retornar un código 201 con los datos de la lectura.
  - **Caso 2:** Si el título es demasiado corto, debe retornar un error 400.
  - **Caso 3:** Si el idioma no es válido, debe retornar un error 400.
  - **Caso 4:** Si el formato no es válido, debe retornar un error 400.
  - **Caso 5:** Si ocurre un error al guardar la lectura, debe retornar un error 500.

4. **DELETE /api/v1/read-and-download/onlineReadings/:id**

- Objetivo: Validar la eliminación de una lectura en línea por ID.
- Pruebas:
  - **Caso 1:** Si la eliminación es exitosa, debe retornar un código 200 con el mensaje adecuado.
  - **Caso 2:** Si la lectura no se encuentra, debe retornar un error 404 con el mensaje adecuado.

5.2. Testing de Integración

El testing de integración se realiza para verificar que los diferentes módulos o componentes interactúan correctamente entre sí. Este tipo de prueba asegura que las interfaces y las dependencias entre los componentes funcionan según lo esperado, y ayuda a detectar errores relacionados con la integración y la comunicación entre las distintas partes del sistema.

5.2.1. Descargas

1. **Conexión con la Base de Datos**

- **beforeAll:** Conecta la base de datos antes de las pruebas. Si la conexión falla, se interrumpen las pruebas.
- **beforeEach:** Limpia la colección de descargas antes de cada prueba.
- **afterAll:** Limpia la base de datos y cierra la conexión después de las pruebas.

2. Operaciones CRUD

- **Caso 1:** Crear y guardar una descarga. Verificar que los datos guardados coinciden con los proporcionados.
- **Caso 2:** Obtener una descarga por usuarioId y verificar que el ISBN es correcto.
- **Caso 3:** Actualizar una descarga. Verificar que el campo formato se actualice correctamente.
- **Caso 4:** Eliminar una descarga. Verificar que el documento se haya eliminado correctamente.

3. Validación del Modelo Download

- **Caso 1:** Intentar crear una descarga sin proporcionar campos obligatorios. Verificar que se devuelvan los errores de validación.
- **Caso 2:** Intentar crear una descarga con un estado no válido. Verificar que se lance un error de validación.
- **Caso 3:** Crear una descarga sin fecha y verificar que se establezca correctamente la fecha actual.

5.2.2. Lecturas en Línea

1. Conexión a la Base de Datos

- **beforeAll:** Establece una conexión con la base de datos y lanza un error si la conexión falla.
- **beforeEach:** Elimina todos los documentos de la colección OnlineReading.
- **afterAll:** Limpia y cierra la conexión después de las pruebas.

2. Operaciones CRUD sobre OnlineReading

- **Caso 1:** Crear y guardar una lectura en línea. Verificar que los datos guardados coinciden con los proporcionados.
- **Caso 2:** Obtener una lectura por usuarioId y verificar que el ISBN es correcto.
- **Caso 3:** Actualizar una lectura en línea y verificar que el campo formato se actualice correctamente.
- **Caso 4:** Eliminar una lectura y verificar que el documento haya sido eliminado.

3. Validaciones del Modelo OnlineReading

- **Caso 1:** Crear una lectura sin proporcionar datos obligatorios. Verificar que se generen errores de validación.
- **Caso 2:** Intentar crear una lectura con un ISBN no válido y verificar que se genere un error.

- **Caso 3:** Intentar crear una lectura con un idioma no válido y verificar que se genere un error de validación.
- **Caso 4:** Crear una lectura sin especificar la fecha y verificar que se establezca correctamente la fecha actual.

5.3. Cobertura de Código

En esta sección, se detalla el análisis de cobertura de pruebas realizado en el proyecto, utilizando Jest como herramienta de pruebas. Este análisis nos permite evaluar qué porcentaje del código ha sido probado mediante las pruebas automatizadas.

Cobertura General:

- **Stmts (Sentencias):** 77.64% de las líneas de código han sido ejecutadas.
- **Branch (Condicionales):** 58.33% de las condiciones han sido probadas.
- **Funcs (Funciones):** 62.5% de las funciones han sido cubiertas.
- **Lines (Líneas):** 77.64% de las líneas de código han sido ejecutadas.

Cobertura por Archivos:

- **app.js:** 100% de cobertura.
- **db.js:** 85.71% de cobertura, con una línea sin cubrir.
- **Modelos (downloads.js, onlineReadings.js):** 85.71% de cobertura, con líneas sin cubrir.

6. Análisis de esfuerzos

El análisis de esfuerzos se ve reflejado en el reporte de Clockify: **Anexo 4: Análisis de esfuerzo (Reporte Clockify)**

7. Balance de dificultades

Durante todo el desarrollo del microservicio de descargas y lecturas, nos hemos enfrentado a varios desafíos y dificultades. Estos problemas no solo estuvieron relacionados con la falta de experiencia inicial, sino también con la complejidad añadida de gestionar dos funcionalidades separadas dentro de un único microservicio. A continuación, describimos las principales dificultades encontradas:

1. **Falta de experiencia en desarrollo de microservicios y backend.** Al ser ingenieras de la salud, nunca habíamos trabajado con tecnologías como Node.js, Express, MongoDB y la creación de APIs RESTful. Ni tampoco habíamos trabajado con entornos como github. Esto supuso un reto inicial, ya que tuvimos que aprender rápidamente sobre estas herramientas y conceptos para poder implementar el microservicio de manera efectiva.

2. **Problemas con la Instalación y Uso de Github Codespaces.** Desde el inicio del proyecto, una de las componentes de la pareja creadora del microservicio enfrentó problemas con la instalación de Docker, lo que impidió trabajar con esta herramienta de manera directa. Consultamos esta situación con nuestro profesor, quien, al no encontrar una solución viable, nos sugirió trabajar con GitHub Codespaces como alternativa.

   Sin embargo, esta solución presentó sus propios inconvenientes. El manejo de archivos .env y la ejecución de pruebas en herramientas como Postman no funcionaron de manera óptima en Codespaces, lo que complicó el flujo de desarrollo y las pruebas de funcionalidad del microservicio. Esto afectó especialmente las integraciones y validaciones entre los diferentes componentes del sistema.

3. **Duplicación de Esfuerzo**. Al tratarse de un microservicio que gestiona dos recursos, descargas y lecturas, fue necesario duplicar varias tareas y componentes. Esto incluyó:

   - **Modelos**: crear y mantener dos modelos en MongoDB para gestionar la información de descargas y lecturas, cada uno con sus respectivos campos, validaciones y relaciones con otros microservicios.
   - **Colecciones en la Base de Datos**: configurar y gestionar dos colecciones distintas en MongoDB, una para las descargas y otra para las lecturas, lo que incrementó la complejidad del manejo de datos.
   - **Pruebas**: duplicar las pruebas unitarias e integradas, asegurándonos de que cada conjunto de funcionalidades (descargas y lecturas) estuviera correctamente validado. Esto incluyó casos positivos y negativos para cada uno

de los endpoints, así como pruebas de integración con la base de datos y entre microservicios.

- **Endpoints**: diseñar y probar de manera independiente los endpoints RESTful para descargas y lecturas, asegurándonos de que siguieran buenas prácticas y que tuvieran un comportamiento consistente.

4. **Errores en los tests.** Los tests fueron otra área que nos presentó varias dificultades. A pesar de contar con un enfoque estructurado para las pruebas unitarias e integradas, nos encontramos con constantes errores y fallos en la ejecución de los tests. Tuvimos que pasar bastante tiempo depurando y revisando las pruebas, ya que algunas no se ejecutaban como esperábamos o fallaban debido a la configuración de las rutas y la interacción con la base de datos.