FISBook

LEE, DESCUBRE Y DESCARGA

MICROSERVICIO DE USUARIOS

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 amplias 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 al **Customer Agreement de Fisbook en el Anexo 3.Customer Agreement (CA).**

1.1 Características Microservicio Básico:

Este microservicio ha sido diseñado para gestionar los usuarios de 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-users'**.

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

- **API RESTful con métodos GET, POST, PUT y DELETE (/routes)**. El microservicio ofrece una API REST que permite realizar operaciones sobre los usuarios, como consulta (GET), registrar (POST), actualizar (PUT) y eliminar (DELETE) perfiles de usuario.
- **Mecanismo de Autenticación (/authentication)**: Implementamos la autenticación mediante JWT (JSON Web Tokens), asegurando que solo los usuarios autenticados puedan acceder a ciertos recursos.
- **Frontend (/src/feature/downloadsAndOnline)**. La lógica de autenticación y la gestión de usuarios están listas para ser consumidas por un frontend. Este está integrado con otras partes de la aplicación FISBook.

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

- **Despliegue en la Nube**. El microservicio está preparado para ser desplegado en la nube, en un entorno como Azure, con un proceso de integración continua y contenedores Docker que facilitan el despliegue.

Base URL: http://57.152.88.187

- **Base URL y Versionado**. La API está accesible a través de una URL versionada que sigue las mejores prácticas de API REST, permitiendo el fácil mantenimiento y evolución de la aplicación.

URL Versionada: http://57.152.88.187/api/v1/auth

- **Documentación de la API**. La API está documentada usando Swagger, permitiendo a los desarrolladores visualizar y probar los endpoints interactivos de forma sencilla. Asimismo, cuenta con un README y el presente documento para una documentación más detallada de todas las características de las que consta.

Swagger: http://57.152.88.187/api/v1/auth/api-docs/#/

- **Persistencia en MongoDB (db.js)**. Utilizamos MongoDB como base de datos NoSQL, lo que permite almacenar y gestionar los perfiles de usuario de manera eficiente.
- **Validación de Datos (/models).**. Los datos de los usuarios son validados antes de ser almacenados, utilizando Mongoose para asegurar la integridad de los datos.
- **Gestión de Código Fuente e Integración Continua (/.github/workflows)..** El código del proyecto está almacenado en un repositorio de GitHub siguiendo GitHub flow, y se ejecutan GitHub Actions para la integración continua, lo que garantiza que el código se pruebe y compile automáticamente en cada commit.
- **Dockerización (Dockerfile).**. El microservicio está empaquetado en un contenedor Docker, lo que facilita su despliegue y escalabilidad.
- **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 para garantizar el correcto funcionamiento de la API.
- **Pruebas de Integración con la Base de Datos (/tests/integration).** Se han implementado pruebas de integración que validan las interacciones entre la API y la base de datos MongoDB.

1.2 Características Microservicio Avanzado:

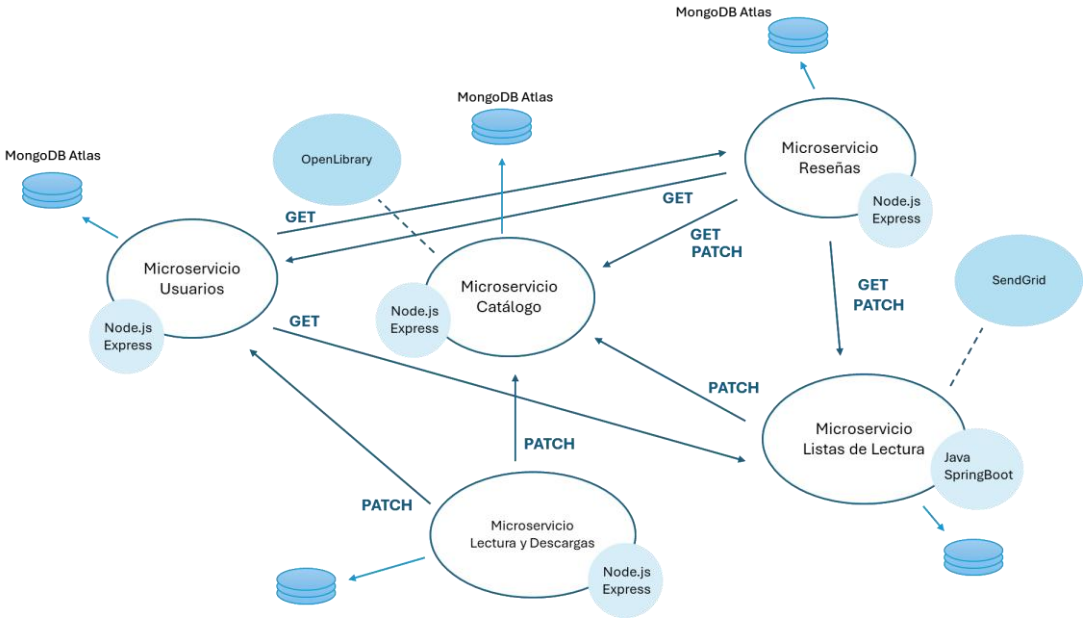
Además de cumplir con las características de un microservicio básico, hemos incorporado características adicionales que lo hacen más avanzado:

- **Autenticación JWT Avanzada**. Se ha implementado un sistema de autenticación basado en **JWT** que asegura la protección de las rutas y garantiza que solo los usuarios autenticados puedan interactuar con la API.
- **Implementación de un 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 online 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.



La arquitectura de **FISBook**, basada en microservicios, asegura la independencia y la comunicación eficiente entre las distintas funcionalidades de la aplicación. El microservicio de Usuarios juega un rol central en esta integración, proporcionando autenticación y gestión de permisos esenciales para el correcto funcionamiento de los demás microservicios.

- **Microservicio de Catálogo de Libros.** Se relaciona con el microservicio de Usuarios de la siguiente forma:
    - Los usuarios necesitan estar autenticados para poder acceder a funcionalidades avanzadas como agregar libros a sus listas de lectura, hacer reseñas, y descargar libros.
    - Las reseñas de libros están vinculadas a los usuarios, por lo que al acceder a la información de un libro, se puede consultar si el usuario ha dejado una reseña o calificación para dicho libro.
    - El microservicio de **Catálogo de Libros** no maneja la autenticación, pero se apoya en el microservicio de **Usuarios** para gestionar el acceso y las interacciones del usuario con los libros.
- **Microservicio de Listas de Lectura.** Su relación con el microservicio de Usuarios consiste en:
    - Los usuarios deben estar autenticados para gestionar su lista de lecturas.
    - El microservicio de Lista de Lecturas se asocia con el **ID de usuario** para permitir que cada usuario tenga su propia lista personalizada.

  - Los usuarios pueden agregar libros del microservicio de Catálogo de Libros a sus listas de lectura.
  - El microservicio de Lista de Lecturas no gestiona la autenticación directamente, sino que depende del microservicio de Usuarios para validar el acceso del usuario y garantizar que solo los usuarios autenticados puedan modificar su lista.

- **Microservicio de Reseñas.** Se relaciona con el microservicio de **Usuarios** de la siguiente forma:

  - Los usuarios deben estar autenticados para poder dejar reseñas y calificaciones en los libros.
  - El microservicio de Reseñas almacena las reseñas asociadas a los **ID de usuario** y **ID de libro**, lo que permite que los usuarios gestionen sus propias reseñas y las consulten posteriormente.
  - El microservicio de Reseñas no maneja la autenticación directamente, sino que se apoya en el microservicio de Usuarios para validar el acceso y asegurarse de que solo los usuarios autenticados puedan realizar estas acciones.

- **Microservicio de Descargas y Lecturas Online.** Se relaciona con el microservicio de **Usuarios** de la siguiente forma:

  - Los usuarios deben estar autenticados para acceder a la funcionalidad de descarga de libros.
  - El microservicio de Descargas no gestiona la autenticación, sino que depende del microservicio de Usuarios para validar que el usuario tenga acceso a los libros que intenta descargar.
  - Además, se pueden almacenar los registros de las descargas de los usuarios, lo que permite llevar un control de los libros descargados por cada usuario.

Cada microservicio se comunica con los demás para proporcionar una experiencia fluida e integrada. Por ejemplo, el microservicio de **Usuarios** gestiona la autenticación de los usuarios y sus permisos para acceder a funcionalidades como reseñas, listas de lectura o descargas, mientras que otros microservicios (como Catálogo de Libros o Listas de Lectura) dependen de estos roles para determinar qué recursos están disponibles para cada usuario.

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/auth/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_USERS. 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 usuarios

Registro de un nuevo usuario

- POST /api/v1/auth/register
- Descripción: Registra un nuevo usuario en la aplicación.
- Cuerpo de la solicitud:
  - username: Nombre de usuario.
  - email: Correo electrónico del usuario.
  - password: Contraseña del usuario.
- Respuesta:
  - 201 Created: Usuario creado exitosamente.
  - 400 Bad Request: Datos inválidos o faltantes.
  - 409 Conflict: El correo electrónico o nombre de usuario ya está en uso.

Inicio de sesión de un usuario

- POST /api/v1/auth/login
- Descripción: Permite al usuario iniciar sesión y obtener un JWT para autenticarse en la aplicación.
- Cuerpo de la solicitud:
  - email: Correo electrónico del usuario.
  - password: Contraseña del usuario.

- Respuesta:
  - 200 OK: Devuelve un token JWT para la autenticación.
  - 401 Unauthorized: Credenciales inválidas.

Obtener detalles del usuario

- GET /api/v1/auth/users/{userId}
- Descripción: Obtiene los detalles de un usuario específico mediante su ID.
- Parámetros:
  - userId: ID del usuario a consultar.
- Respuesta:
  - 200 OK: Detalles del usuario.
  - 404 Not Found: Usuario no encontrado.

Actualizar los detalles de un usuario

- PATCH /api/v1/auth/users/{userId}
- Descripción: Actualiza la información del usuario (como nombre, email, etc.).
- Parámetros:
  - userId: ID del usuario a actualizar.
- Cuerpo de la solicitud:
  - username: Nombre de usuario (opcional).
  - email: Correo electrónico (opcional).
  - password: Nueva contraseña (opcional).
- Respuesta:
  - 200 OK: Usuario actualizado correctamente.
  - 400 Bad Request: Datos inválidos.
  - 404 Not Found: Usuario no encontrado.

Eliminar un usuario

- DELETE /api/v1/auth/users/{userId}
- Descripción: Elimina un usuario de la base de datos.

- Parámetros:
  - userId: ID del usuario a eliminar.
- Respuesta:
  - 204 No Content: Usuario eliminado exitosamente.
  - 404 Not Found: Usuario no encontrado.

Actualizar el número de descargas de un usuario

- PATCH /api/v1/auth/users/{userId}/downloads
- Descripción: Permite actualizar el número de descargas asociadas a un usuario.
- Parámetros:
  - userId: ID del usuario.
- Cuerpo de la solicitud:
  - numDescargas: Nuevo número de descargas del usuario.
- Respuesta:
  - 200 OK: El número de descargas del usuario se ha actualizado correctamente.
  - 400 Bad Request: Error en los datos proporcionados.
  - 404 Not Found: Usuario no encontrado.

Obtener listas de lectura de un usuario

- GET /api/v1/auth/users/{userId}/readings
- Descripción: Obtiene las listas de lectura de un usuario.
- Parámetros:
  - userId: ID del usuario cuya lista de lecturas se quiere obtener.
- Respuesta:
  - 200 OK: Se obtienen las listas de lectura del usuario.
  - 404 Not Found: No se encontraron listas de lectura para el usuario.

Obtener reseñas de un usuario para libros

- GET /api/v1/auth/users/reviews/user/{userId}/book

- Descripción: Obtiene todas las reseñas realizadas por un usuario para libros específicos.
- Parámetros:
  - userId: ID del usuario cuyas reseñas se desean consultar.
- Respuesta:
  - 200 OK: Reseñas obtenidas exitosamente.
  - 404 Not Found: No se encontraron reseñas para este usuario.

Obtener estado de salud del servicio

- GET /api/v1/auth/healthz
- Descripción: Endpoint para verificar el estado de salud del microservicio de usuarios.
- Respuesta:
  - 200 OK: El microservicio está operativo.
  - 500 Internal Server Error: Error en el servidor.

3.5. Modelo de datos

El modelo de datos del microservicio de Usuarios está definido en el archivo ***users.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 Usuario

Campo	Tipo de Dato	Descripción	Ejemplo
nombre	String	Nombre del usuario.	"Juan"
apellidos	String	Apellidos del usuario.	"Pérez López"
username	String	Nombre de usuario único.	"juanperez99"
email	String	Dirección de correo electrónico del usuario.	"juan.perez@mail.com"
password	String	Contraseña del usuario.	"123456"

plan	String	Plan del usuario ('Plan1', 'Plan2', 'Plan3').	"Plan2"
rol	String	Rol del usuario ('Admin' o 'User').	"User"
listaLecturasId	Array	Lista de IDs de lecturas asociadas al usuario.	["60d3b41abd545a246389b7f4"]
numDescargas	Number	Número de descargas realizadas por el usuario.	10
resenasId	Array	Lista de IDs de reseñas asociadas al usuario.	["60d3b41abd545a246389b7f5"]

Tabla 2: Restricciones por Atributo y Validaciones Implementadas

Campo	Restricciones	Descripción
nombre	Requerido (required: true)	Longitud mínima de 2 caracteres.
apellidos	Requerido (required: true)	Longitud mínima de 2 caracteres.
username	Requerido (required: true)	Único, longitud mínima de 3 caracteres, longitud máxima de 30 caracteres, solo letras, números, puntos, guiones bajos y guiones.
email	Requerido (required: true)	Único, validación del formato del email con expresión regular.
password	Requerido (required: true)	Longitud mínima de 6 caracteres.
plan	Requerido (required: true)	Solo puede ser uno de los siguientes: 'Plan1', 'Plan2' o 'Plan3'.
rol	Requerido (required: true)	Solo puede ser uno de los siguientes: 'Admin' o 'User'.
listaLecturasId	No requerido	Almacena una lista de IDs de lecturas asociadas al usuario.
numDescargas	No requerido	Valor por defecto de 0.
resenasId	No requerido	Almacena una lista de IDs de reseñas asociadas al usuario.

3.6. 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_USERS:

- 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_USERS_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 ***Usuarios*** 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 online.
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.

1. GET /api/v1/auth/users

- **Objetivo**: Obtener la lista completa de usuarios.
- **Pruebas**:
    - Caso 1: Verifica que se devuelvan todos los usuarios con una respuesta correcta (200)
    - Caso 2: Simula un error de base de datos y asegura que la respuesta sea un error 500.

2. GET /api/v1/auth/users/:id

- **Objetivo:** Obtener los detalles de un usuario específico.
- **Pruebas:**
    - **Caso 1:** Verifica que se devuelvan los datos del usuario con el ID correcto.

- **Caso 2:** Simula que no se encuentra el usuario y verifica que la respuesta sea 404 con el mensaje adecuado.
- **Caso 3:** Simula un error de base de datos y asegura que la respuesta sea un error 500.

3. PUT /api/v1/auth/users/:id

- **Objetivo:** Actualizar los detalles de un usuario.
- **Pruebas:**
  - **Caso 1:** Verifica que la actualización de los datos del usuario se realice correctamente.
  - **Caso 2:** Simula que no se encuentra el usuario y verifica que la respuesta sea 404 con el mensaje adecuado.
  - **Caso 3:** Simula un error de base de datos y asegura que la respuesta sea un error 500.

4. DELETE /api/v1/auth/users/:id

- **Objetivo:** Eliminar un usuario específico.
- **Pruebas:**
  - **Caso 1:** Verifica que el usuario se elimine correctamente y devuelva el mensaje adecuado.
  - **Caso 2:** Simula una solicitud con un ID inválido y verifica que se devuelva el error 400.
  - **Caso 3:** Simula que no se encuentra el usuario y verifica que la respuesta sea 404 con el mensaje adecuado.
  - **Caso 4:** Simula un error de base de datos y asegura que la respuesta sea un error 500.

5. PATCH /users/:username/downloads

- **Objetivo:** Incrementar el número de descargas de un usuario específico.
- **Pruebas:**
  - **Caso 1:** Verifica que el contador de descargas se incremente correctamente.
  - **Caso 2:** Simula que no se encuentra el usuario y verifica que la respuesta sea 404 con el mensaje adecuado.
  - **Caso 3:** Simula un error de base de datos y asegura que la respuesta sea un error 500.

6. POST /users/register

- **Objetivo:** Registrar un nuevo usuario en el sistema.
- **Pruebas:**
    - **Caso 1:** Verifica que al enviar datos válidos, el usuario se registre correctamente con un mensaje de éxito.
    - **Caso 2:** Asegura que se devuelva un error 409 si el email o nombre de usuario ya existen en la base de datos.
    - **Caso 3:** Verifica que se devuelva un error 400 si ocurre un fallo de base de datos.

7. GET /users/:id/readings

- **Objetivo:** Obtener las lecturas de un usuario.
- **Pruebas:**
    - **Caso 1:** Verifica que se devuelva un error 401 si no se envía un token de autorización.
    - **Caso 2:** Verifica que las lecturas sean devueltas con éxito cuando el token es válido.
    - **Caso 3:** Verifica que se devuelva un mensaje 404 si el usuario no tiene lecturas.
    - **Caso 4:** Verifica que se devuelva un error 401 si el token es inválido.
    - **Caso 5:** Verifica que se devuelva un error 500 si ocurre un error inesperado del servidor.

8. GET /users/reviews/user/:userId/book

- **Objetivo:** Obtener las reseñas de libros de un usuario.
- **Pruebas:**
    - **Caso 1:** Verifica que las reseñas sean devueltas correctamente con éxito.
    - **Caso 2:** Verifica que se devuelva un mensaje 404 si no hay reseñas para el usuario.
    - **Caso 3:** Verifica que se devuelva un error 401 si el token es inválido.
    - **Caso 4:** Verifica que se devuelva un error 500 si ocurre un error inesperado.

9. POST /users/login

Objetivo: Iniciar sesión con las credenciales de un usuario.

- **Casos de prueba:**
    - **Debe iniciar sesión exitosamente (200):** Verifica que al enviar credenciales válidas, el usuario inicie sesión correctamente y se devuelva un token.
    - **Debe retornar 404 si el usuario no se encuentra (usuario no encontrado):** Verifica que se devuelva un error 404 si el usuario no es encontrado en la base de datos.
    - **Debe retornar 400 si ocurre un error al iniciar sesión:** Verifica que se devuelva un error 400 si hay un problema al buscar al usuario.

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.

1. Conexión a la base de datos y configuración inicial

Descripción: El test verifica si la conexión a la base de datos de usuarios se establece correctamente y se configuran las colecciones necesarias.

Casos:

- **(200)** Verifica que la conexión a la base de datos esté activa y correcta.
    - **Test**: Confirma que mongoose.connection.readyState sea igual a 1 (conectado).
- **(500)** Verifica que el nombre de la base de datos sea el esperado (en este caso, 'test').
    - **Test**: Verifica que el nombre de la base de datos sea 'test'.

2. Operaciones CRUD a través de la API

Descripción: Verifica las operaciones CRUD básicas (crear, leer, actualizar y eliminar) sobre los usuarios.

Casos:

- **(201)** Crear un nuevo usuario.
    - **Test**: Verifica que un usuario se cree y se guarde correctamente, validando su username y email.
- **(200)** Actualizar un usuario existente.

    - **Test**: Verifica que la actualización de los datos de un usuario funcione correctamente (nombre de usuario, rol, etc.).

- **(200)** Eliminar un usuario.
    - **Test**: Verifica que un usuario se elimine correctamente, asegurándose de que no se encuentre en la base de datos después de la eliminación.

3. Validaciones del Modelo de Usuario

Descripción: Verifica que las validaciones del modelo de usuario (como la duplicación de email, la longitud de la contraseña y el nombre de usuario) estén funcionando correctamente.

Casos:

- **(400)** Verificar que no se permita crear un usuario con un email ya existente.
    - **Test**: Simula un intento de crear un usuario con un email duplicado y asegúrate de que se arroje un error MongoServerError con el código 11000.
- **(400)** Verificar que la contraseña sea válida (al menos 6 caracteres).
    - **Test**: Simula un intento de crear un usuario con una contraseña demasiado corta y verifica que se lance el error esperado.
- **(400)** Verificar que el nombre de usuario tenga al menos 3 caracteres.
    - **Test**: Simula un intento de crear un usuario con un nombre de usuario demasiado corto y verifica que se lance el error esperado.

4. Limpieza de la base de datos

Descripción: Verifica que después de las pruebas la base de datos se limpie correctamente.

Casos:

- **(200)** Eliminar todos los registros de la colección de usuarios.
    - **Test**: Verifica que la base de datos se limpie después de las pruebas, eliminando los usuarios creados.

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):** 94.8% de las líneas de código han sido ejecutadas.
- **Branch (Condicionales):** 75.43% de las condiciones han sido probadas.
- **Funcs (Funciones):** 84.61% de las funciones han sido cubiertas.
- **Lines (Líneas):** 94.73% de las líneas de código han sido ejecutadas.

Cobertura por Archivos:

- **app.js:** 100% de cobertura.
- **db.js:** 33.33% de cobertura.
- **Modelos (user.js):** 100% de cobertura.
- **Rutas (users.**js): 96.63% de cobertura.

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 nos hemos enfrentado a varios desafíos y dificultades. A continuación, describimos algunas de las principales dificultades que encontramos a lo largo del proceso:

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, enfrentamos problemas con la instalación de Docker, lo que nos 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. **Importancia crítica del microservicio Users.** El microservicio de *Users* era fundamental para el correcto funcionamiento de toda la aplicación, ya que gestiona la autenticación y los perfiles de los usuarios, lo cual es esencial para la interacción

con los demás microservicios (catálogo, listas de lectura, reseñas y descargas). Debido a esta dependencia, teníamos la responsabilidad de implementarlo lo antes posible para que el resto de nuestros compañeros pudieran continuar con sus tareas sin retrasos.

4. **Desafíos con la autenticación.** La implementación de la autenticación mediante JWT (JSON Web Token) fue uno de los aspectos más complejos de este microservicio. A pesar de su documentación y ejemplos disponibles, tuvimos dificultades para integrar correctamente la validación y verificación de los tokens en las rutas de la API, lo que nos llevó a varios intentos fallidos y a la necesidad de investigar más profundamente sobre cómo manejar de forma segura los tokens en el servidor.

5. **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.