Documentación del Microservicio de Reseñas

Enlace a los repositorios donde se encuentra el código fuente y documentación adicional:
https://github.com/FIS-Book/frontend-fisbook

https://github.com/FIS-Book/fisbookBE-reviews

Nivel de Acabado

Nivel de finalización: **Hasta 7 puntos**.

Descripción de la Aplicación

Nuestra aplicación se llama FISBook y se trata de una librería online en la que se dispone de un catálogo, con sus diferentes libros y listas de lecturas. Esta librería permite hacer reseñas de los libros y además leerlos y descargarlos.

El microservicio **Reseñas** es una parte integral de la aplicación de librería virtual FISbook. Este microservicio permite a los usuarios:

- Escribir reseñas para los libros presentes en el catálogo.
- Modificar o eliminar sus propias reseñas.
- Visualizar reseñas realizadas por otros usuarios sobre un libro específico.
- Escribir, editar y modificar sus propias reseñas sobre listas de lecturas y ver las escritas por otros usuarios.
- Además, como administrador te permite tener un listado de todas las reviews realizadas y además tener control sobre ellas, es decir, se pueden eliminar y editar como si fueran tus propias reseñas.

Propósito del microservicio:

- Ofrecer una plataforma modular y autónoma para la gestión de reseñas.
- Garantizar una fácil integración con otros microservicios, como el Catálogo de Libros, el microservicio de Usuarios o el de Listas de lecturas.

Descomposición en Microservicios

La aplicación está dividida en los siguientes microservicios:

1. **Gestión de Usuarios**
    - Registro y autenticación de usuarios.
    - Administración de los usuarios
2. **Catálogo de Libros**

- Disposición de un catálogo de libros con toda la información sobre ellos.

3. **Reseñas (Microservicio implementado por esta pareja)**
    - Creación, modificación, eliminación y visualización de reseñas de libros y listas de lecturas.
    - Administración de reseñas
4. **Listas de lectura**
    - Gestión de listas de lecturas creadas por los usuarios.
5. **Lecturas y Descargas**
    - Microservicio encargado de la lectura de libros y las descargas de estos.

Acuerdo del Cliente

El acuerdo de cliente se encuentra en el repositorio, en el apartado de documentación.

Descripción de la API REST del Microservicio de Reseñas

También disponible en swagger: *http://57.152.88.187/api/v1/reviews/api-docs/*

Endpoints Principales:

1. Operaciones de Recuperación (GET):

Endpoint	Descripción	Parámetros	Respuestas
GET /	Recupera todas las reseñas de libros y listas de lectura.	Ninguno	200: { book_reviews: [...], reading_list_reviews: [...] } 500: Error interno del servidor.
GET /books	Recupera todas las reseñas de libros.	Ninguno	200: Lista de reseñas. 404: No se encontraron reseñas. 500: Error interno del servidor.
GET /reading_lists	Recupera todas las reseñas de listas de lectura.	Ninguno	200: Lista de reseñas. 404: No se encontraron reseñas. 500: Error interno del servidor.
GET /books/bk/:bookID	Recupera todas las reseñas de un libro específico.	:bookID - ID del libro	200: Lista de reseñas. 404: No se encontraron reseñas. 500: Error interno del servidor.
GET /reading_lists/rl/:readingListID	Recupera todas las reseñas de una lista específica.	:readingListID - ID de la lista de lectura	200: Lista de reseñas. 404: No se encontraron reseñas. 500: Error interno del servidor.

Endpoint	Descripción	Parámetros	Respuestas
GET /books/rev/:reviewID	Recupera una reseña específica de un libro por su ID.	:reviewID - ID de la reseña	200: Detalles de la reseña. 404: Reseña no encontrada. 500: Error interno del servidor.
GET /reading_lists/rev/:reviewID	Recupera una reseña específica de una lista de lectura por su ID.	:reviewID - ID de la reseña	200: Detalles de la reseña. 404: Reseña no encontrada. 500: Error interno del servidor.
GET /users/:userID/bk	Obtiene todas las reseñas de libros realizadas por un usuario específico	{userID}: ID del usuario	200: Detalles de las reseñas. 404: Reseñas no encontradas. 500: Error interno del servidor
GET /users/:userID/rl	Obtiene todas las reseñas de listas de lectura realizadas por un usuario específico	{userID}: ID del usuario	200: Detalles de las reseñas. 404: Reseñas no encontradas. 500: Error interno del servidor

---

2. Operaciones de Creación (POST)

Endpoint	Descripción	Cuerpo	Respuestas
POST /books	Crea una nueva reseña para un libro.	{ user_id, book_id, score, title, comment }	201: Reseña creada. 400: Error de validación. 500: Error interno del servidor.
POST /reading_lists	Crea una nueva reseña para una lista de lectura.	{ user_id, reading_list_id, score, comment }	201: Reseña creada. 400: Error de validación. 500: Error interno del servidor.

---

3. Operaciones de Actualización (PUT)

Endpoint	Descripción	Parámetros	Cuerpo	Respuestas
PUT /books/:reviewID	Actualiza una reseña de un libro por su ID.	:reviewID - ID de la reseña	{ score, title, comment }	201: Reseña actualizada. 404: Reseña no encontrada. 400: Error de validación. 500: Error interno del servidor.
PUT /reading_lists/:reviewID	Actualiza una reseña de una lista de lectura por su ID.	:reviewID - ID de la reseña	{ score, comment }	201: Reseña actualizada. 404: Reseña no encontrada. 400: Error de validación. 500: Error interno del servidor.

4. Operaciones de Eliminación (DELETE)

Endpoint	Descripción	Parámetros	Respuestas
DELETE /books/:reviewID	Elimina una reseña de un libro por su ID.	:reviewID - ID de la reseña	200: Reseña eliminada. 404: Reseña no encontrada. 500: Error interno del servidor.
DELETE /reading_lists/:reviewID	Elimina una reseña de una lista de lectura por su ID.	:reviewID - ID de la reseña	200: Reseña eliminada. 404: Reseña no encontrada. 500: Error interno del servidor.

Justificación de los Requisitos

REQUISITOS BÁSICOS DEL MICROSERVICIO

El backend debe ser una API REST tal como se ha visto en clase implementando al menos los métodos GET, POST, PUT y DELETE y devolviendo un conjunto de códigos de estado adecuado

Implementación: Archivos app.js, routes/reviews.js (y db.js para la conexión con la BBDD)

Para la persecución de este requisito al principio, nuestro microservicio consistía en métodos estático (con una variable que tenía las reseñas) ya que no se disponía de BD. A continuación, se desarrolló la conexión con la base de datos y el modelo de datos y se adecuaron los métodos para que se realizara la actualización de la base de datos.

Finalmente, para conseguir consistencia y comunicación con los otros microservicios, se implementaron los métodos correspondientes. Por ejemplo, para actualizar el valor medio de reseñas (también llamado puntuación) y el número de reseñas (de un libro o una lista de lectura), realizando la petición patch a los otros microservicios y teniendo en cuenta que, si la petición patch fallaba, se debían deshacer las transacciones anteriores.

La API debe tener un mecanismo de autenticación.

Implementación: Archivos authentication/authenticateAndAuthorize.js para la implementación de JWT , y routes/reviews.js para la aplicación de authentication.

Para la persecución de este requisito, se ha desarrollado un método authenticateAndAuthorize que se encarga de verificar si las peticiones REST reciben un token en la cabecera, comprobar y validar el token y permitir o denegar el acceso a las peticiones. A continuación, se ha utilizado este método en todos los endpoints que los necesitaban, indicando el rol que podía acceder a cada recurso.

Debe tener un frontend que permita hacer todas las operaciones de la API (este frontend puede ser individual o estar integrado con el resto de frontends).

Implementación: Archivos del repositorio del frontend : archivos de estilo(css), components/reviews/* , feature/reviews/*, parcialmente el archivo de feature/catalogue/BookDetails, hooks/useReviewsAdminHooks.js y useREviewsHooks.js

Para la persecución de este requisito, se han ido desarrollando los componentes y hooks necesarios para mostrar las reseñas, editarlas, borrarlas y crearlas. En este caso se han creado dos métodos, uno para las reviews de libros y otro para las reviews de listas de lecturas ya que tienen diferentes características y endpoints que complicaban la intregación en un sólo método. Además, se han creado los métodos necesarios para también tratar con las acciones administrativas como tener acceso sobre las reviews de los usuarios y un listado de reviews.

Debe estar desplegado y ser accesible desde la nube.

Implementación: Dockerfile

Para este requisito, en un inicio se hicieron despliegues manuales de las imagenes generadas por docker para poder realizar la integración de los microservicios. Posteriormente se usó kubernetes y se hizo una gestión de pods para poder ir desplegando sin “cortes” o downtime.

Hay que decir que este requisito ha sido ejecutado no solo por este grupo, quienes nos encargamos de los primeros despliegues manuales, si no que ha sido el conjunto de esfuerzos de los miembros del grupo de trabajo los que han ayudado a conseguir este resultado.

La API que gestione el recurso también debe ser accesible en una dirección bien versionada.

Se ha hecho uso de los conocimientos impartidos en la asignatura y de los consejos proporcionados para mantener los recursos accesibles bajo una dirección bien versionada.

Se debe tener una documentación de todas las operaciones de la API incluyendo las posibles peticiones y las respuestas recibidas.

Implementación: archivo app.js, reviews.js, swagger.js, swagger-output.json

Para este requisito se ha hecho uso de la librería de swagger autogen y se ha actualizado el código para conseguir un swagger que implemente autenticación, y todos los métodos disponibles en nuestro microservicio. Disponible en: http://57.152.88.187/api/v1/reviews/API-DOCS/

Debe tener persistencia utilizando MongoDB u otra base de datos no SQL.

Implementación: db.js, models/* , reviews.js

Para la persecución de este requisito, se ha utilizado mongoose, definiendo los dos modelos de recursos y utilizando estos modelos para asegurar la persistencia de los datos en la base de datos de MongoDB (haciendo uso de MongoDB Atlas).

Deben validarse los datos antes de almacenarlos en la base de datos (por ejemplo, haciendo uso de mongoose)

Al igual que el requisito anterior, se han validado los datos antes de ser almacenados haciendo uso de mongoose, estableciendo unos modelos con los requisitos de los recursos.

Se debe utilizar gestión del código fuente y mecanismos de integración continua

Enlace a Github: https://github.com/FIS-Book/frontend-fisbook

https://github.com/FIS-Book/fisbookBE-reviews

Para conseguir este objetivo, se definió nuestro workflow y como ibamos a usar Github desde el inicio de los repositorios. Cada miembro dispone de su rama y realiza los cambios ahí, indicando en los commits si se trataba de una nueva feature con (feat), de un fix, de un refactor. Por ejemplo: feat – Added connection with DB.

Cada rama haría merge con develop y abriría una pull request que debería ser revisada y aceptada por el compañero.

Por otra parte, para conseguir compilar y probar el código de manera automática haciendo uso de GithubActions, se definió el workflow de test.yml y sería necesario que los test pasaran para poder aceptar una pull request.

Debe haber definida una imagen Docker del proyecto

Implementación: Dockerfile

Como ya se ha comentado, se desarrolló un dockerfile simple para poder desarrollar y desplegar los cambios de una manera más eficiente.

Debe haber pruebas de integración con la base de datos

Implementación: Archivo tests/integration/db-integration.test.js para probar la interacción con la base de datos, en particular todas las principales operaciones CRUD.

Para ello, hemos utilizado una base de datos de prueba para facilitar los diferentes controles y mantener todo más limpio. En la prueba se incluye la verificación de la adición y creación de una nueva reseña, la cancelación y la edición.

Debe haber pruebas de componente implementadas en Javascript para el código del backend utilizando Jest (el usado en los ejercicios) o similar.

Implementación: archivo tests/api.test.js para realizar la prueba de componentes para cada endpoint de la API de nuestro microservicio.

Hemos utilizado Jest para simular el comportamiento de diversas funciones que requerían la comunicación con otros microservicios y para simular la autenticación. Se probaron los principales escenarios, tanto positivos como negativos, para obtener una buena cobertura del código.

REQUISITOS AVANZADOS DEL MICROSERVICIO

Implementar un frontend con rutas y navegación.

Para la implementación de este requisito, se ha utilizado la navegación con react y la creación de componentes para mostrar los recursos o navegar por la aplicación.

Implementar pruebas en la interfaz de usuario

Este requisito ha sido conseguido con pruebas end-to-end, haciendo uso de cypress y probando diferentes casos de uso del frontend y concretamente de la parte de reviews.

Estas pruebas se encuentran en el archivo tests/e2e/* . Se tratan de 3 archivos que se encargan de probar las funcionalidades bases de las 3 vistas principales: Reseñas de un libro, reseñas de una lista de lectura y listado total de reseñas.

Además, para la configuración de cypress se ha utilizado el archivo cypress.config.js y cypress/support/e2e.js.

Es importante destacar que estos test se han probado y superado en local ya que debido al despliegue realizado no se puede navegar utilizando las urls y esto no permite realizar los test. Para suplir este problema, se ha realizado de forma manual también estas comprobaciones en el frontend desplegado, consiguiendo un resultado esperado.

Implementar un mecanismo de autenticación basado en JWT o equivalente.

Implementación: Archivos authentication/authenticateAndAuthorize.js para la implementación de JWT, y routes/reviews.js para la aplicación de authentication.

Como hemos indicado en el apartado básico de autenticación, se ha desarrollado un sistema de autenticación basado en JWT. Realizando las comprobaciones de token correcto o incorrecto y haciendo uso de los roles para permitir y denegar el acceso a los diferentes recursos.

Videos demostración:

Video como usuario

Video como administrador

Análisis de los Esfuerzos

Este análisis está detallado en el documento de Clockify que se muestra en el repositorio en el apartado de documentación.

Nombre	**Actividades realizadas**
Donato Lepore	Desarrollo de endpoints REST, conexión con la base de datos de Mongo Atlas, validación de datos, pruebas de componentes de la

Nombre	Actividades realizadas
Noelia López Durán	API, pruebas de integración con la base de datos, inicio del frontend, documentación. Desarrollo inicial de endpoints REST, desarrollo de la conexión con el resto de los microservicios con validación de datos, despliegue de los microservicios iniciales en azure, desarrollo del dockerfile, desarrollo del frontend, testing del frontend, documentación, desarrollo de swagger, de la autenticación y gestión del cors.