

FUNDAMENTOS DE INGENIERÍA DEL SOFTWARE PARA CLOUD



JULIA GARCÍA FLORES

CRISTIAN ANTONIO CABELLO ARANGO

1. Introducción

En el transcurso del desarrollo de nuestro proyecto de aplicación web basada en microservicios, hemos explorado e intentado aplicar los principios clave de ingeniería del software explicados en clase, algo muy novedoso y desconocido para los participantes del grupo que conforman este microservicio. Este documento presenta un análisis detallado de la descomposición en microservicios, sus funcionalidades específicas y cómo cada componente contribuye a la integridad y eficiencia del sistema, concretamente desde el punto de vista del microservicio “Books”. A través de esta documentación, buscamos compartir nuestras decisiones fundamentales y representar los conocimientos adquiridos y aplicados al sistema durante este fascinante desarrollo de software.

Los componentes del equipo de desarrollo de este microservicio “Books” son Julia García Flores y Cristian Antonio Cabello Arango.

2. Nivel de acabado

En este apartado, se comenta el nivel al que se presentan ambos integrantes del grupo que desarrolló este microservicio “Books”.

Concretamente, nos presentamos al nivel 7 de acabado para nuestro proyecto software, en el cual hemos logrado cumplir con todos los requisitos especificados para el nivel 5 y hemos llevado a cabo implementaciones adicionales que elevan la calidad y complejidad de nuestra aplicación. A continuación, detallamos los logros alcanzados como grupo:

- API RESTful con Métodos CRUD y Autenticación: Implementamos una API REST con una gran cantidad de métodos GET, POST, PUT y DELETE, asegurando un manejo adecuado de códigos de estado, así como de las rutas.
- Incorporamos un sólido mecanismo de autenticación para garantizar la seguridad de nuestra aplicación.
- Frontend Integral: Desarrollamos un frontend que facilita todas las operaciones de la API integrado en conjunto con el frontend de los compañeros desarrolladores del resto de microservicios del sistema.
- API Bien Versionada: La API que gestiona el recurso está accesible en una dirección bien versionada, facilitando la gestión de versiones futuras (‘api/v1/books/’).
- Documentación Completa: Creamos una documentación exhaustiva que detalla todas las operaciones de la API, incluyendo las posibles peticiones y respuestas recibidas a través de la herramienta swagger.

2023/2024

Julia García Flores; Cristian Antonio Cabello Arango

- Persistencia con MongoDB: Implementamos la persistencia utilizando MongoDB y mongoose, cumpliendo con los requisitos de una base de datos no SQL.
- Validación de Datos: Garantizamos la validación de datos antes de almacenarlos en la base de datos, utilizando mongoose para asegurar la integridad de la información.
- Gestión del Código Fuente e Integración Continua: Se ha ido realizando el Control de Versiones a lo largo de todo el desarrollo del microservicio desde un repositorio de GitHub al cual se puede acceder desde el siguiente enlace:
<https://github.com/Noctua-sapientia/books-service>
- Imagen Docker: Definimos una imagen Docker del proyecto, simplificando su despliegue y asegurando consistencia en diferentes entornos.
- Pruebas de Componentes y de Integración: Implementamos pruebas de componente en JavaScript utilizando la herramienta Jest, abarcando todas las funciones tanto de la API (tests de la API), como de la base de datos (tests de integración) para garantizar su correcto funcionamiento en conjunto con la aplicación.

Al presentarnos al nivel 7 como grupo, no solo cumplimos con todos los requisitos del nivel 5, sino que también llevamos a cabo la implementación de una aplicación basada en microservicios básica. Extendimos el documento del microservicio con un análisis detallado de los límites de capacidad del customer agreement, asumiendo que esto no afecta al precio. Además, incorporamos al menos tres características avanzadas de microservicios para enriquecer la arquitectura de nuestra aplicación. A continuación, se muestran las características avanzadas de microservicios implementadas en nuestro sistema:

- Implementación de un frontend con rutas y navegación entre distintas pantallas conectadas tanto al propio microservicio como a otros microservicios pertenecientes a la organización.
- Pruebas en interfaz de usuario realizadas a nivel de aplicación en conjunto con todo el personal de la organización NoctuaSapientia.
- Consumo de API externa a través del backend: se ha desarrollado el consumo de API externas a través del backend tanto a con las APIs de los microservicios del resto de integrantes de la organización, como con una API de un servicio externo de envío de correos (MailGun). El objetivo de desarrollo y comunicación con esta API externa consiste en poder enviar un correo a un vendedor cuando este añada un libro en la tienda para notificar y confirmar subida y agradecer la confianza puesta en nuestra organización.
- Implementación de mecanismo de autenticación basado en JWT para poder mantener la sesión y poder navegar por la aplicación.

A nivel de aplicación avanzada, es de suma importancia comentar que, tanto la implementación del mecanismo de autenticación, como las pruebas de integración automatizadas, como el frontend han sido desarrollados de forma homogénea, integrados de forma común a todos los compañeros desarrolladores del resto de microservicios.

Este nivel de acabado refleja nuestro compromiso y habilidades como equipo en el desarrollo de un proyecto software completo y avanzado. Comentar que ha faltado poco para poder cumplir los requisitos del nivel 9 ya que se ha extendido el documento del microservicio con un análisis de límites de capacidad del customer agreement (que se verá más adelante) en base al análisis de la capacidad, así como tener el API REST documentado con swagger e implementado al menos tres de las características de la aplicación basada en microservicios avanzada (ya comentado) y cinco de las características de microservicio avanzado implementados.

3. Descripción de la aplicación

La aplicación desarrollada en este proyecto, llamada “Noctua Sapienta”, presenta una web de venta de libros, en la que se aborda desde el control de los libros en venta por parte de un vendedor hasta la gestión de los pedidos de los usuarios compradores de un libro. En la aplicación se diferencian dos tipos de usuarios, compradores y vendedores, a continuación, se detallan las principales funcionalidades que puede llevar a cabo cada uno de ellos en la aplicación y que se corresponden con sus funcionalidades principales.

3.1. Usuario Comprador.

- Registrarse como nuevo usuario en la aplicación.
- Iniciar sesión mediante la pantalla de login de la aplicación.
- Visualizar una pantalla de información personal del usuario (Nombre y dirección).
- Visualizar un listado de todos los libros en venta en la aplicación.
- Para cada libro disponible a la venta, el usuario puede ver sus detalles (Nombre, autor, categoría, valoración), ver las diferentes opciones de vendedores para el libro, ver las valoraciones que el libro tiene actualmente y añadirlo al carrito.
- El usuario puede comprar libros del carrito.
- Una vez comprado un libro, se genera un pedido, el usuario comprador puede acceder a la sección de mis pedidos de la aplicación y visualizar todos los pedidos que ha realizado.
- El comprador tiene acceso a la sección de la aplicación “mis reseñas”, en la que podrá visualizar todas las valoraciones que ha creado tanto para libros como para vendedores.
- El usuario puede cerrar sesión en la aplicación.

3.2. Usuario Vendedor.

- Registrarse como nuevo usuario en la aplicación.
- Iniciar sesión mediante la pantalla de login de la aplicación.
- Visualizar una pantalla de información personal del usuario (Nombre, número de pedidos, número de valoraciones, valoración).

- Visualizar un listado de todos los libros que el vendedor tiene en venta en la aplicación.
- Para cada libro disponible a la venta por parte del vendedor, el usuario puede ver sus detalles (ISBN, Nombre, autor, categoría, valoración, precio, stock), puede modificar estos valores o eliminar libros para que dejen de estar a la venta. Además, también podrá añadir nuevos libros a la venta.
- El vendedor tiene acceso a la sección de la aplicación “mis pedidos” en la que podrá visualizar todos los pedidos correspondientes con sus ventas, además podrá modificar el estado o cancelar estos pedidos.
- El vendedor tiene acceso a la sección de la aplicación “mis reseñas” en la que visualizará un listado de todas las valoraciones que ha recibido como vendedor.
- El usuario puede cerrar sesión en la aplicación.

4. Etapas de definición de microservicios

Tras unas series de reuniones, en las que se realizaron varios análisis exhaustivos de los requisitos del sistema completo, para comprender sus funciones y poder definir los límites de contexto para cada microservicio. En este sentido, la definición de los microservicios de nuestro centro se realizó de forma progresiva, tal y como se representa a continuación:

Definición de la temática: en una primera instancia, se realizó una puesta en común de una serie de ideas propuestas por los integrantes del grupo para escoger una temática sobre la cual realizar este proyecto. Se decidió optar a diseñar un sistema de servicio de compra y venta de libros.

Definición de requisitos: en una segunda instancia, se realizó una definición de requisitos en función de los objetivos establecidos en el plan docente.

Análisis del Sistema: una vez definidos los requisitos, se realizó un análisis exhaustivo de los requisitos del sistema completo para comprender sus funciones y componentes principales. En este instante, se identificaron que las principales áreas funcionales clave podrían ser Users, Books, Orders y Reviews.

Identificación de Límites de Contexto: en este punto, se identificaron los límites de contexto para cada microservicio. Cada microservicio tiene responsabilidades bien definidas y no depende, en gran medida, de otros microservicios para realizar sus tareas.

Interfaz de Microservicios: el sistema debe ser adaptado a un enfoque de API REST para facilitar la interacción, con versionado claro (api/v1/), uso de formato JSON y métodos HTTP estándar (GET, POST, PUT, DELETE).

Manejo de Estados y Errores: el sistema debe poder mostrar los estados posibles de cada microservicio y cómo manejar errores. Los códigos de estado adecuados pueden ser 200, 204, 404, 500, etc. para garantizar una comunicación robusta y comprensible.

Documentación Detallada: el sistema debe incluir información detallada para cada microservicio, describiendo sus funciones, interfaces, estados posibles y cualquier

requisito especial. La documentación detallada de cada microservicio que compone el sistema se encuentra a continuación.

Microservicio Users: El microservicio Users se encarga de gestionar todo lo relacionado con la autenticación y el manejo de usuarios en nuestra aplicación. Permite el registro de nuevos usuarios, recopilando información esencial como nombres, direcciones de correo electrónico y contraseñas. Además, se encarga de validar la unicidad de las direcciones de correo electrónico para asegurar que cada usuario tenga una identidad única en el sistema. Este microservicio asume la responsabilidad de mantener y gestionar la base de datos de usuarios, asegurando la integridad y privacidad de la información del usuario. Además, se integra de manera eficiente con el servicio externo de envío de correos para la gestión de archivos y datos multimedia.

Microservicio Orders: El microservicio Orders se ocupa de la gestión completa de los pedidos realizados por los usuarios en nuestra aplicación. Permite la selección de libros, proporciona información de envío y almacena un historial detallado de los pedidos para cada usuario. Además, se integra de manera eficiente con el servicio externo de envío de correos para la gestión de archivos y datos multimedia.

Microservicio Reviews: El microservicio Reviews se encarga de las interacciones y opiniones de los usuarios sobre los libros. Permite a los usuarios dejar comentarios y reseñas, ofreciendo también la opción de editar y eliminar sus propios comentarios. Colabora estrechamente con el microservicio Books para obtener información actualizada sobre los libros, garantizando la sincronización y coherencia de los datos relacionados con reseñas. Su responsabilidad principal radica en el mantenimiento y gestión de la base de datos de reseñas y comentarios. Además, se integra de manera eficiente con el servicio externo de envío de correos para la gestión de archivos y datos multimedia.

Microservicio Books: El microservicio Books se dedica a la administración de información relacionada con los libros en nuestra aplicación a través de un catálogo general. Desde la adición y edición hasta la eliminación de libros, se garantiza la consistencia y disponibilidad de datos. Este microservicio almacena los valores más importantes y significativos en el atributo libros, como título, autor, género, identificador único (ISBN), así como opciones de venta para el cliente en función del vendedor, el precio y el stock. Además, se integra de manera eficiente con el servicio externo de envío de correos para notificación y confirmación de subida de libros a la aplicación, por parte del vendedor.

El microservicio Books es el microservicio que ha sido implementado por el personal desarrollador de la organización: Julia García Flores Y Cristian Antonio Cabello Arango.

5.API REST Microservicio Book

En este apartado se muestra la descripción del API REST de este microservicio “Book”. A continuación, aparece una tabla que contiene la información más importante (rutas, métodos, descripción) de la API REST de nuestro microservicio, así como un breve desarrollo de la descripción del funcionamiento de esta API:

| Especificación de API | | | |
|-----------------------|---|--|-----------|
| Tipo | Endpoint | Descripción | Realizado |
| GET | /api/v1/books | Devuelve todos los libros de la aplicación | SÍ |
| | /api/v1/books/:isbn | Devuelve la información de un libro por id. | SÍ |
| | /api/v1/books/:isbn/:seller | Devuelve un vendedor de un libro. | SÍ |
| | api/v1/books /:isbn/:seller/options?options=* Donde * puede ser stock/prize/reviews/id | Devuelve el stock, precio, reseñas o id de un libro en base a un vendedor. | SÍ |
| POST | /api/v1/books | Dar de alta un nuevo libro en el sistema. | SÍ |
| | /api/v1/books/:isbn/:seller | Dar de alta a un nuevo vendedor en un libro. | SÍ |
| PUT | /api/v1/books/:isbn | Modifica cualquiera de las características principales de un libro (título/autor, año/genero). El ISBN no se puede modificar | SÍ |
| | api/v1/books /:isbn/:seller/options?options=* Donde * puede ser stock/prize/reviews/id | Modifica el stock, precio o reseñas un libro en base a un vendedor. El id del vendedor no se puede modificar. | SÍ |
| DELETE | /api/v1/books/:id | Elimina un libro al completo por id. | SÍ |
| | /api/v1/books/:id/:seller | Elimina un libro por id. Si el libro solo tiene un vendedor, se borra todo el libro, si tiene más de un vendedor, se borra | SÍ |

| | | | |
|--|--|---|--|
| | | únicamente los datos del vendedor del libro | |
|--|--|---|--|

http://localhost:4002/api/v1/apidocs/books/#/ Books API

Books Operations related to books.

default

| | | |
|--------|--------------------------------|--|
| GET | /books | Obtiene la lista de libros. |
| POST | /books | Crea un nuevo libro. |
| GET | /books/{isbn} | Obtiene detalles de un libro por ISBN. |
| PUT | /books/{isbn} | Actualiza un libro por ISBN. |
| DELETE | /books/{isbn} | Elimina un libro por ISBN. |
| GET | /books/{isbn}/{seller} | Obtiene detalles de un libro por ISBN y vendedor. |
| POST | /books/{isbn}/{seller} | Agrega un nuevo vendedor a un libro por ISBN. |
| DELETE | /books/{isbn}/{seller} | Elimina los datos de un vendedor para un libro por ISBN. |
| GET | /books/{isbn}/{rating} | Obtiene detalles de un libro por ISBN y calificación. |
| POST | /books/{isbn}/{rating} | Agrega una nueva valoración a un libro por ISBN. |
| GET | /books/{isbn}/{seller}/options | Obtiene opciones de un libro por ISBN y vendedor. |
| PUT | /books/{isbn}/{seller}/options | Actualiza las opciones de un libro por ISBN y vendedor. |
| PUT | /books/{isbn}/{seller}/stock | Aumenta el stock en el número de unidades especificado para un libro y vendedor. |

Descripción de funcionamiento de la API:

- Un vendedor puede crear un libro (añadir el libro como una biblioteca). Se almacenará el título, el autor, el año, el género, el vendedor, el stock y el precio del libro que haya añadido el vendedor.
- Un vendedor puede cualquier campo de un libro subido por el.
- Un cliente puede modificar la valoración de un libro.
- Cuando un cliente compre un libro, se modificará el stock del libro.
- Cuando se añada una reseña de un cliente (se reciba un comando de reviews el libro), se actualizará la puntuación del libro (se suma a la puntuación que tiene y se divide entre el número de reviews que tenga) y el número de reviews del libro.
- Los vendedores no podrán añadir reviews a sus propios libros.
- Se eliminarán los libros relacionados con el vendedor cuando ellos se eliminen. Si el libro solo tiene un vendedor, se eliminará el libro, si no, se eliminará de las opciones del libro.
- Se analizarán las visitas/compras de libros cada cierto tiempo y aparecerán en la búsqueda los libros que más visitas/compras tengan (conexión de servicio externo).
- Se hará un get de los libros cuando se acceda a la pestaña “Books” de la aplicación ordenado por título de “Books”.

Eventos del microservicio

DELETE VENDEDOR: dado un vendedor de un libro, elimina el vendedor del libro teniendo en cuenta que, si es el único vendedor de un libro: se borra el libro también, sino: se borra el vendedor (y sus datos de stock y precio) de las opciones (options) del libro.

GET USER: dado el id del vendedor, obtener los valores name y email a través de una API.

Servicio externo de envío de correos: MailGun. Se acude a una función de este servicio externo, importado y solicitado

```
Nombre Cristian Antonio
Email julgarflo@alum.us.es
Correo enviado con éxito: {
  id: '<20240123190928.d47251b882e838eb@sandbox9a41b9e9b51b4356bd90b00a920b85cb.mailgun.org>',
  message: 'Queued. Thank you.'
}
POST /api/v1/books 201 647.341 ms - 7
```

6. Customer Agreement de la Aplicación

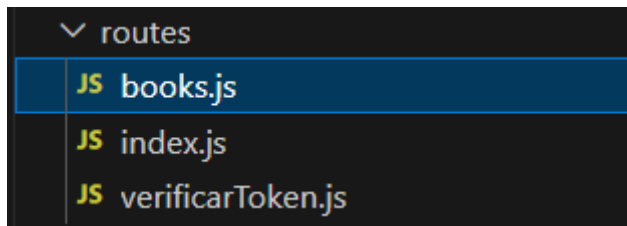
El documento asociado al Customer Agreement de la aplicación se encuentra en el repositorio de documentos de la organización, Noctua Sapientia, disponible para acceder desde el siguiente enlace: <https://github.com/Noctua-sapientia>. Acceder a la carpeta Análisis de la Capacidad, incluida en el repositorio docs. En esta carpeta se muestran las diferentes versiones de los documentos desarrollados durante la realización de este proyecto.

7. Justificaciones

A continuación, se muestran las justificaciones sobre el cumplimiento de los requisitos que forman parte de la evaluación del proyecto. Los requisitos son los que han sido mencionados y definidos en función del nivel de acabado del microservicio:

Backend como API REST

El backend de nuestro microservicio se implementa como una API REST que incluye una gran cantidad de métodos GET, POST, PUT y DELETE. Además, se asegura de devolver conjuntos de códigos de estado adecuados según las operaciones realizadas. Todo ello ha sido desarrollado en el fichero books.js incluido dentro de la carpeta routes del repositorio del proyecto:



```
router.get('/:isbn', verificarToken, async function(req, res, next) {
  const isbn = req.params.isbn;

  try {
    const foundBook = await Book.findOne({ isbn });

    if (foundBook) {
      res.status(200).send(foundBook.cleanup());
    } else {
      res.status(404).send("Libro no encontrado");
    }
  } catch (error) {
    console.error(error);
    res.sendStatus(500);
  }
});
```

```
router.post('/', verificarToken, async function(req, res, next) {
  const { isbn, author, title, year, genre, rating, options } = req.body;
  const accessToken = req.headers.authorization;

  const newBook = new Book({
    isbn,
    author,
    title,
    year,
    genre,
    rating,
    options
  });
  console.log("Opciones", options.seller);
  try {
    const userOfReview = await User.getSellersInfo(accessToken, options.seller);
    console.log("Nombre", userOfReview.name);
    console.log("Email", userOfReview.email);

    await emailSender.sendEmail(userOfReview.name, userOfReview.email, newBook.title);
    await newBook.save();
    res.sendStatus(201);
  } catch (e) {
    console.error("Error:", e);
    res.status(500).send(e);
  }
});
```

```
router.put('/:isbn', verificarToken, async function(req, res, next) {
  const isbn = req.params.isbn;

  try {
    const foundBook = await Book.findOne({ isbn });

    if (!foundBook) {
      return res.status(404).send("Libro no encontrado");
    }

    if (foundBook.isbn !== isbn) {
      return res.status(400).send("ISBN incorrecto, debe indicar correctamente el ISBN del libro que quiere modificar (No es posible)");
    }

    foundBook.title = req.body.title;
    foundBook.author = req.body.author;
    foundBook.year = req.body.year;
    foundBook.genre = req.body.genre;
    foundBook.rating = req.body.rating;
    foundBook.options = req.body.options;

    await foundBook.save();

    res.status(200).send("Libro actualizado exitosamente");
  } catch (error) {
    console.error(error);
    res.sendStatus(500);
  }
});
```

```
router.delete('/:isbn/:seller', verificarToken, async function(req, res, next) {
  const isbn = req.params.isbn;
  const sellerId = parseInt(req.params.seller);
  const accessToken = req.headers.authorization;

  try {
    const foundBook = await Book.findOne({ isbn });
    if (!foundBook) {
      return res.status(404).send("Libro no encontrado");
    }
    if (!foundBook.options) {
      return res.status(404).send("Opciones no disponibles");
    }
    const findOptionIndex = foundBook.options.findIndex(option => option.seller === sellerId);
    if (findOptionIndex === -1) {
      return res.status(404).send("Vendedor no encontrado para este libro");
    }
    const numVendedores = foundBook.options.length;
    if (numVendedores > 1) {
      foundBook.options.splice(findOptionIndex, 1);
      await foundBook.save();
      Order.cancelDeletedBookOrders(accessToken, isbn, sellerId);
    } else {
      await Book.findOneAndDelete({ isbn });
      Order.cancelDeletedBookOrders(accessToken, isbn, sellerId);
    }

    res.status(200).send("Datos del vendedor borrados exitosamente");
  } catch (error) {
    console.error(error);
    res.sendStatus(500);
  }
});
```

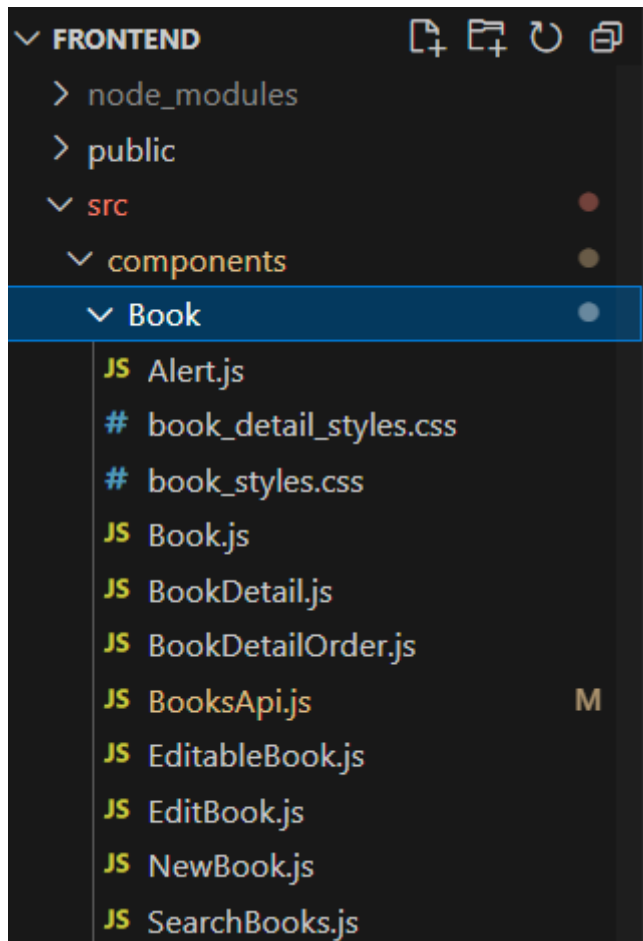
Mecanismo de Autenticación de la API

La API cuenta con un sólido mecanismo de autenticación para garantizar la seguridad de las operaciones. Se implementa un método que verifica la identidad de los usuarios antes de permitir el acceso a ciertos recursos a través del fichero: `verificarToken.js`

```
routes > JS verificarToken.js > ...
...
1  const jwt = require('jsonwebtoken');
2  const SECRET_KEY = 'a56d1f7c0c817387a072692731ea60df7c3a6c19d82ddac228a9a4461f8c5a72';
3
4  function verificarToken(req, res, next) {
5    const token = req.header('Authorization');
6
7    if (!token) {
8      return res.status(401).json({ error: 'Acceso denegado. Token no proporcionado.' });
9    }
10
11    jwt.verify(token, SECRET_KEY, (err, decoded) => {
12      if (err) {
13        return res.status(403).json({ error: 'Acceso denegado. Token inválido.' });
14      }
15
16      req.user = decoded;
17      next();
18    });
19  }
20
21  module.exports = verificarToken;
```

Frontend Integral

Se ha desarrollado un frontend integral que permite realizar todas las operaciones de la API. Este frontend puede encontrarse integrado en un monorepositorio con el resto de frontend de la aplicación, concretamente en la carpeta `components/Books`.



API Bien Versionada

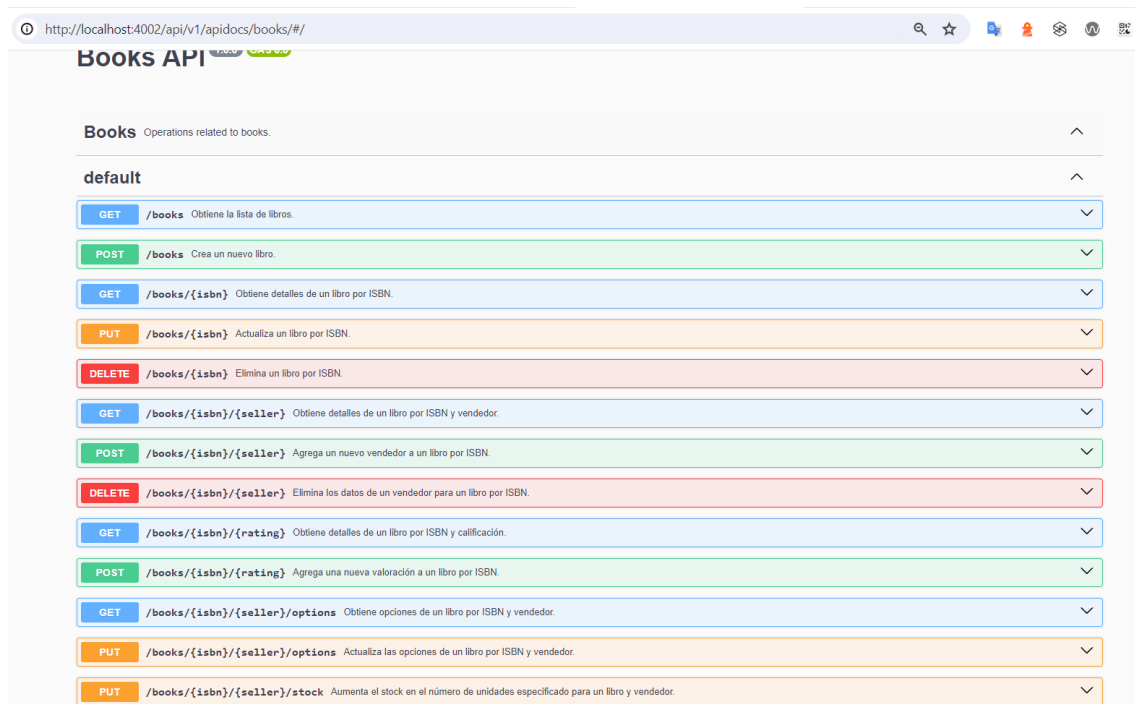
La API que gestiona el recurso se encuentra accesible en una dirección bien versionada ('api/v1/books'), facilitando la gestión de versiones futuras.

```
app.use('/', indexRouter);  
app.use('/api/v1/books', booksRouter);
```

A su vez, se puede observar en el presente documento todas las direcciones (endpoints) definidas para este microservicio en el apartado 5 (API REST Microservicio Books).

Documentación Completa de la API

La documentación exhaustiva de todas las operaciones de la API ha sido incluida en el fichero books.js disponible desde el repositorio de este microservicio, incluyendo las posibles peticiones y respuestas recibidas, utilizando la herramienta Swagger. También se puede acceder a esta documentación lanzando el microservicio y accediendo a la siguiente dirección: <http://localhost:4002/api/v1/apidocs/books/#/>



Persistencia con MongoDB

Se implementa la persistencia utilizando MongoDB y mongoose, cumpliendo con los requisitos de una base de datos no SQL. El esquema de la base de datos definido para este microservicio se compone de los siguientes atributos:

```
{
  isbn: { type: String, unique: true, required: true },
  title: { type: String, required: true },
  author: { type: String, required: true },
  year: { type: Number, required: true },
  genre: { type: String, required: true },
  rating: { type: Number, required: false },
  options: [{
    seller: { type: Number, required: true },
    stock: { type: Number, required: true },
    prize: { type: Number, required: true }
  ]
}
```

Este esquema se encuentra dentro la carpeta models en el fichero book.js. LA configuración apropiada se puede observar en los archivos de configuración del microservicio.

Validación de Datos

Antes de almacenar datos en la base de datos, se garantiza su validación utilizando mongoose para asegurar la integridad de la información.

```

10  "dependencies": {
11    "axios": "^1.6.5",
12    "cookie-parser": "~1.4.4",
13    "cors": "^2.8.5",
14    "debug": "~2.6.9",
15    "express": "^4.18.2",
16    "jsonwebtoken": "^9.0.2",
17    "mailgun": "^0.5.0",
18    "mailgun-js": "^0.22.0",
19    "mongoose": "^8.0.4",
20    "morgan": "~1.9.1",
21    "swagger-jsdoc": "^6.2.8",
22    "swagger-ui-express": "^5.0.0",
23    "url-join": "^4.0.1"

```

```

models > JS book.js > ...
...
1  const mongoose = require('mongoose');
2
3  const bookSchema = new mongoose.Schema(
4    {
5      isbn: { type: String, unique: true, required: true },
6      title: { type: String, required: true },
7      author: { type: String, required: true },
8      year: { type: Number, required: true },
9      genre: { type: String, required: true },
10     rating: { type: Number, required: false },
11     options: [{
12       seller: { type: Number, required: true },
13       stock: { type: Number, required: true },
14       prize: { type: Number, required: true } }]
15   }
16 );
17
18
19 bookSchema.methods.cleanup = function() {
20   return {
21     isbn: this.isbn,
22     title: this.title,
23     author: this.author,
24     year: this.year,
25     genre: this.genre,
26     rating: this.rating,
27     options: this.options.map(option => ({
28       seller: option.seller,
29       stock: option.stock,
30       prize: option.prize
31     })))
32   }
33 }
34 const Book = mongoose.model('Book', bookSchema);
35
36 module.exports = Book;
37

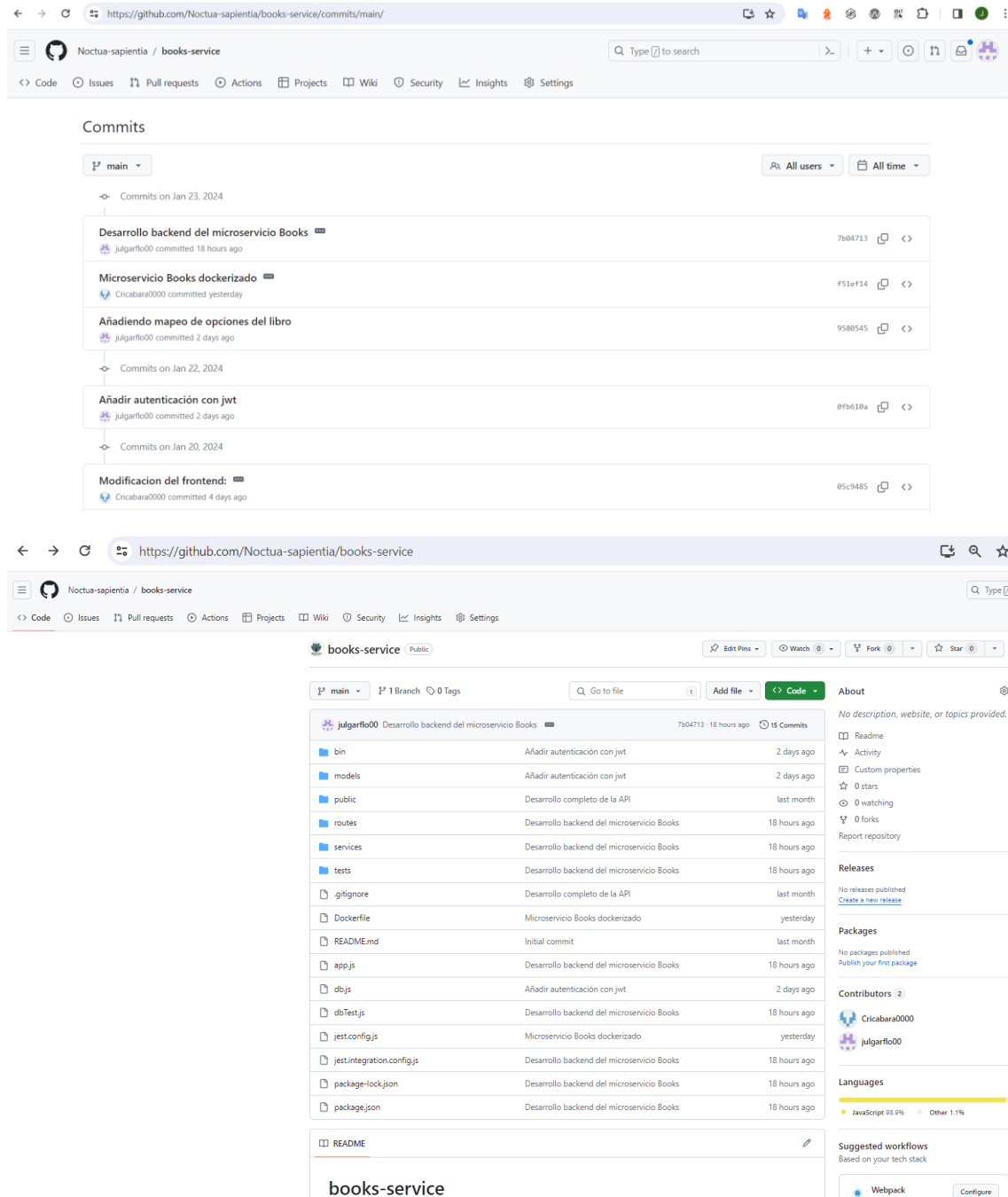
```

Este esquema se encuentra dentro la carpeta models en el fichero book.js. LA configuración apropiada se puede observar en los archivos de configuración del microservicio.

Gestión del Código Fuente e Integración Continua

El código se gestiona a través de un repositorio de GitHub siguiendo el flujo de trabajo GitHub flow. Además, se implementan mecanismos de integración continua mediante GitHub Actions para compilar y probar automáticamente el código en cada commit. Se

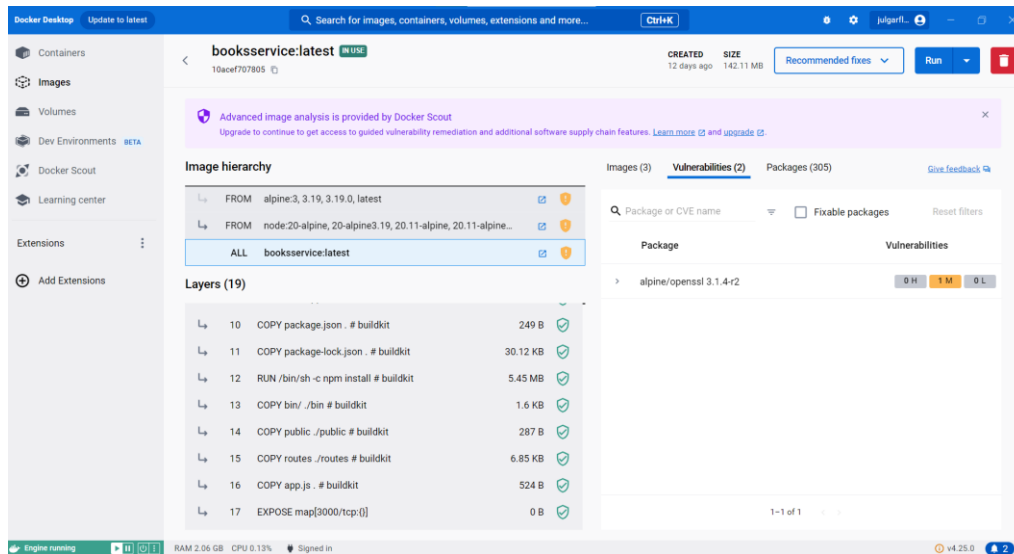
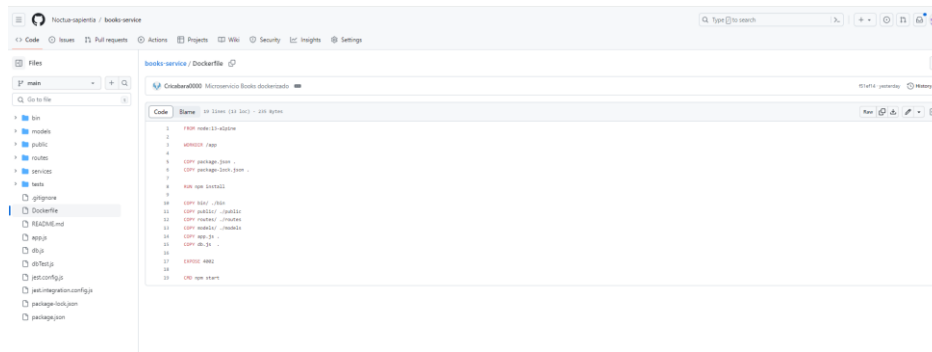
puede acceder al repositorio a través del siguiente enlace: <https://github.com/Noctua-sapientia/books-service>



The screenshot displays the GitHub repository page for `Noctua-sapientia/books-service`. The top section, titled 'Commits', shows a list of recent commits on the `main` branch. The bottom section shows the repository's file structure, including folders like `bin`, `models`, `public`, `routes`, `services`, `tests`, and files like `.gitignore`, `Dockerfile`, `README.md`, `app.js`, `db.js`, `dbTest.js`, `jest.config.js`, `jest.integration.config.js`, `package-lock.json`, and `package.json`.

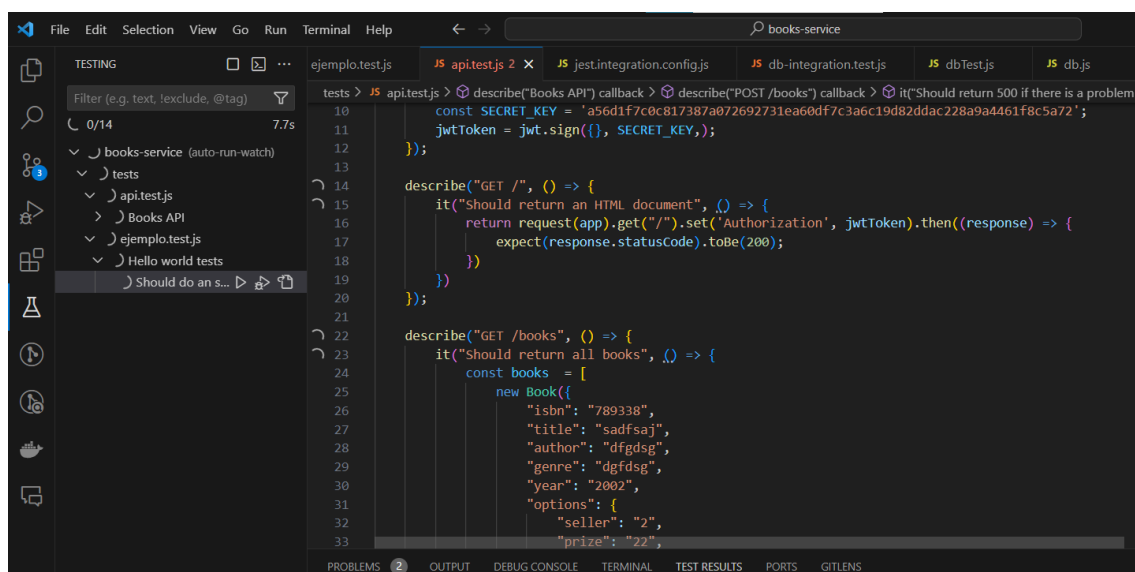
Imagen Docker del Proyecto

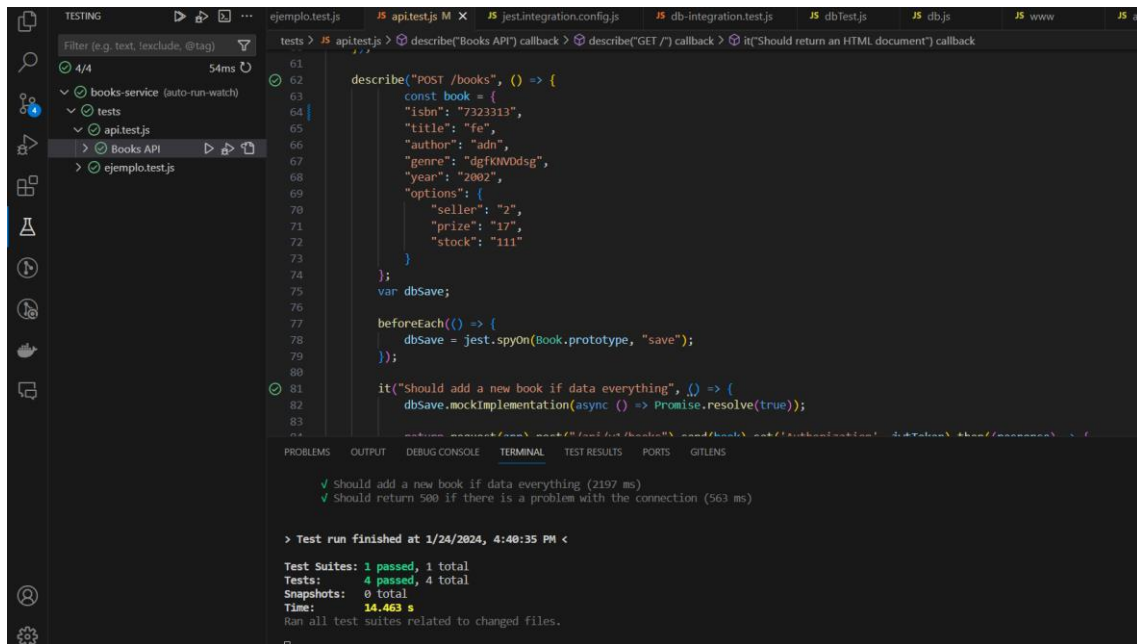
Se ha definido una imagen Docker del proyecto accesible desde el repositorio del microservicio indicado en el punto anterior:



Pruebas de Componente en Javascript

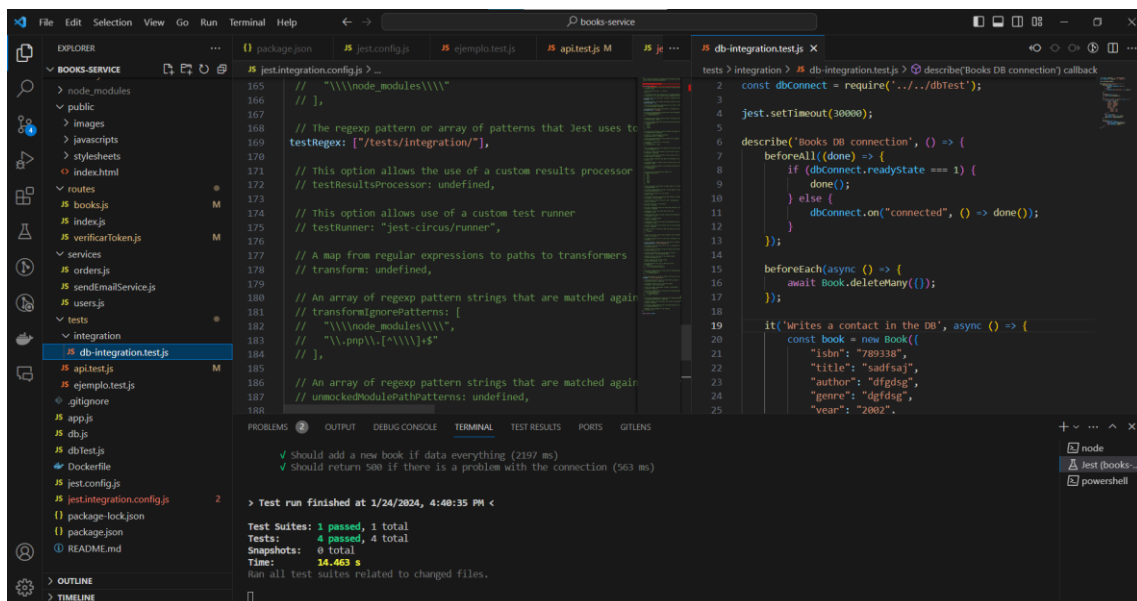
Se han implementado pruebas de componente en JavaScript utilizando Jest para el código del backend, cubriendo funciones del API no triviales de la aplicación, siguiendo los pasos comentados en los vídeos de la asignatura. Se realizan pruebas tanto para escenarios positivos como negativos.





Pruebas de Integración con la Base de Datos

Se han implementado pruebas de integración exhaustivas con la base de datos para garantizar su correcto funcionamiento en conjunto con la aplicación, siguiendo los estándares establecidos en la asignatura.

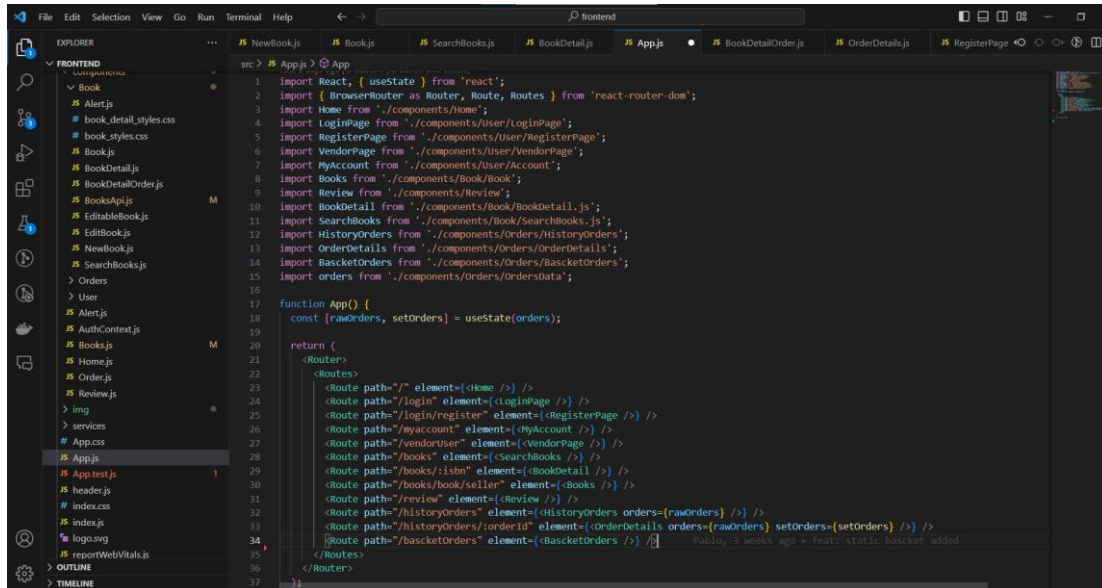


Todos estos requisitos forman parte del desarrollo del microservicio básico Books.

A continuación, se muestran las características que forman parte del desarrollo del microservicio avanzado Books con la justificación de la consecución de los mismos:

Implementación de Frontend con Rutas y Navegación

Se ha desarrollado un frontend común a toda la organización que incluye las rutas y navegación, integrado con el resto de los microservicios para proporcionar una experiencia de usuario coherente. El historial de rutas puede ser observado a través del fichero App.js disponible en el repositorio de frontend monorepo común a todos los microservicios que dan lugar a la organización: <https://github.com/Noctua-sapientia/frontend>



```

src > App.js > App
1 import React, { useState } from 'react';
2 import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
3 import Home from './components/Home';
4 import LoginPage from './components/User/LoginPage';
5 import RegisterPage from './components/User/RegisterPage';
6 import VendorPage from './components/User/VendorPage';
7 import MyAccount from './components/User/Account';
8 import Books from './components/Book/Book';
9 import Review from './components/Review';
10 import BookDetail from './components/Book/BookDetail.js';
11 import SearchBooks from './components/Orders/SearchBooks.js';
12 import HistoryOrders from './components/Orders/HistoryOrders';
13 import OrderDetails from './components/Orders/OrderDetails';
14 import BasketOrders from './components/Orders/BasketOrders';
15 import orders from './components/Orders/OrdersData';
16
17 function App() {
18   const [rawOrders, setOrders] = useState(orders);
19
20   return (
21     <Router>
22       <Routes>
23         <Route path="/" element=<Home /> />
24         <Route path="/login" element=<LoginPage /> />
25         <Route path="/login/register" element=<RegisterPage /> />
26         <Route path="/myaccount" element=<MyAccount /> />
27         <Route path="/vendoruser" element=<VendorPage /> />
28         <Route path="/books" element=<SearchBooks /> />
29         <Route path="/books/:id" element=<BookDetail /> />
30         <Route path="/books/book/seller" element=<Books /> />
31         <Route path="/review" element=<Review /> />
32         <Route path="/historyOrders" element=<HistoryOrders orders={rawOrders} /> />
33         <Route path="/historyOrders/orderId" element=<OrderDetails orders={rawOrders} setOrders={setOrders} /> />
34         <Route path="/basketOrders" element=<BasketOrders /> />
35       </Routes>
36     </Router>
37   );

```

Implementación de Pruebas en la Interfaz de Usuario

Las pruebas de interfaz de usuario han sido realizadas a nivel de aplicación. El acceso al documento que contiene la información sobre estas pruebas se encuentra en el repositorio docs de la organización NoctuaSapientia: <https://github.com/Noctua-sapientia/docs>

Consumo de API Externa a través del Backend

Se ha desarrollado el consumo de API externas a través de la comunicación entre nuestro microservicio y un servicio externo de envío de correos (MailGun). Este sistema permite notificar a un usuario de tipo vendedor, a través de un correo electrónico, cuando este añada un libro en la aplicación. El fichero sendEmailService.js se encuentra dentro de la carpeta services del repositorio del microservicio "Books".

```
services > JS sendEmailService.js > [0] sendEmail > [0] data > text
You, yesterday | 1 author (You) | Click here to ask Blackbox to help you code faster
1 const mailgun = require('mailgun-js');
2
3 const mg = mailgun({
4   apiKey: '43783e39653315761a0be9d79edee7ad-063062da-72a57ef4',
5   domain: 'sandboxe4768fd1b324ff6ad99ebeb156ad0df.mailgun.org'
6 });
7
8 const sendEmail = async function(name, email, title) {
9   console.log(name);
10  console.log(email);
11  console.log(title);
12  const data = {
13    from: 'BOOKS NEWSLETTER NOCTUA SAPIENTIA <booksnewsletter@noctuasapientia.com>',
14    to: email,
15    subject: 'Nuevo Libro en Noctua Sapientia',
16    text: `¡Hola ${name}! ¡Estamos encantados de que leas este correo! Te escribimos para confirmarte que el libro: ${title} ha
17    sido subido a la plataforma. Disfruta de tu experiencia en Noctua Sapientia.`
18  };
19
20  try {
21    const result = await mg.messages().send(data);
22    console.log('Correo enviado con éxito:', result);
23  } catch (error) {
24    console.error('Error al enviar el correo:', error);
25  }
26 };
27
28 module.exports = {
29   sendEmail: sendEmail
30 };
31
```

Cabe resaltar que en este microservicio se realiza comunicación con el resto de microservicios de la organización y, que la carpeta services del repositorio del microservicio “Books” se alojan todas las comunicaciones con el resto de microservicios de la organización.

Implementación de Mecanismo de Autenticación basado en JWT

Se menciona la implementación de un mecanismo de autenticación basado en JWT, cumpliendo con el requisito mencionado, en conjunto con todos los integrantes de la organización. El fichero de acceso a este mecanismo se encuentra en el fichero verificarToken.js disponible en el repositorio de este microservicio: [Noctua-sapientia/books-service \(github.com\)](https://github.com/Noctua-sapientia/books-service)

```
routes > JS verificarToken.js > ...
You, last week | 1 author (You) | Click here to ask Blackbox to help you code faster
1 const jwt = require('jsonwebtoken');
2 const SECRET_KEY = 'a56d1f7c0c817387a072692731ea60df7c3a6c19d82ddac228a9a4461f8c5a72';
3
4 Comment Code
5 function verificarToken(req, res, next) {
6   const token = req.header('Authorization');
7
8   if (!token) {
9     return res.status(401).json({ error: 'Acceso denegado. Token no proporcionado.' });
10  }
11
12  jwt.verify(token, SECRET_KEY, (err, decoded) => {
13    if (err) {
14      return res.status(403).json({ error: 'Acceso denegado. Token inválido.' });
15    }
16
17    req.user = decoded;
18    next();
19  });
20
21  module.exports = verificarToken;
```

Todos estos requisitos forman parte del desarrollo del microservicio avanzado Books para alcanzar el nivel 7 de acabado.

A continuación, se muestra un requisito cumplido que forma parte del desarrollo del nivel 9 de acabado, así como la justificación de la consecución del mismo:

API REST documentada con swagger

La documentación exhaustiva de todas las operaciones de la API ha sido incluida en el fichero books.js disponible desde el repositorio de este microservicio, incluyendo las posibles peticiones y respuestas recibidas, utilizando la herramienta Swagger. También se puede acceder a esta documentación lanzando el microservicio y accediendo a la siguiente dirección: <http://localhost:4002/api/v1/apidocs/books/#/>

```
routes > JS books.js > ...
64 * @swagger
65 * /books/{isbn}:
66 * get:
67 *   summary: Obtiene detalles de un libro por ISBN.
68 *   description: Requiere token de autenticación.
69 *   parameters:
70 *     - in: path
71 *       name: isbn
72 *       description: ISBN del libro.
73 *       required: true
74 *       schema:
75 *         type: string
76 *   responses:
77 *     200:
78 *       description: Detalles del libro obtenidos correctamente.
79 *     404:
80 *       description: Libro no encontrado.
81 *     500:
82 *       description: Error en el servidor.
83 */
84 /*GET books listing*/
85 router.get('/:isbn', verificarToken, async function(req, res, next) {
86   const isbn = req.params.isbn;
87
88   try {
89     const foundBook = await Book.findOne({ isbn });
90
91     if (foundBook) {
92       res.status(200).send(foundBook.cleanup());
93     } else {
94       res.status(404).send("Libro no encontrado");
95     }
96   } catch (error) {
97     console.error(error);
98     res.sendStatus(500);
99   }
100 });
```

8. Análisis de esfuerzos

En este apartado, se muestra un análisis esquematizado de la distribución de tareas y el esfuerzo realizado de los dos componentes del equipo desarrollador de este microservicio "Books". Cada una de las filas de la tabla contiene un resumen de las horas que se han dedicado a cada tarea de implementación de microservicios básicos.

| Acción | Julia | Cristian |
|--|----------|----------|
| Planificación y Diseño (en conjunto con la organización) | 8 horas | 8 horas |
| Implementación de API RESTful | 30 horas | 10 horas |
| Desarrollo del Frontend Integral | 7 horas | 21 horas |
| Persistencia con MongoDB, Validación de Datos y Docker | 2 horas | 2 horas |

| | | |
|---|-----------------|-----------------|
| Pruebas de Componentes y de Integración | 5 horas | 5 horas |
| Consumo de APIs y Servicios Externos a través del Backend | 10 horas | 5 horas |
| Total Backend | 47 horas | 22 horas |
| Total Frontend | 7 horas | 21 horas |

9. Vídeo de demostración

El vídeo de la demostración del microservicio “Books”, integrado en la organización Noctua Sapientia, se encuentra en el repositorio de documentos de dicha organización, disponible para acceder desde el siguiente enlace: <https://github.com/Noctua-sapientia/books-service>

10. Presentación

El documento asociado a la Presentación de la Aplicación se encuentra en el repositorio de documentos de la organización, Noctua Sapientia, disponible para acceder desde el siguiente enlace: <https://github.com/Noctua-sapientia>. Acceder al repositorio docs.

11. Aplicación basada en microservicios implementada

El repositorio asociado a la aplicación basada en microservicios implementada se encuentra en GitHub disponible para acceder desde el siguiente enlace: <https://github.com/Noctua-sapientia>. Nuestro microservicio se encuentra disponible desde el repositorio books-service, incluido dentro de la organización Noctua Sapientia.

12. Análisis de la capacidad

El documento asociado al Análisis de la Capacidad se encuentra en el repositorio de documentos de la organización, Noctua Sapientia, disponible para acceder desde el siguiente enlace: <https://github.com/Noctua-sapientia>. Acceder a la carpeta Análisis de la Capacidad, incluida en el repositorio docs. En esta carpeta se muestran las diferentes versiones de los documentos desarrollados durante la realización de este proyecto.

13. Service Level Agreement (SLA)

El documento asociado al SLA se encuentra en el repositorio de documentos de la organización, Noctua Sapientia, disponible para acceder desde el siguiente enlace: <https://github.com/Noctua-sapientia>. Acceder a la carpeta SLA, incluida en el repositorio docs. En esta carpeta se muestran las diferentes versiones de los documentos desarrollados durante la realización de este proyecto.

14. Pruebas de Aceptación (UAT)

El documento asociado al SLA se encuentra en el repositorio de documentos de la organización, Noctua Sapientia, disponible para acceder desde el siguiente enlace: <https://github.com/Noctua-sapientia>. Acceder a la carpeta SLA, incluida en el repositorio docs. En esta carpeta se muestran las diferentes versiones de los documentos desarrollados durante la realización de este proyecto.

15. Agradecimientos

Como partes del equipo de la organización Noctua Sapientia y desarrolladores del microservicio “Books” agradecemos la colaboración y participación de todos los integrantes del equipo a lo largo del desarrollo de este proyecto.