**MICROSERVICIO READING**

**Nivel de acabado al que nos presentamos es de 7 a 9**

|  |  |
| --- | --- |
| **MICROSERVICIO BÁSICO QUE GESTIONE UN RECURSO:** |  |
| 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 adecuado1. | X |
| La API debe tener un mecanismo de autenticación. | X |
| 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) | X |
| Debe estar desplegado y ser accesible desde la nube | X |
| La API que gestione el recurso también debe ser accesible en una dirección bien versionada | X |
| Se debe tener una documentación de todas las operaciones de la API incluyendo las posibles peticiones y  las respuestas recibidas | X |
| Debe tener persistencia utilizando MongoDB u otra base de datos no SQL | X |
| Deben validarse los datos antes de almacenarlos en la base de datos (por ejemplo, haciendo uso de mongoose) **Usamos MongoRepository** |  |
| Se debe utilizar gestión del código fuente y mecanismos de integración continua:  o El código debe estar subido a un repositorio de Github siguiendo Github flow  o El código debe compilarse y probarse automáticamente usando GitHub Actions u otro sistema de integración continua en cada commit | X |
| Debe haber definida una imagen Docker del proyecto | X |
| Debe haber pruebas de componente implementadas en Javascript para el código del backend utilizando Jest (el usado en los ejercicios) o similar. **Usamos JUnit y Mockito**  Como norma general debe haber tests para todas las funciones  del API no triviales de la aplicación. Probando tanto escenarios positivos como negativos | X |
| Debe haber pruebas de integración con la base de datos | X |
| **MICROSERVICIO AVANZADO QUE GESTIONE UN RECURSO:** |  |
| Implementar un frontend con rutas y navegación. | X |
| Consumir alguna API externa (distinta de las de los grupos de práctica) a través del backend o algún otro tipo de almacenamiento de datos en cloud como Amazon S3. **Nos integramos con SENDGRID para envío de emails** | X |
| Implementar un mecanismo de autenticación basado en JWT o equivalente. | X |
| **APLICACIÓN BASADA EN MICROSERVICIOS BÁSICA:** |  |
| Interacción completa entre todos los microservicios de la aplicación integrando información. La integración debe realizarse a través del backend. | X |
| **APLICACIÓN BASADA EN MICROSERVICIOS AVANZADA:** |  |
| Tener un frontend común que integre los frontends de cada uno de los microservicios. Cada pareja debe ocuparse, al menos, de la parte específica de su microservicio en el frontend común. | X |

**Descripción de la aplicación**

FISBOOK es una plataforma o aplicaicón web en línea diseñada para facilitar la gestión personal de libros y listas de lectura.

A través de la aplicación, los usuarios registrados pueden:

* Consultar una amplia variedad de libros disponibles en la base de datos de la plataforma.
* Crear, gestionar y eliminar sus propias listas de lectura personalizadas.
* Añadir y eliminar libros de sus listas de lectura.
* Dar reseñas y puntuaciones a libros y listas de lectura de otros usuarios.
* Descargar libros disponibles en la plataforma.

**Descomposición de los microservicios**

1. **Users:** http://57.152.88.187/api/v1/auth/api-docs/

Microservicio encargardo de la gestión de usuarios: creación y autenticación.

1. **Catalogo:** <http://57.152.88.187/api/v1/books/api-docs/>

Microservicio encargado de la gestión de libros, proporciona información de libros disponibles en la aplicación.

1. **Downloads:** [http://57.152.88.187/api/v1/read-and-download/api-docs/](http://57.152.88.187/api/v1/books/api-docs/)

Microservicio encargado de la descarga de libros.

1. **Reviews:** <http://57.152.88.187/api/v1/reviews/doc/>

Microservicio encargado de la gestión de reseñas y valoraciones de libros y listas de lecturas.

1. **Readings:** <http://57.152.88.187/api/v1/readings/swagger-ui/index.html>#

Microservicio encargado de la gestión de listas de lecturas. Crear listas, eliminar listas, añadir libros, sacar libros de listas.

Este microservicio es el implementado por **Desarrolladores/Estudiantes:**

* Edwin Patricio Arévalo Angulo edwareang@alum.us.es
* Mathis Goujon [matgou@alum.us.es](mailto:matgou@alum.us.es)

**Descripción del API REST del microservicio.**

El microservicio readings o listas de lecturas es un **microservicio REST** diseñado para gestionar una lista organizada de lecturas preferidas por usuarios.

El microservicio está compuesto por un total de 11 endpoints cuya documentación y descripción está desplegada bajo swagger en las siguientes rutas:

* UI: http://57.152.88.187/api/v1/readings/swagger-ui/index.html
* JSON: <http://57.152.88.187/api/v1/readings/api-docs>

También la descripción de la API REST está disponible en el fichero readme del proyecto fisbookBE-reading

**Justificación de cómo se ha ido consiguiendo:**

Al tratarse de un trabajo colaborativo y según las habilidades de cada desarrollador en algunos casos hemos aportado más y en otros menos. Dicho lo anterior hemos ido construyendo e implementando el microservicio de la siguiente manera.

1. Reuniónes con equipo FISBOOK para definir aplicación, casos de uso y asignación de microservicios.
2. Elección de tecnología.

**Backend**

1. Diseño arquitectónico.
2. Alta, configuración y conexión a Base de datos.
3. Integración con docker.
4. Creación de repositorio.
5. Desarrollo de la primera API REST de prueba integrando toda la infraestructura.
6. Integración con Swagger.
7. Análisis y primera documentación de las APIs en readme.
8. Alta e integración con API externa SendGrid.
9. Empezamos con los tests unitarios.
10. Empezamos con las pruebas de integración en GITHUB.
11. Reuniones con equipo de trabajo Fisbook para ver estado del trabajo.
12. Implementación y desarrollo de casos de uso del microservicio.
13. Pruebas unitarias y de integración sobre todo lo desarrollado.
14. Alta en AZURE, generación de imégen y primer despliegue del microservicio en AZURE
15. Añadimos seguridad a las APIs con JWT.
16. Pruebas de consumo de APIs con POSTMAN.
17. Integración con microservicio externo de catálogo.
18. Configuración de variables, generación de imagen y despliegue en Kubernetes.

**Frontend**

1. Reunión con equipo FISBOOK
2. Diseño e implementación de primeras vistas para mostrar las listas de lecturas
3. Configuración de Cors en el backend
4. Añadir botónes e implementar funciones para llamar a las APIs
5. Añadir estilo a las vistas
6. Navegación entre vistas
7. Configuración de variables, generación de imagen y despliegue en Kubernetes.
8. Reunión con equipo FISBOOK
9. Pruebas de interacción entre el front, back y demás microservicios.

**Evidencias**

Los recursos que corroboran lo anterior tanto en tiempo como en trabajo se encuentran disponibles en las siguientes rutas:

**Código fuente en GITHUB:**

* Backend: <https://github.com/FIS-Book/fisbookBE-reading>
* Frontend: https://github.com/FIS-Book/frontend-fisbook

**Temporalización - Clockify**

<https://app.clockify.me/tracker>

**A continuación, y muy relevante para completar este documento, se añade la siguiente información:**

**Backend**

**Tecnología**

* Java 17
* Spring Boot
* MongoAtlasDB
* Maven
* Docker
* JUnit y Mockito
* GitHub

**Arquitectura**

La arquitectura usada en el desarrollo ha sido “The Clean Architecture” en la que hemos separado con éxito la lógica de negocio de la infraestructura.

Para conseguir esto hemos implementado las capas core:

* **Entities**: ReadingEntity (Genre, Book)
* **User Cases:** ReadingService, SendGridEmailService
* **Controllers:** EmailController, HealthzController, ReadingController
* **Infraestructure:** ReadingRepository

**Módulos complementarios**

Para completar el microservicio hemos tenido que implementar los siguientes módulos:

**api:** Aglutina los DTO para hacer la transferencia de información desde la capa de Copntrollers a las capas internas

**config:** En este módulo se crean, implementan las clases encargadas de configurar tanto la seguridad jwt (SecurityConfig), redireccionamientos a los docs (SwaggerConfig), habilitación de cors (WebConfig)

**exception:** Clases diseñadas para controlar las excepciones con mayor precisión

* BadRequestException
* ConflictoException
* GlobalExceptionHandler
* PreconditionException
* ResourceNotFoundException

**jwt:** en este módulo está la clase JWTAuthenticationFilter encargada de filtrar qué endpoints que no necesitan autenticación y los que sí.

**template:** en este módulo se implmenta la clase encargada de la comunicación con otros microservicios.

**utils:** módulo con clases de utilidad para hacer validaciones de campos o crear constantes

**Recursos de configuración**

1. .env: Para el funcionamiento y configuración de entorno Docker
2. application.properties: Para el funcionamiento y configuración interna de la aplicación
3. integration-test: para el correcto lanzamiento de pruebas de integración

**EndPoints**

Tal como se destalla al principio de este documento todos los endpoints están disponibles y configurados para funcionar de manera adecuada según las reglas de negocio de la aplicación.

También cabe destacar que existen dos endpoints que no necesitan de autenticación, y para evitar el uso indebido de dos de ellos se necesita una clave que será proporcionada por el equipo de desarrollo de este microservicio cuando la soliciten vía email (ver emails de desarrolladores).

**Sin Autenticación:**

POST: api/v1/readings/email (necesita clave)

GET: api/v1/readings/healthz

**Con autenticación:** Todos los demás

**Test**

Para verificar el correcto funcionamiento de la aplicación y de todos los módulos en conjunto se han realizado un total de 77pruebas

**Unitarios:**

Paquete api: 32 test que incluyen escenarios positivos y negativos.

Paquete entity: 28 test que incluyen escenarios positivos y negativos.

**Integración:**

Paquete controller:13 test que incluyen escenarios positivos y negativos.

Paquete repository: 4 test que incluyen

**Integración continua**

Se realiza mediate la integración con GITHUB configurando el respectivo Action en el fichero integration-test.yml. Para el lanzamiento correcto de las pruebas de integración se tiene que tener en cuenta que se deben configurar 4 secretos en el repositorio de GIT.

**Frontend**

**Tecnología Front**

* axios
* bootstrap
* cors
* jwt-decode
* react
* web-vitals

Se generan e implementan tres vistas

**Todas las listas de lecturas:** vista en la que se muestran todas las listas de lectura del sistema. Desde esta vista se puede navegar a las listas de lectura del usuario autenticado

**Mis listas de lecturas:** vista en la se muestran solamente las listas de lectura del usuario autenticado. En esta vista se ofrecen las siguientes funcionalidades:

* Crear nuevas listas de lecturas
* Eliminar listas de lecturas
* Añadir libros a una lista de lectura
* Eliminar libros de una lista de lectura
* Alta de lista de lectura
* Navegar hacia todas las lecturas del sistema

**Crear nuevas listas de lecturas:** vista que ofrece un formulario para crear una nueva lista de lectura según el género que desee el usuario o previamente establecido por el sistema.

Previamente establecido por el sistema quiere decir que debe existir una lista de genero igual al género del libro que se quiera añadir. Por defecto el género se considera primer parte de la categoría a la que pertenece el libro.

**Nivel de acabado al que nos presentamos es de 7 a 9**