Prueba técnica para Spring Boot

Francisco José Díaz Robles

Versiones:

- Java 21 LTS
- Spring Boot 3.3.0
- Maven 4.0.0
- IDE: STS

Arquitectura:

Se ha realizado la prueba bajo una arquitectura basada en capas MVC.

He tomado esa decisión porque me ha parecido demasiado sencillo como para utilizar arquitectura hexagonal.

Desarrollo:

- Sistema de perfiles: usándose solo el perfil de dev
- Base de datos utilizada H2 en memoria RAM, es decir cuando reiniciamos la aplicación se reinicia la estructura y los datos.
- Usamos ficheros yamls en lugar de ficheros properties.
- Para la estructura de la Base de datos (DDL) usaremos la propia funcionalidad de JPA+Hibernate para la creación de tablas a partir de clases JAVA. Los datos (DML) los introducimos en un fichero data.sql que será ejecutado por Spring al arranque del proyecto después de la creación de tablas.
- Las capas utilizadas en el proyecto son:
    - Repository: La capa de repositorio o los DAOs. La capa que permite el acceso a la BD. Utiliza los Beans denominados Entity para correlacionarlos con las tablas de la BD.
    - Service: Capa con las clases donde se introducirán la lógica de negocio y las validaciones. Utiliza los Beans dentro del paquete ValueObject para realizar con ellos las operaciones de negocio.
    - API: Aquí se encuentran los controladores JAVA para las APIs. Utiliza objetos DTO que recibirá a través de la API y también los enviará.
    -
- Mapa mapear los objetos usado entre capas utilizaremos MapStruct

- La documentación de la API esta expuesta a través de la herramienta Swagger de OpenAPI. Incluye:
    - Nombre de controladores
    - Descripción endpoints
    - Descripción de parámetros (required, type, etc)
    - Descripción de objetos DTOs que necesitan las APIs pero también de los objetos que devuelve.
    - Mensajes de error posibles devueltos por los endpoints (200,404, etc.)
    - Ejemplos peticiones en cada uno de los endpoints.
    - Posibles excepciones que pudiera devolver cada endpoint.
- Validaciones en las APIs de los parametros a traves de las validaciones @Bean de JAVA
- Manejo global de las excepciones con un controlador de Spring.
- Incluye registro de Log con de log4j.
- Se utiliza una cache en la ejecución del método Service.findById. Usamos la cache propia de Spring. En este caso caffeine. Se puede configurar o deshabilitar en el fichero de propiedades.
- Test unitarios con Spring Boot y Mockito de la clase Service.
- Test de integración con Spring Boot.
- Respuestas de las API con ResponseEntity.
- Control de versiones: GIT
- Incluye un consumidor y un productor de Kafka que puede ser habilitado listener en el fichero de propiedades desde la propiedad auto-startup: false a true

No existen comentarios ya que he escrito este fichero de documentación y es solo una prueba técnica.

La prueba técnica ha sido larga. No me ha parecido pertinente dockerizarlo.