Prueba Técnica

Ejecutar los microservicios

1. Debe colocarse en la ruta del proyecto Technical Test
2. Abrir una consola en la ubicación
3. Ejecutar uno de estos comandos mvn install o mvn spring-boot:run
4. Esperar a que finalice.

   En caso que desee ejecutarlos manualmente debe realizarlo en el mismo orden que se encuentran los módulos en el pom.xml

Tecnologías y herramientas

- Intellij idea
- Maven
- H2 database
- Spring boot, spring cloud, spring security, spring gateway
- Postman
- Navegador Vivaldi

Funcionalidades de los microservicios

WasteManagerAddressService

Carpeta Entity: se crea la clase entidad y Dto (Objeto de Transferencia de Datos). En la clase entidad utilizo las anotaciones de validación que luego se llamaran en la clase controladora a la hora de recibir un objeto y las propiedades de la columna en la BD H2. Con las anotaciones @Prepersist y @PreUpdate me aseguro de crear y actualizar los atributos consecuentes con el tiempo respectivamente. Dto es un record, este tipo de java es comúnmente utilizado por ser inmutables.

Carpeta Repository: Se crea la interface Repository que extiende de CrudRepository, la cual contiene los métodos necesarios para las consultas a la BD (CRUD). La etiqueta @Repository es opcional, pero por convención se debe introducir.

Carpeta Service: Se encuentran dos carpetas más, Contract que es donde se encuentra la interface Service, dentro de esta se especifica las funciones a realizar para esta entidad. Por otro lado, se encuentra la carpeta Implements, dentro de ella está la clase implementación que implementa la interface Service; la clase está anotada por @Service especificándole a spring que la clase es un bean de Servicio; está presente la anotación @Autowired para injectar las dependencias en el este caso el Repositorio y también @Transactional para especificar que es una transacción si solo una consulta select(readOnly) u otro tipo (anotación sin valor).

Carpeta Controller: Dentro está presente la clase controladora donde se establece todos los endpoints a través de las anotaciones @GetMapping, @PutMapping, @PostMapping y @DeleteMaping (CRUD), y @RequestMapping para la clase en general (a partir de aquí es donde empiezan las urls de las demás anotaciones). Las demás carpetas son para crear los errores y excepciones (se crea una excepción padre que implementa una interface, el objetivo mantener la estructura de las excepciones en la respuesta) que se enviarán el response en caso de que ocurran.

Clase DataBaseInitialize: clase abstracta que se llama en la clase principal de spring después que termina su ejecución. Su función es que al correr el servidor se inserten los datos de ejemplo en la BD H2.

Resumen

El objetivo de este microservicio es realizar el crud de la entidad a través de la clase controladora. Este debe ser manipulado por el microservicio WasteManagerService.

WasteManagerAddressService

En cuanto a estructura de carpetas y archivos es bastante similar al microservicio anterior, ahora hablaremos de sus particularidades.

Se crea la Entidad con sus respectivas anotaciones. En ella cree una variable que no se encuentra en el documento con el objetivo de solo guardad el id de la entidad WasteManagerAddress. Este identificador es omitido en la respuesta del Json por la anotación @JsonIgnore. Más abajo tengo el objeto de la entidad WasteManagerAddress que esta anotado con @Transient para cuando spring realice la persistencia se omita para que no quede almacenado en la BD, lo que se quiere es enviar este objeto en la respuesta.

Existe una lista con la anotación @OneToMany con el tipo de cascada en All para que cualquier cambio de la clase entidad eliminación, creación, actualización y demás también la afecten al objeto de esta lista en la BD. De tipo set para que no permita duplicaciones.

Carpeta Service: Esta tiene en los contratos una interfaz genérica de la cual extienden las demás interfaces con sus respectivas entidades. Esto es para evitar la repetición de código. Se encuentra en estos contratos la interface encargada de realizar las peticiones al microservicio WasteManagerAddress, en la anotación @FeignClient su valor es el nombre del microservicio registrado en NameService el cual utiliza Eureka. En las implantaciones se crea algo similar como en los servicios, una clase genérica que implementa la interfaz genérica. Los métodos lo que hacen es heredar de la misma e implementar su interfaz correspondiente (En estos se puede crear otras implementaciones que deben estar antes su interface service)

Nota: mucha de las funciones en la implementación genérica lanza sus excepciones

Controller: A la hora de resolver una petición y necesita del objeto que proporciona el otro microservicio se utiliza la interfaz con la anotación @FeignClient que previamente fue inyectada. En los response no se envía el atributo de la entidad si no el objeto que fue rescatado del otro microservicio.

CloudConfigServer

Aquí se almacenan los archivos de configuración de los microservicios. Estos archivos deben tener el mismo nombre de la aplicación (microservicio). Además se crea un aplication.properties que es automáticamente consumido por los demás microservicios. Los microservicios que lo consumen cargan las configuraciones a partir de aquí porque se especifican la url donde se va encontrar el mismo y la dependencia ConfigClient. La anotación @EnableConfigServer es la que especifica que es un servidor de configuración. El @RefreshScope permite que a la hora de hacer

un cambio de configuración en tiempo de ejecución se pueda realizar a través de actuator con el enlace actuator/refresh. Se debe especificar el repositorio git donde se encuentran los archivos. Para ver las configuraciones el endpoint es localhost:<puerto>/<nombre del archive>/default <o perfil>.

NamingService

Este contiene la dependencia de Eureka Server el cual permite registrar los microservicios y para la comunicación con ellos solo hace falta conocer el nombre del mismo, lo demás lo maneja Eureka. Se establece esta anotación @EnableEurekaServer. Los microservicios para que se registren deben presentar esta anotación @EnableDiscoveryClient que no solo funciona con eureka.

ApiGateway

Esta se debe registrar en el servidor de Eureka, para que encuentre los microservicios y datos que necesita. En las properties se especifica el microservicio al cual va tener acceso el Gateway dándole su id, socket y el path al cual va redirigir el Gateway. Se especifica el tiempo de conexión y lectura, se incluye también lo registros. Ahora se utiliza este artefacto Gateway como dependencia porque Zuul se encuentra descontinuado.

Este además se le implementa un filtro de seguridad a través de microservicio IdentityService que implementa la autenticación con JWT. En caso de que el usuario este autenticado pasa la petición a su destino final en caso contrario es UNAUTHORIZED. Este Filtro se implementa en la carpeta Filters. Uno de sus archivos es FilterFunction que es una clase que implementa una función estática de tipo Function<ServerRequest, ServerRequest> que se va encargar de realizar la petición al microservicio IdentityService para ver si el usuario se encuentra autenticado.

Luego se encuentra la clase AuthenticationFilter que está anotada con @Configuration que implementa una función de tipo RouterFunction<ServerResponse> que se encarga de crear las rutas con métodos estáticos y se le aplica el método before para verificar antes de hacer la petición la autenticidad del usuario.

En resumen, este se encarga de dirigir las peticiones al microservicio WasteManagerService.

Nota: La lista de `wasteCenterAuthorizationEntities` genera los datos automáticamente, solo debe especificar la cantidad de objeto con llaves vacías.

Json de Muestra para realizar peticiones a la BD

Para crear un **WasteManagerEntity**

```
{
  "name": "John Doe",
  "nif": "12345678A",
  "isEnabled": true,
  "version": 1,
  "wasteManagerAddressEntity": {
    "address": "123 Main St",
    "isEnabled": true,
    "version": 1
  },
  "wasteCenterAuthorizationEntities": [{},{},{}]
}
```

IdentityService

Este microservicio se encarga de la autenticación y registro de usuarios, tiene como dependencia Spring Security. Expone dos enlaces uno para la validación del token y otro para el registro. Se implementa la entidad User con sus repositorios y servicios. Estos servicios deben implementar la interfaz UserDetailsService para el autenticado.

Carpeta Security: Es donde se maneja la lógica de seguridad y sus configuraciones. Token es una clase abstracta que expone métodos estáticos para la validación del token y creación del mismo. Para el ejemplo en la creación del token se tomo solo el nombre el cual es enviado en subject, se establece un tiempo de expiración, el momento en que fue creado, los claim (se utiliza una lista clave-valor vacía ya que no es necesario enviar contenido a través del mismo) y finalmente una clave de acceso.

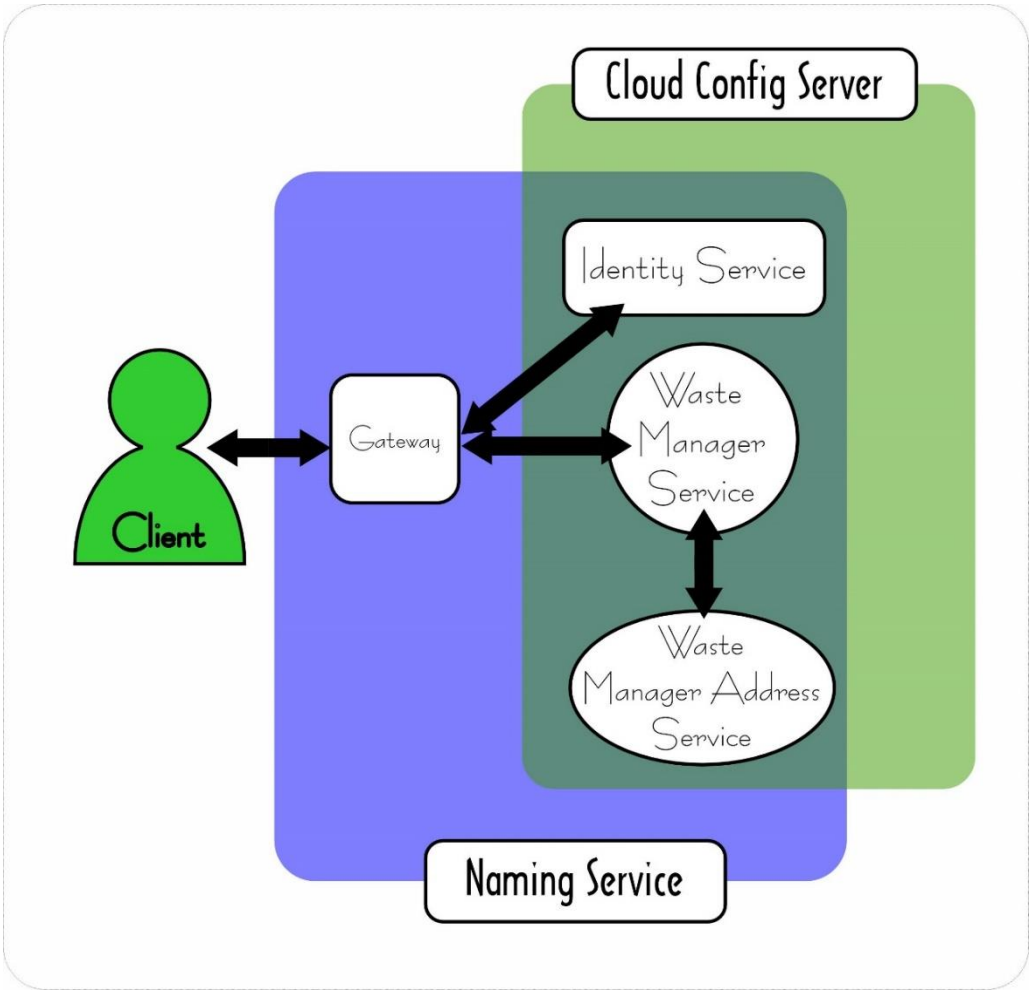
AuthenticationFilter es la clase que hereda los métodos para autenticar donde primero se obtienen los datos y en caso de que sean correcto se ejecuta el método de successfully donde se añade una cookie con el token con un tiempo de 1h.

Config es la clase que se encarga de manejar las configuraciones, en el cual se crean beans como el tipo de decodificación de contraseña que luego es agregado en el bean de manejador de autenticación y el ultimo bean es el filtro donde se establecen los métodos, header permitidos (todos) y se expone el header de la cookie. Y se agrega el filtro de autenticación.

Está decisión fue tomada para establecer una seguridad básica para la protección de los microservicios. Solamente debe estar expuesto el Gateway ya que es la puerta de enlace por tanto es quien debe verificar antes de comunicarse con los microservicios la autenticación del usuario. El resto de los microservicios se mantienen en la red privada del sistema.

Nota: Se implementaron los test a los microservicios de WasteManagerAddressService y WasteManagerService ya que el objetivo principal es la correcta inserción de los datos en estos.

Esquema de los microservicios



Guía

Se explica paso a paso como realizar la operación de crear un WasteManager. Las demás consultas son similares.

1-Registrarse: Debe crear un usuario sino lo ha hecho.

POST http://localhost:8800/auth/register
Body:

```
{
      “username” : ”usuario”,
      “password” : ”password”
}
```

2-Inicia sesión

POST http://localhost:8800/auth/login
Body:

```
{
      “username” : ”usuario”,
      “password” : ”password”
}
```

3-Consulta: Una vez que haya iniciado sesión podrá realizar cualquier consulta, el token de autenticación es una cookie que se le fue almacenada. En este caso usaremos crear un usuario.

POST: http://localhost:8800/api/wasteManager/create
Body:

```
{
  "name": "John Doe",
  "nif": "12345678A",
  "isEnabled": true,
  "version": 1,
  "wasteManagerAddressEntity": {
    "address": "123 Main St",
    "isEnabled": true,
    "version": 1
  },
  "wasteCenterAuthorizationEntities": [{},{},{}]
}
```

4-Si desea ver todos los WasteManager almacenados puede realizar un GET http://localhost:8800/api/wasteManager