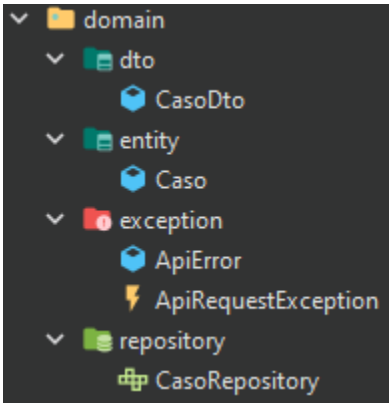
SOLUCIÓN PROPUESTA

Para la actividad planteada se decidió desarrollar 2 microservicios, los cuales son los siguientes:

- **ms-afiliados:** Este microservicio es el encargado de todas las operaciones que se realizan sobre la tabla de afiliados en la base de datos. Podemos encontrar las operaciones CRUD (Create, Read, Update, Delete) y algunos filtros de búsqueda como por ejemplo obtener un afiliado por Id, por numero de identificación, por usuario de creación y por un intervalo que filtra los afiliados que fueron creados entre una fecha inicial y una fecha final.
- **ms-casos:** De manera similar al ms-afiliados, este microservicio esta encargado de realizar todas las consultas y operaciones sobre la tabla de casos en la base de datos. Podemos realizar operaciones CRUD (Create, Read, Update, Delete) y filtrar nuestras búsquedas por el Id del caso, usuario de creación, el Id del gestor del caso y de igual manera contamos con un filtro para obtener los casos cuya fecha de inicio de caso se encuentre entre un intervalo de una fecha inicial y una fecha final.

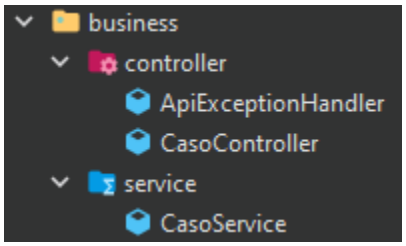
Los microservicios están desarrollados con la versión 8 de Java y la versión 2.7.0 de Spring Boot. Cada uno de los proyectos cuenta con la misma arquitectura, la cual está dividida en 3 capas:

- **domain:** En la capa de dominio vamos a encontrar todas las clases que van a definir las estructuras de nuestros objetos principales, es decir, vamos a encontrar las clases que hacen referencia a las tablas en la base de datos, de igual manera, encontraremos clases encargadas de la transferencia de información de nuestros objetos, mas conocidos como DTO, por otro lado, en esta capa se encuentran la estructura de las excepciones que van a ser lanzadas en nuestra aplicación y por último, vamos a encontrar la interacción con la base de datos, la cual se hace mediante un interface que implementa los métodos necesarios para realizar todo tipo de consultas sobre la base de datos.
  Es por esta razón que en la capa 'domain' vamos a encontrar algunos paquetes como lo son: entity, dto, exception, repository, entre otros.

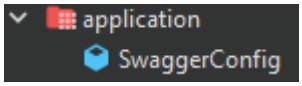


- **business:** La capa 'business', es la encargada de toda la lógica de negocio de la aplicación, es decir, vamos a encontrar todas las funcionalidades que resuelven el problema para el cual se creó, es decir, se deben realizar validaciones de las reglas de negocio, se deben

realizar flujo de acuerdo con ciertas condiciones cumplidas y se debe realizar el manejo de errores. Es por esto, que en esta capa vamos a encontrar los controladores de la aplicación, los cuales son los encargados de exponer rutas en una url, la cual hace referencia a un único recurso, estas rutas son consumidas por cualquier tipo de cliente, que necesite hacer uso de los recursos de nuestra aplicación. Y, por otro lado, encontraremos las clases encargadas de procesar toda esta información que es solicitada a la aplicación, aplicando las reglas de negocio y validaciones previamente mencionadas, para de esta manera responder de manera correcta con el recurso que fue solicitado. Los paquetes que hacen parte de esta capa son los siguientes:



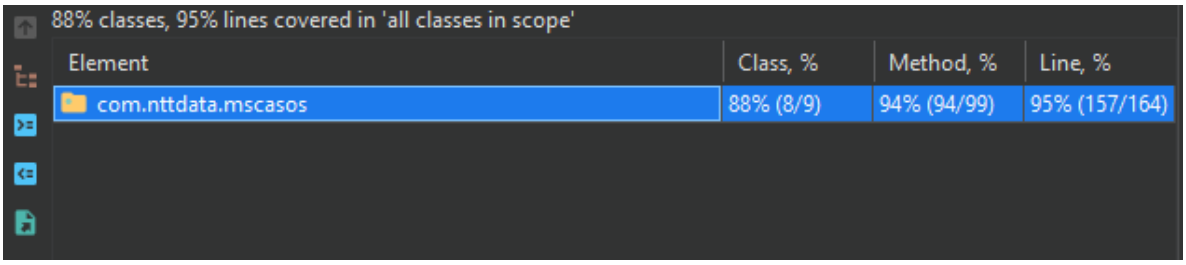
- **application:** En la capa de application, vamos a encontrar clases de configuración de arranque para nuestra aplicación, estas clases las utilizamos para implementar algunas características que nos ofrece Spring Boot, en este caso es necesario realizar una documentación de la aplicación con Swagger, por lo cual, utilizamos una clase de configuración que nos ayuda a crear una url en nuestro servidor, para poder consultar el swagger generado automáticamente.

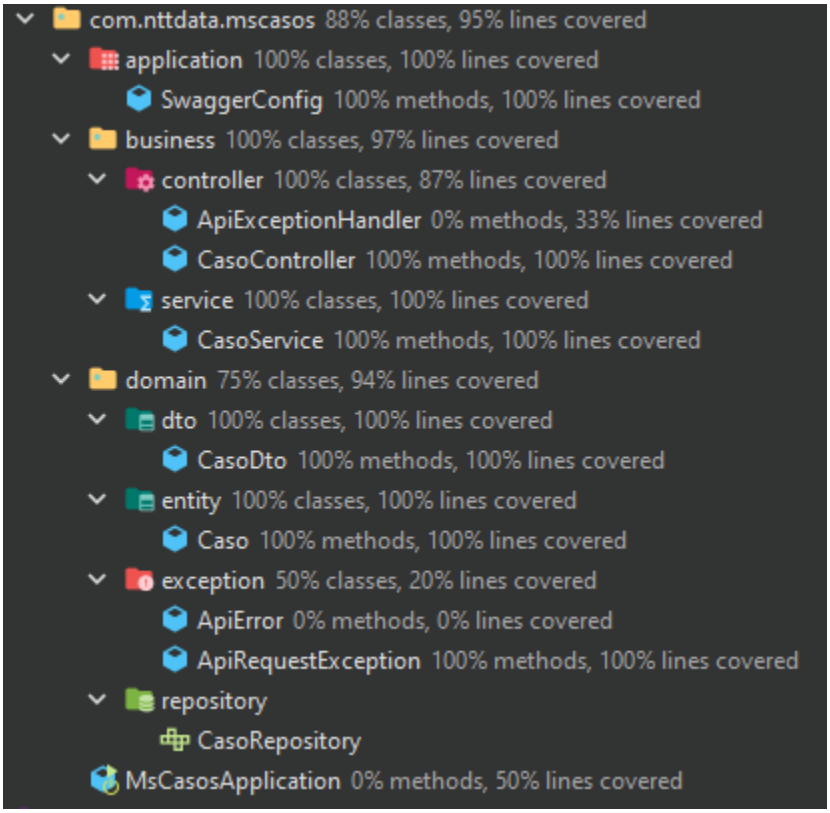


Swagger:

- **ms-afiliados:** **GitHub (Swagger ms-afiliados).**
- **ms-casos:** **GitHub (Swagger ms-casos).**

Finalmente, es necesario probar nuestra aplicación, para debemos desarrollar pruebas unitarias que se encarguen de verificar que cada una de nuestras líneas de código escritas funcionen como lo deben hacer. JUnit es la tecnología que nos ayuda a escribir y configurar nuestras pruebas unitarias, estas las podemos encontrar en el paquete de 'test' y están separadas para los controladores y para los servicios, ya que, estos requieren de una configuración diferente, en los controladores se deben simular peticiones a nuestro servidor y en los servicios, se debe hacer llamado de los métodos creados. Con alguna herramienta de análisis de código estático, se asegura que las pruebas unitarias desarrollados cuentan con un porcentaje de cobertura mínimo de 80%, esto para asegurar que escribimos un código de calidad y que cumple con la función para el cual fue desarrollado.


88% classes, 95% lines covered in 'all classes in scope'
Element
Class, %
Method, %
Line, %
com.nttdata.mscasos
88% (8/9)
94% (94/99)
95% (157/164)


com.nttdata.mscasos 88% classes, 95% lines covered
application 100% classes, 100% lines covered
SwaggerConfig 100% methods, 100% lines covered
business 100% classes, 97% lines covered
controller 100% classes, 87% lines covered
ApiExceptionHandler 0% methods, 33% lines covered
CasoController 100% methods, 100% lines covered
service 100% classes, 100% lines covered
CasoService 100% methods, 100% lines covered
domain 75% classes, 94% lines covered
dto 100% classes, 100% lines covered
CasoDto 100% methods, 100% lines covered
entity 100% classes, 100% lines covered
Caso 100% methods, 100% lines covered
exception 50% classes, 20% lines covered
ApiError 0% methods, 0% lines covered
ApiRequestException 100% methods, 100% lines covered
repository
CasoRepository
MsCasosApplication 0% methods, 50% lines covered