Universitat Oberta de Catalunya

uoc.edu

API modular segura

Webster Cosme de la Rosa
Protección de APIs REST
Seguridad empresarial

Nombre Tutor/a de TF
Nombre Profesor/a responsable de la asignatura

9 de enero del 2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	*API modular segura*
Nombre del autor:	*Webster Cosme de la Rosa*
Nombre del consultor/a:	*Pau del Canto Rodrigo*
Nombre del PRA:	*Victor García Font* *Andreu Pere Isern*
Fecha de entrega (mm/aaaa):	*01/2023*
Titulación o programa:	Máster Universitario en Ciberseguridad y Privacidad
Área del Trabajo Final:	*Seguridad empresarial, protección de APIs REST*
Idioma del trabajo:	*Castellano*
Palabras clave	*Seguridad, Modular, Servicios*

Resumen del Trabajo

Se plantea una metodología desarrollo orientada hacia la extensión del negocio y a la escalabilidad operacional donde además se pretende la reducción de los costes iniciales. Por ello, se plantea una arquitectura modular monolítica donde un factor clave de la seguridad se delega en el API Gateway.

Los sistemas monolíticos modulares permiten la reducción del coste de la gestión de los servicios a la vez que permite dividir el problema del negocio en otros más simples. De esta manera, se dispone de la metodología adecuada para enfrentar un mercado exigente.

Un sistema capaz de evolucionar linealmente, sin que se haga más difícil de desarrollar a medida que aumentamos tanto la complejidad de negocio como el volumen del mismo, requiere de un planteamiento basado en la teoría y en la experiencia.

Gracias a la aplicación de la metodología de desarrollo de una API monolítica que es capaz de evolucionar de manera independiente en cada uno de módulos. El API Gateway protege cada uno de los servicios que se ofrecen. La vertebra central del proyecto es el OpenAPI dado que toda funcionalidad parte de ella.

En este proyecto se garantiza que solo las personas autorizadas a acceder a los datos pueden hacerlo y que estos a su vez se encuentren protegidos contra amenazas de diversa índole. No obstante, al final de esta memoria se hace evidente que la seguridad se puede seguir mejorando en por diferentes vías.

Abstract

A development methodology oriented towards the extension of the business and operational scalability is proposed, where it is also intended to reduce initial costs. For this reason, a monolithic modular architecture is proposed where a high grade of security factor is delegated to the API Gateway.

Modular monolithic systems allow reducing the cost of service management as well as the business problem to be divided into simpler ones. In this way, the appropriate methodology is available to face a demanding market.

A system capable of evolving linearly, without becoming more difficult to develop as both business complexity and volume increase, requires an approach based on theory and experience.

Thanks to the application of the development methodology, a monolithic API is capable of evolving independently in each of the modules. The API Gateway protects each of the services offered. The central vertebra of the project is the OpenAPI since all the functionality starts from it.

This project guarantees that only people authorized to access the data can do so and that these, in turn, are protected against threats of various kinds. However, at the end of this report it becomes evident that security can be further improved in different ways.

Índice

Lista de figuras

1. Introducción

Los sistemas monolíticos modulares [5] son el resultado de una arquitectura que no exige la creación de microservicios [6] desde un principio y que, por otra parte, da la posibilidad de crearlos sin apenas esfuerzo cuando sean necesarios. En este sentido, la seguridad y la privacidad en este plano modular tomará un papel fundamental.

1.1. Contexto y justificación del Trabajo

La arquitectura monolítica era la más común hasta hace más bien poco. En esta, los servicios ofrecidos por el sistema se encuentran desplegados bajo única aplicación. La gran ventaja de este modelo es que se dispone de todas funciones en un único proyecto por lo que el coste de gestión de servicios asociados del mismo es bajo al principio. Por otra parte, la principal desventaja resulta cuando el proyecto crece mucho y el proyecto se vuelve difícil de gestionar dando lugar a numerosos problemas entre ellos la escalabilidad.

En el lado opuesto tenemos a la arquitectura de microservicios que ha ido ganando fuerza en los últimos años. En esta, los servicios ofrecidos por el sistema se encuentran desplegados en diferentes aplicaciones. La gran ventaja de este modelo es que no cuenta con los inconvenientes de la arquitectura monolítica dado que ahora las funcionalidades se encuentran divididas por dominio, en múltiples proyectos. Por otra parte, la principal desventaja es que los costes iniciales de la gestión del proyecto son elevados.

Ante esta situación surgen los sistemas monolíticos modulares que aúnan las ventajas de la arquitectura monolítica y la de microservicios en una sola. En esta, las funcionalidades se encuentran divididas en múltiples dominios completamente independientes que interactúan entre sí mediante adaptadores o interfaces. Otra gran ventaja es que este método permite realizar cambios con facilidad durante la etapa inicial del proyecto con facilidad de manera que cuando el proyecto alcanza cierta madurez o bien cuando sea necesario los dominios se puedan dividir en aplicaciones independientes.

1.2. Objetivos del Trabajo

El objetivo de este trabajo es ser un ejemplo de cómo podemos llevar a cabo la implementación de una arquitectura monolítica modular bajo altos estándares de calidad donde hará hincapié en la seguridad y privacidad.

- Protección genérica: Hacer frente al mayor número de las vulnerabilidades que se encuentran descritas en el Top Ten de OWASP [7].

- Proxy inverso para la seguridad: Proteger de los servicios del API en diferentes aspectos mediante un API Gateway [8].

- Firewall de aplicaciones web: Supervisar, filtrar o bloquear el tráfico HTTP hacia y desde la aplicación web mediante un WAF [9].

Se limitará las cuotas del API para evitar que se vean desbordadas por un número excesivo de solicitudes.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

A través de la comprensión de la guía transversal sobre la CCEG dirigida al estudiantado [10] del trabajo final se puede determinar el impacto que tiene el desarrollo del TF en tres dimensiones:

- Sostenibilidad: Tiene un impacto inicialmente negativo se empleará potencia computacional para llevarlo a cabo con el consecuente gasto de energía y componentes informáticos. No obstante, debido a que su objetivo principal es la protección de los recursos que, en menor o mayor medida, han tenido un impacto digital, tiene un impacto positivo a largo plazo y por ello encaja mejor con el “ODS 9 – Industry, innovation and infraestructure”.

- Comportamiento ético y responsabilidad social: Tiene un impacto positivo ya que contribuye al bien de la sociedad, principalmente en un aspecto importante como la protección de los datos y por ello encaja mejor con “ODS 8 Decent work and economic growth”.

- Diversidad (género, entre otros) y derechos humanos: No tiene ningún impacto ni positivo ni negativo en aspectos de género, diversidad o de derechos humanos. En cambio, sí que lo hace y, además, positivamente en legislaciones tales como la RGPD [11].

1.4. Enfoque y método seguido

Las estrategias de infraestructura que valoramos en el TF son dos:

- AWS: Emplear la nube pública mediante los créditos que Amazon proporciona gratuitamente a través de AWS Educate [12] a los estudiantes.

- Herramienta de contenedores: Emplear Docker [13] para desplegar de manera local distintas herramientas que faciliten y/o posibiliten el desarrollo.

Se escoge la estrategia de contenedores por facilitar el desarrollo. Además, no se dispone de los créditos de AWS Educate.

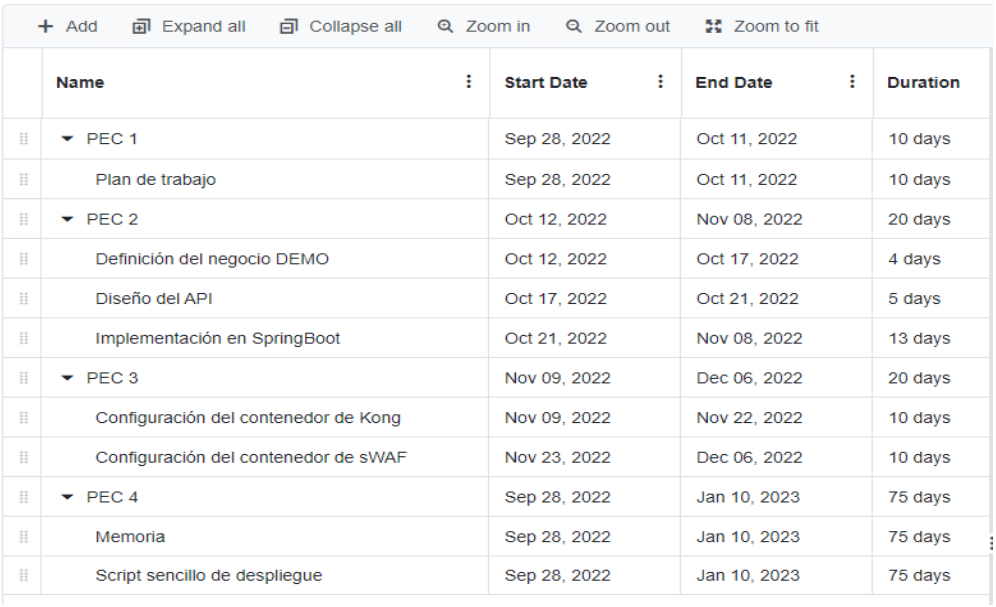
En definitiva, la estrategia principal de este proyecto es la que nos permite centrarnos en realizar diferentes usos de tecnologías que en base a la experiencia con ellas nos permiten ahorrar tiempo de gestión en las operaciones previas al uso de estas.

Teniendo esto en cuenta para el desarrollo del código se prevé emplear las siguientes tecnologías:

- Spring Framework [14] para el desarrollo del API monolítica modular.
- Kong [15] como API Gateway.
- sWAF[16] como Firewall de aplicaciones web.

Finalmente, en cuanto a la metodología de desarrollo, se empleará la Arquitectura Hexagonal [17] para la distribución del código del proyecto Java.

1.5. Planificación del Trabajo



Name	Start Date	End Date	Duration
PEC 1	Sep 28, 2022	Oct 11, 2022	10 days
Plan de trabajo	Sep 28, 2022	Oct 11, 2022	10 days
PEC 2	Oct 12, 2022	Nov 08, 2022	20 days
Definición del negocio DEMO	Oct 12, 2022	Oct 17, 2022	4 days
Diseño del API	Oct 17, 2022	Oct 21, 2022	5 days
Implementación en SpringBoot	Oct 21, 2022	Nov 08, 2022	13 days
PEC 3	Nov 09, 2022	Dec 06, 2022	20 days
Configuración del contenedor de Kong	Nov 09, 2022	Nov 22, 2022	10 days
Configuración del contenedor de sWAF	Nov 23, 2022	Dec 06, 2022	10 days
PEC 4	Sep 28, 2022	Jan 10, 2023	75 days
Memoria	Sep 28, 2022	Jan 10, 2023	75 days
Script sencillo de despliegue	Sep 28, 2022	Jan 10, 2023	75 days

Figura 1: Plan de trabajo

Este plan de trabajo define una serie de objetivos a gran escala bien marcados con la idea de que permitan ser flexibles durante el desarrollo del producto y de esta manera poder hacer frente a los imprevistos sin problema.

1.6. Breve sumario de productos obtenidos

- PEC 1: En esta primera entrega, se establecen los objetivos y metodologías con la cual se desarrollarán cada una de las posteriores entregas.

- PEC 2: En la segunda entrega se implementa un API modular basada en Spring cuyo fin no es plantear un gran problema de negocio si no realizar una arquitectura.

- PEC 3: En al tercera entrega se protege al API modular mediante un API Gateway. Se analiza la utilización de un cortafuegos de aplicaciones web (WAF).

- PEC 4: En esta cuarta y última entrega se revisa toda la memoria de cara a solucionar posibles erratas y a mejorar todo el contenido de la misma teniendo en cuenta las sugerencias del tutor.

1.7. Breve descripción de los otros capítulos de la memoria

- Diseño: Antes de la puesta en marcha del desarrollo del proyecto se llevará a cabo un diseño de tal manera que plante las bases del producto final.

- Implementación: En esta fase se emplean un gran conjunto de conceptos de ingeniería del software para obtener un producto modular dotado de un fuerte componente teórico.

- Integración continua: Se dedica un apartado exclusivo para la explicación de la metodología de operaciones que se emplea para empaquetar el producto y su posterior despliegue al alcance del API Gateway.

- Conclusión: En último término, se determinan las ideas originadas durante las fases de diseño y desarrollo obtenidos donde se contrasta la planificación con los resultados obtenidos.

2. API Modular

2.1. Diseño

2.1.1. Objetivo del producto

El objetivo de este producto es crear una interfaz de programación de aplicaciones (API) modular, extensible y preparada para ser integrada con el resto de productos que se desarrollarán en los módulos de API Gateway y WAF.

En definitiva, la definición del dominio o un alcance alto del mismo no son el objetivo principal de este producto y, por el contrario, sí que lo es crear una sería de interfaces de entrada y/o de salida que permitan aplicar diferentes mecánicas de seguridad.

2.1.2. Definición del dominio

El negocio del producto se basa en la tienda de mascotas de mascotas de Swagger[20]. Es un dominio ampliamente utilizado como modo de ejemplo para la implementación demostraciones de tecnología y metodologías.

Hemos escogido este dominio ya que es lo suficientemente amplio como para que nos permita cumplir los objetivos del producto y, en especial, el de la aplicación de diferentes mecánicas de seguridad.

En este sentido, el negocio está formado por los siguientes dominios:

- Mascota: Operaciones sobre las mascotas de la tienda.
- Pedidos: Operaciones sobre pedidos de la tienda.
- Usuarios: Operaciones sobre usuarios de la tienda

Figura 2: Dominio de la aplicación

Cada uno de estos dominios permite, como mínimo, realizar las operaciones de creación, selección, actualización y borrado, también conocidas por el acrónimo CRUD en el ámbito de programación informática.

2.1.3. Metodología de desarrollo

La metodología de desarrollo que se emplea en el proyecto es la siguiente:

1) Determinar la especificación del API.
2) Crear pruebas para definir funcionamiento en función de la especificación. Inicialmente estos fallan.
3) Implementar el código de manera que las pruebas pasen.

Es importante destacar que todos los pasos empleados para desarrollar el producto, o de la metodología, cumplen con los criterios de seguridad establecidos.

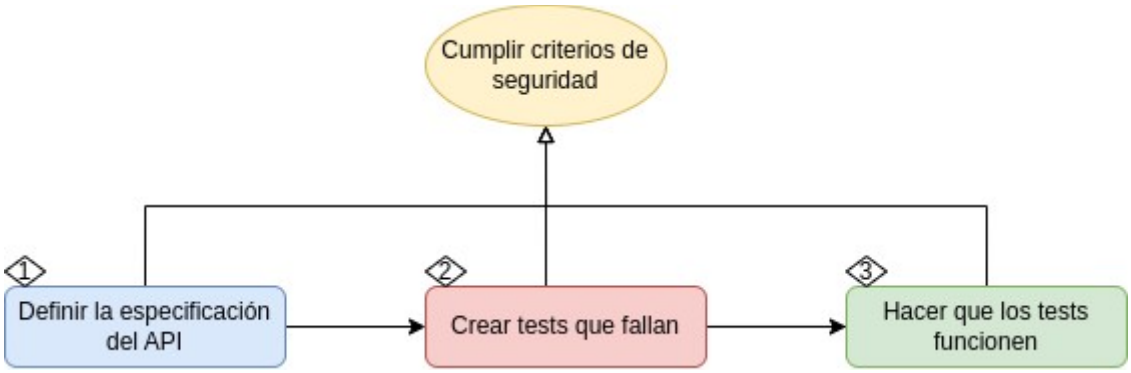


Figura 3: Metodología

Se escoge esta metodología porque es muy común que, en un contexto empresarial donde se pretende resolver un problema complejo, múltiples equipos independientes trabajen a la vez sobre la solución y, por este motivo, es necesario que todos los colaboradores dispongan de la especificación de cada una de las partes desde el inicio de la fase de desarrollo del proyecto.

En los siguientes apartados se detalla el diseño de cada una de las partes y/o conceptos que conforman la metodología de desarrollo.

2.1.4. API First

API First [1] es una metodología de desarrollo que conlleva definir el contrato antes de iniciarse con la implementación de la aplicación. Este hecho supone una serie de impactos positivos en el proyecto:

- La cantidad de tiempo que se necesita desde la concepción del producto hasta que es revelado al mercado, también conocido como time-to-market (TTM).

- Los consumidores del API detectar incoherencias y proponer las soluciones y/o mejoras durante la fase de diseño lo que supone un ahorro de tiempo considerable.

- Como el API se define bajo una especificación, como por ejemplo OpenAPI 3.0 Specification (OAS) [4], permite la automatización de procesos ligados a fases posteriores tal y como la definición del código o bien pruebas de contrato.

- La seguridad mejora dado que permite establecer criterios de seguridad desde la fase de diseño tal como la validación de los datos o bien la definición de las interfaces que requieren autenticación o incluso del tipo de autorización a través del permiso de lectura y/o escritura sobre un recurso determinado.

En definitiva, esta metodología supone diferenciar el diseño del API de la implementación, lo que conlleva a una serie de ventajas que suponen que la colaboración entre diferentes equipos mejore.

2.1.5. OpenAPI 3.0

OAS 3.0 es la especificación agnóstico del lenguaje de implementación que vamos a emplear para definir el API REST. Es un lenguaje de alto nivel puesto que permite tanto a los humanos como a la máquina entender el significado del mismo sin la necesidad de acceder al código fuente o a la documentación adicional.

En la siguiente figura podemos encontrar una descripción visual de los componentes de OAS 3.0. En ella vemos claramente que podremos aplicar requisitos de seguridad desde fases tempranas, en concreto con la fase de diseño.

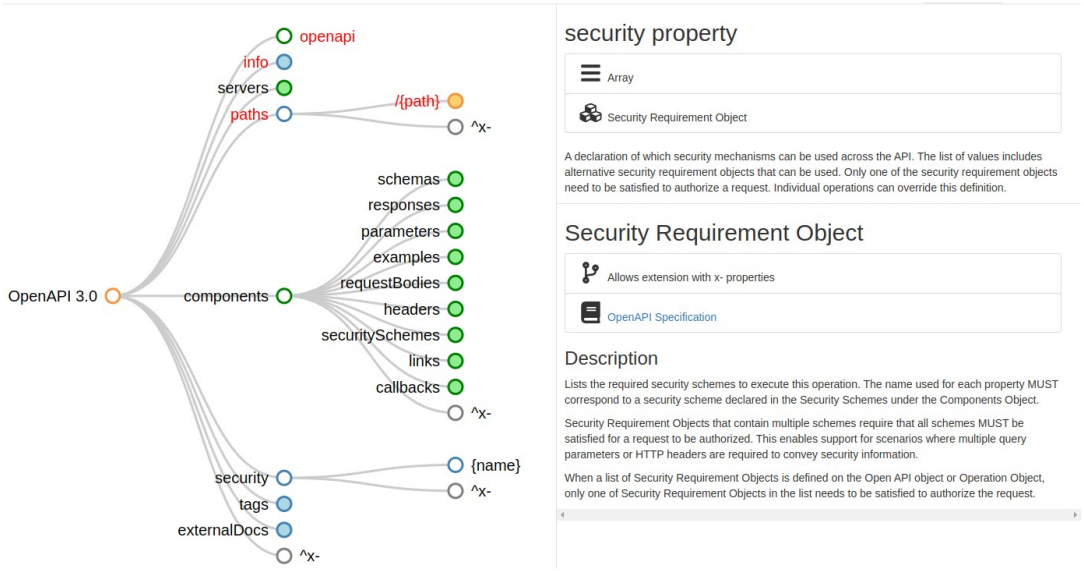


Figura 4: Arnaud Lauret's – OpenApi Map [18]

2.1.6. Mecánicas de seguridad

El desarrollo de este producto cumple con el objetivo del global del trabajo final que es el de conseguir una API segura. Esto se consigue, en gran medida, gracias al respaldo durante la fase de diseño e implementación de soluciones que hacen frente al Top Ten de OWASP de vulnerabilidades que mencionamos en la sección de Objetivos del Trabajo.

2.1.7. Arquitectura hexagonal

El uso de la arquitectura hexagonal nos permite diferenciar muy bien el dominio, es decir, la problemática que se pretende resolver, de los puertos que se emplean para interactuar con el mismo.

Los adaptadores son la implementación de los puertos, es decir, de las necesidades del dominio, de tal manera que estas responsabilidades quedan delegadas exclusivamente a estos.

Podemos diferenciar los adaptadores en dos tipos:

- Primarios: Aquellos que emiten instrucciones a los puertos primarios del dominio. Hacen uso los puertos primarios del dominio. La responsabilidad de estos recae en adecuar las instrucciones de los clientes del dominio.

- Secundarios: Aquellos que reciben instrucciones de los puertos secundarios del dominio. La responsabilidad de estos recaen en implementar una solución que cumpla con las necesidades del dominio.

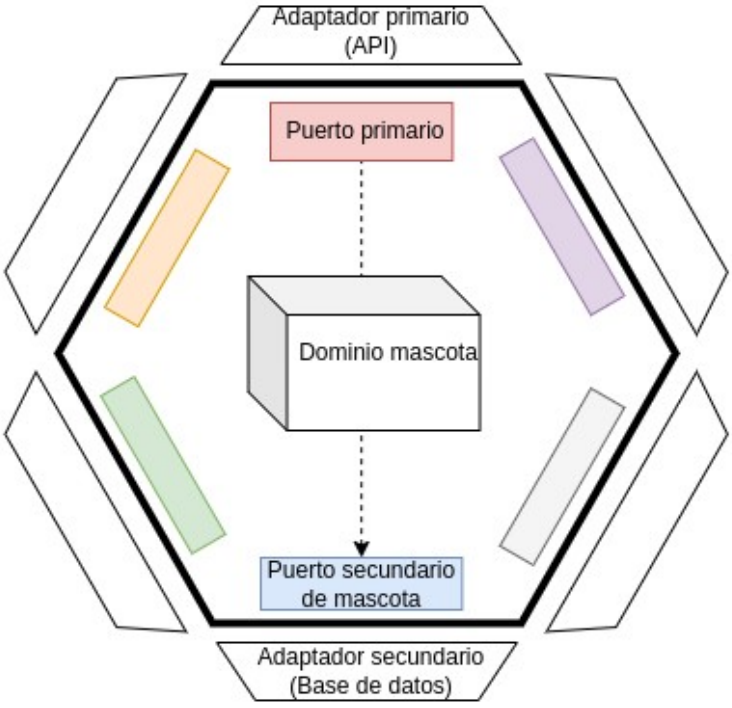


Figura 5: Arquitectura hexagonal

En nuestro caso, los adaptadores primarios son la implementación de la especificación del API, y los adaptadores secundarios son la capa de persistencia que emplearemos para guardar los recursos de la aplicación (mascotas, usuarios y los pedidos de la tienda).

2.1.8. Monolito modular

Uno de los objetivos del proyecto es crear un monolito modular por lo que se crea un único puerto de entrada que incluye a todos los módulos de la aplicación.

2.1.9. Módulos

Uno de los objetivos del proyecto es la división de la solución en módulos. Ahora bien existen diferentes maneras de dividir el código, las más comunes son:

- Vertical: Consiste en la agrupación por capas. En este caso, los adaptadores primarios, los adaptadores secundarios y el dominio relacionados con todas las funcionalidades se encuentran en diferentes unidades.

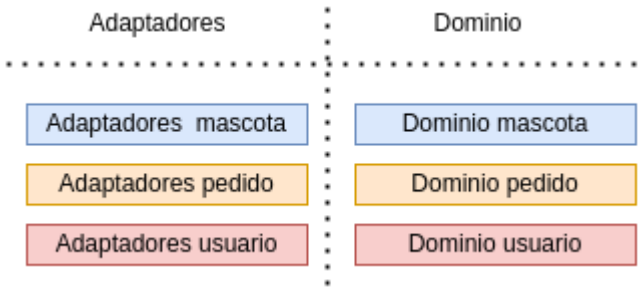


Figura 6 Separación vertical

- Horizontal: Consiste en la agrupación por funcionalidad. En este caso, los adaptadores primarios, los adaptadores secundarios y el dominio relacionados con una misma funcionalidad se encuentran en una misma unidad.

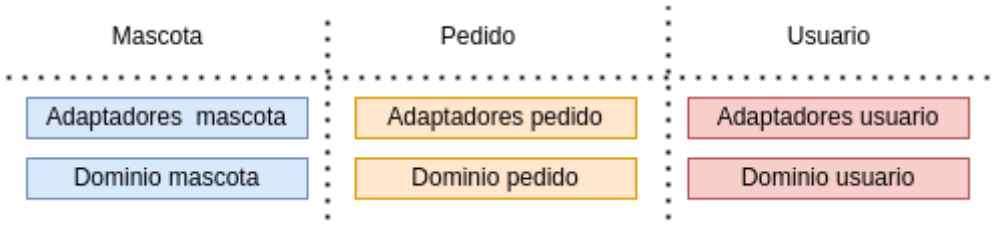


Figura 7: Separación horizontal

- Mixta: Consiste en utilizar la división vertical y la horizontal conjuntamente. En primer lugar, se emplea la división horizontal para dividir por funcionalidad y, en segundo lugar, por capas para continuar con la agrupación tradicional.

En este producto se opta por la división únicamente horizontal de tal manera que nos permita crear módulos de funcionalidad independientes que se puedan convertir en microservicios en un futuro y que a su vez sus componentes puedan ser divididos en otros módulos independientes cuando se requiera.

2.1.10. Desarrollo guiado por pruebas

El diseño guiado por pruebas, que cuenta con el acrónimo TDD (*test driven design*) en inglés, es una práctica en la ingeniería del software que consiste en el empleo de pruebas para diseñar la estructura del programa de manera que este sea fácil de probar y por lo tanto fácil de mantener.

Una ventaja adicional es que el programa permitirá ser modificado (ya sea en la búsqueda de una mayor simpleza, rendimiento o envergadura del alcance funcional) sin riesgo de que el comportamiento del programa cambie puesto que este está asegurado por las pruebas iniciales.

Para aplicar esta práctica hace falta aplicar tres pasos cíclicos:

1. Definición del test en función de un requisito.
2. Comprobación de que este test falla.
3. Modificar el código fuente de manera que todas las pruebas, esta y las anteriores, funcionen.
4. Actualización debido a una necesidad de cambio. Solo si se agrega un nuevo requisito es necesario escribir un nuevo test.

Figura 8: Desarrollo guiado por pruebas

2.2. Implementación

2.2.1. Entorno de desarrollo

Para el desarrollo de este producto se utilizan las siguientes herramientas:

- Sistema operativo y equipo de trabajo:

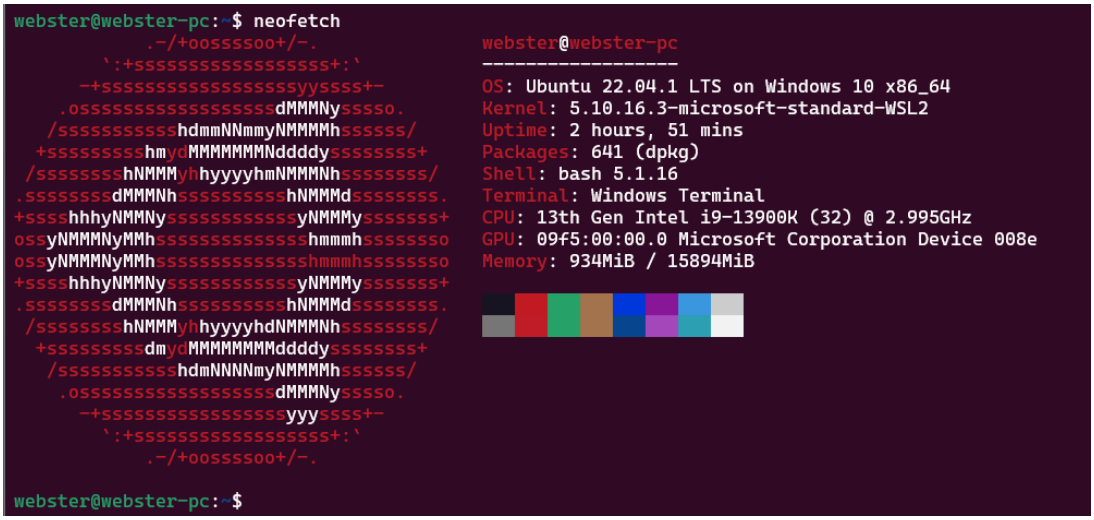


Figura 9: Sistema operativo

- Java

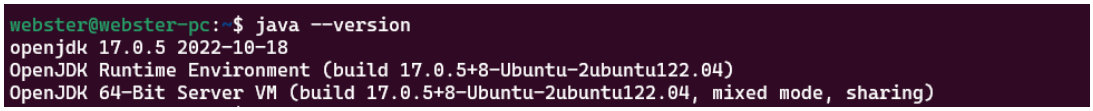


Figura 10: Versión de Java

- Maven

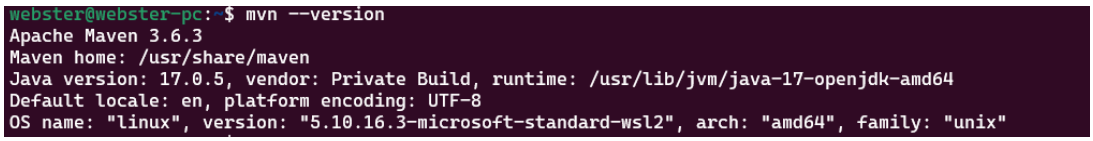


Figura 11: Versión de Maven

- Docker

Figura 12: Versión de Docker

- IntelliJ IDEA

Figura 13: Versión de Intellij IDEA

2.2.2. Módulos maven

Escogemos maven como herramienta de gestión para administrar el proyecto java y sus dependencias. En ella se crean un total de cinco módulos para hacer frente al requisito de Monolito modular establecido que podemos diferenciar en dos tipos.

- Módulos de dominio que nos ofrecen una solución a un problema determinado.
    - Mascota: Agrupa el código relacionado con las mascotas.
    - Tienda: Agrupa el código relacionado con los pedidos.
    - Usuario: Agrupa el código relacionado con los usuarios.
- Módulos de gestión se tienen la finalidad de dar soporte y/o emplear a los módulos de dominio.
    - Parent: Se encarga de definir las dependencias y configuraciones que serán empleadas por todos los módulos de dominio.
    - Application: Emplea a los módulos de dominio como dependencias de tal manera que los convierte en una única aplicación, un monolito.

En la siguiente figura observamos la definición de los módulos y cómo estos se encuentran al mismo nivel jerárquico.

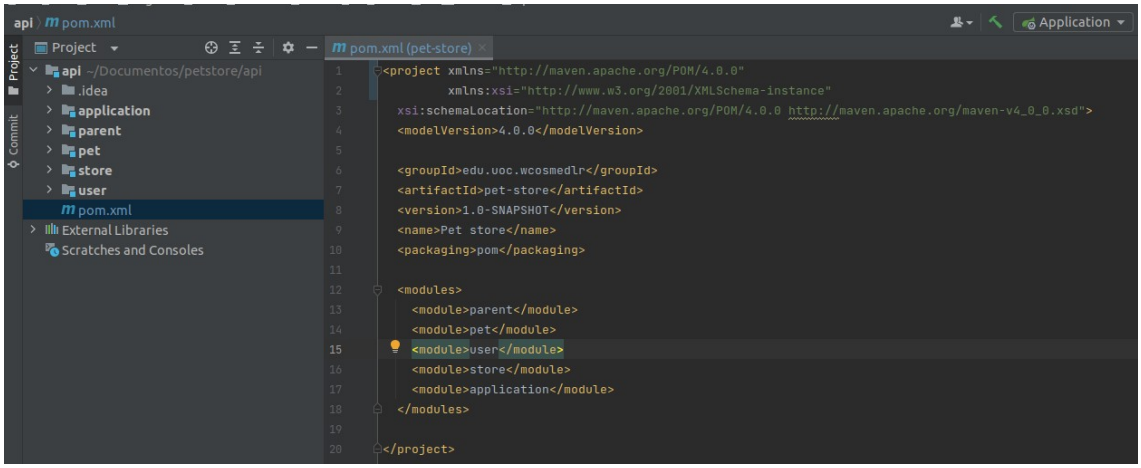


Figura 14: Módulos maven

Por otra parte, se emplea Maven Reactor [19] que es un mecanismo de Maven que se encarga de gestionar los proyectos multi-módulo. Su funcionamiento es el siguiente:

1. Determina la lista de módulos disponibles.
2. Ordena los módulos en función de las dependencias que tengan entre ellos.
3. Construye los módulos de manera ordenada.

Otra ventaja de emplear un proyecto multi-módulo a diferencia de un monolito no modular, es que permite la construcción paralela de diferentes partes del mismo y, en este sentido, el coste temporal de construcción del proyecto puede llegar a ser dividido entre el número de módulos que se tengan definidos.

2.2.3. OpenAPI generator

El primer paso de la Metodología de desarrollo consiste en establecer el OpenAPI 3.0 Specification. Para el ejercicio demostración de tienda de mascotas hay varias disponibles así que en lugar de crearla nosotros lo que haremos será escoger la definida por Swagger [20].

A continuación, lo que hacemos es dividir la especificación en tres partes para que encaje en los módulos Pet, Store y User definidos. Los documentos resultantes los dejamos en la dirección de «módulo / src / main / resources / openapi.yaml».

Después, empleamos el complemento de Maven OpenAPI Generator [21] en su modalidad Spring de la parte del servidor para que nos genere las interfaces en código java y adaptada al framework de Spring asociada al documento OAS.

Por último, se ha convenido apropiado configurar este complemento para que use el patrón de diseño Delegación [22] y, de esta manera, lo que implementamos manualmente a través de los adaptadores primarios nunca es alterado por la generación de código automática del complemento.

2.2.4. Definición del alcance de la implementación

Para llevar a cabo el Objetivo del producto se considera que con implementar uno o dos casos de uso es más que suficiente y, por tanto, escogemos las operaciones de creación y selección del módulo de mascotas.

2.2.5. Pruebas

El segundo paso de la Metodología de desarrollo consiste en establecer las pruebas donde lo normal es que estos inicialmente fallen dado que aún no se ha llevado a cabo la implementación de los mismos.

Para ello, creamos dos pruebas para cada una de las operaciones que se encuentran en la Definición del alcance de la implementación, donde se pretende contemplar el escenario ideal y su opuesto.

- Creación cuando la mascota no existe: Escenario idílico. La clave primaria y/o claves únicas de la mascota determinadas en la capa de persistencia no se encuentran en uso en la base de datos. El código HTTP esperado es el 200 (OK).

- Creación cuando la mascota existe: Escenario no idílico. La clave primaria y/o claves únicas de la mascota determinadas en la capa de persistencia se encuentran en uso en la base de datos. El código HTTP esperado es el 409 (Conflicto).

- Selección cuando la mascota existe: Escenario idílico. El factor de búsqueda que se emplea, en este caso el id de la mascota, se encuentra definido en la base de datos. El código HTTP esperado es el 200 (OK).

- Selección cuando la mascota no existe: Escenario no idílico. El factor de búsqueda que se emplea, en este caso el id de la mascota, no se encuentra definido en la base de datos. El código HTTP esperado es el 404 (No encontrado).

La implementación de estas pruebas se encuentran en « pet / src / test / java / edu / uoc / wcosmedlr / pet / PetApplicationTests ».

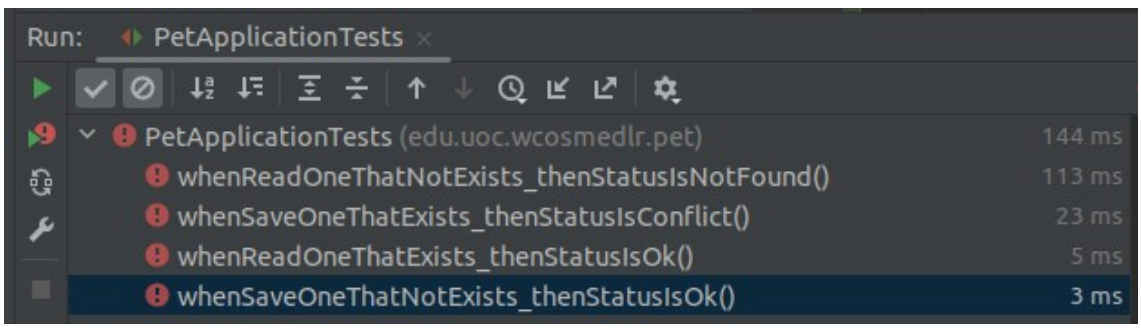


Figura 15: Definición de las pruebas

Observamos que las pruebas fallan inicialmente, tal y como estaba previsto.

2.2.6. Corrección de las pruebas

El tercer paso de la Metodología de desarrollo consiste actualizar la aplicación para que las pruebas dejen de fallar. En este caso, tendremos que implementar el Adaptador primario, el Dominio y el Adaptador secundario tal y como vemos en los siguientes apartados.

2.2.7. Adaptador primario

La implementación de este apartado se encuentra en « pet / src / main / java / edu / uoc / wcosmedlr / pet / PetApiAdapter ».

```java
public class PetApiAdapter implements PetApiDelegate {

    3 usages
    private final PetService petService;

    Webster Cosme de la Rosa
    @Autowired
    public PetApiAdapter(PetService petService) { this.petService = petService; }

    1 usage  Webster Cosme de la Rosa *
    @Override
    public ResponseEntity<Void> addPet(Pet body) {
        PetRepositoryEntity petRepositoryEntity = PetRepositoryEntity.builder()
                .id(body.getId())
                .name(body.getName())
                .build();
        return petService.addPet(petRepositoryEntity) Optional<PetRepositoryEntity>
                .map(petRepository -> ResponseEntity.ok().<~>build()) Optional<ResponseEntity<Void>>
                .orElseGet(() -> ResponseEntity.status(CONFLICT).build());
    }

    1 usage  Webster Cosme de la Rosa
    @Override
    public ResponseEntity<Pet> getPetById(Long petId) {
        return petService.getPetById(petId) Optional<PetRepositoryEntity>
                .map(petRepositoryEntity -> {
                    Pet pet = new Pet();
                    pet.id(petRepositoryEntity.getId());
                    pet.name(petRepositoryEntity.getName());
                    return pet;
                }) Optional<Pet>
                .map(ResponseEntity::ok) Optional<ResponseEntity<Pet>>
                .orElseGet(() -> ResponseEntity.notFound().build());
    }
}
```

Figura 16: Adaptador primario

Por una parte tenemos el método «addPet» que recibe un tipo «Pet», lo convierte al tipo «PetRepositoryEntity» (Un tipo que el dominio entiende), y realiza la petición al dominio a través de PetService. En caso de que tenga éxito devuelve un 200 OK y un 409 en caso contrario.

Por último tenemos el método «getPetById» que recibe un tipo «Long» y realiza la petición al dominio. En caso de que este registro exista convierte el tipo proporcionado por el dominio «PetRepositoryEntity» al tipo que se espera en la respuesta definida en el OAS «Pet». En caso de que existe se devuelve un 200 y un 404 en caso contrario.

2.2.8. Dominio

La implementación de este apartado se encuentra en « pet / src / main / java / edu / uoc / wcosmedlr / pet / PetService ». Esta clase emplea la anotación transaccional para asegurar que el conjunto de comandos que conforman una operación a nivel de servicio solo se persistan cuando todas tengan éxito.

```
2 usages  Webster Cosme de la Rosa *
@Service
@Transactional
public class PetService {

    4 usages
    private final PetRepositoryAdapter petRepositoryAdapter;

    Webster Cosme de la Rosa
    @Autowired
    public PetService(PetRepositoryAdapter petRepositoryAdapter) {
        this.petRepositoryAdapter = petRepositoryAdapter;
    }

    1 usage  Webster Cosme de la Rosa *
    public Optional<PetRepositoryEntity> addPet(PetRepositoryEntity petRepositoryEntity) {
        Optional<PetRepositoryEntity> optionalPetRepositoryEntity =
                petRepositoryAdapter.findById(petRepositoryEntity.getId());
        if (optionalPetRepositoryEntity.isPresent()) {
            return Optional.empty();
        } else {
            return Optional.of(petRepositoryAdapter.save(petRepositoryEntity));
        }
    }

    1 usage  Webster Cosme de la Rosa
    public Optional<PetRepositoryEntity> getPetById(Long petId) {
        return petRepositoryAdapter.findById(petId);
    }
}
```

Figura 17: Dominio

Por una parte tenemos el método «addPet» que recibe un tipo «PetRepositoryEntity» que emite la petición de guardado a la base de datos en caso de que el identificador de la mascota no exista ya en el adaptador secundario.

Por último, tenemos el método «getPetById» que recibe un tipo «Long» y emite la petición al adaptador secundario.

2.2.9. Adaptador secundario

La implementación de este apartado se encuentra en «pet / src / main / java / edu / uoc / wcosmedlr / pet / PetRepositoryAdapter». La base de datos que se emplea es H2 [23].

```
3 usages  Webster Cosme de la Rosa *
@Component
public interface PetRepositoryAdapter extends JpaRepository<PetRepositoryEntity, Long> {
}
```

Como se puede observar, esta clase no cuenta con ningún método manual y sin embargo cumple con todas las necesidades que tenemos. Esto es así porque esta extiende de la clase JpaRepository, que pertenece a Spring Data JPA [24], que ya implementa el CRUD de manera automática para esta entidad.

Por otra parte en «pet / src / main / java / edu / uoc / wcosmedlr / pet / PetRepositoryEntity».

```
9 usages  Webster Cosme de la Rosa
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Data
public class PetRepositoryEntity {
    @Id
    private Long id;
    private String name;
}
```

Esta clase sirve para determinar el modelo del adaptador secundario. Es importante remarcar que estamos empleando esta clase también para el dominio y que en un proyecto de más envergadura conviene emplear clases separadas porque las necesidades entre estos dos niveles de abstracción difieren siempre.

2.2.10. Comprobación de las pruebas

Cuando se completa la implementación de todas las partes se comprueba que las pruebas que inicialmente fallaban ahora funcionan. Esto es lo que se puede observar en la siguiente figura.

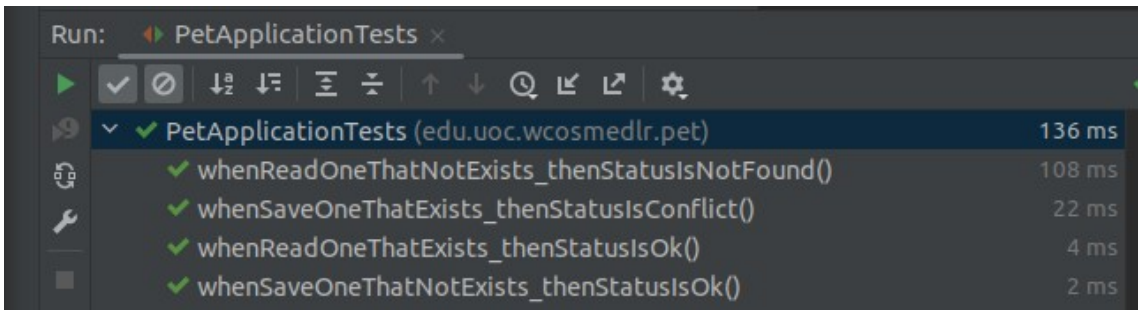


Figura 18: Pruebas exitosas

2.2.11. Spring-doc-api

Para finalizar el producto empleamos el complemento spring-docapi [25] de manera que automatice la generación del OAS en función del código implementado. Este hecho nos permite lanzar módulos individuales y/o conjuntos de tal manera que el API que se reflejará será la de los módulos que se estén empleando. En el caso del módulo Application se reflejarán el API de los tres módulos de dominio.

2.3. Producto obtenido

Tras la aplicación rigurosa de la metodología de desarrollo se obtiene un API monolítica modular.

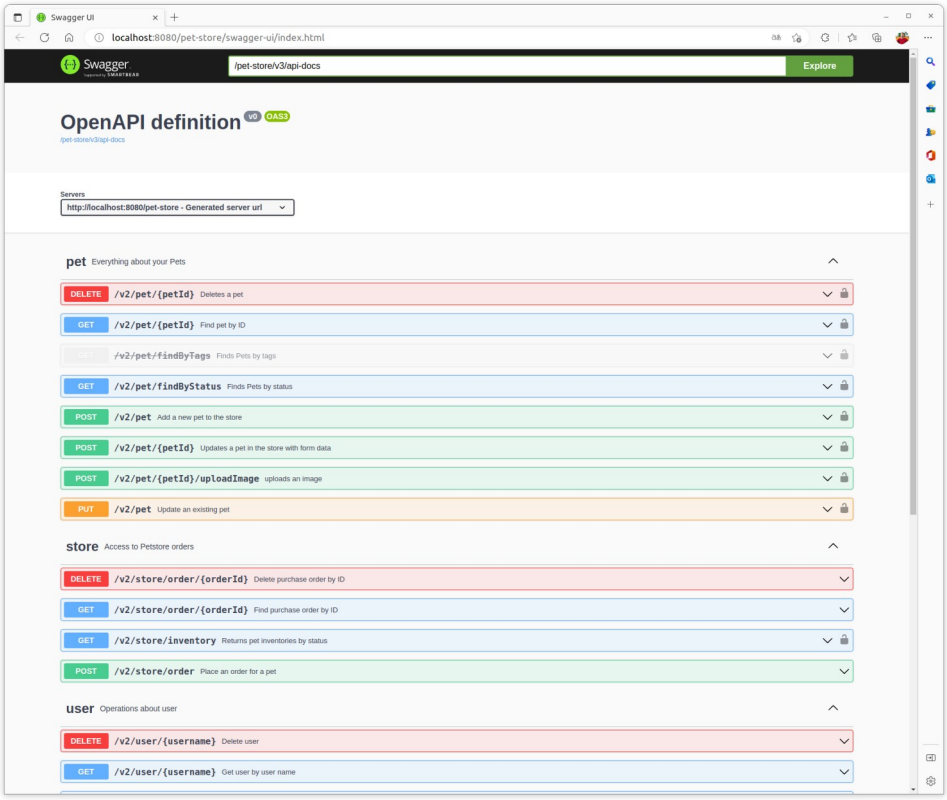


Figura 19: API generada por el módulo de aplicación

En la figura anterior observamos que el módulo de aplicación, a pesar de no tener ninguna clase implementada por sí mismo, nos permite visualizar e interactuar con el API de todos los módulos de dominio.

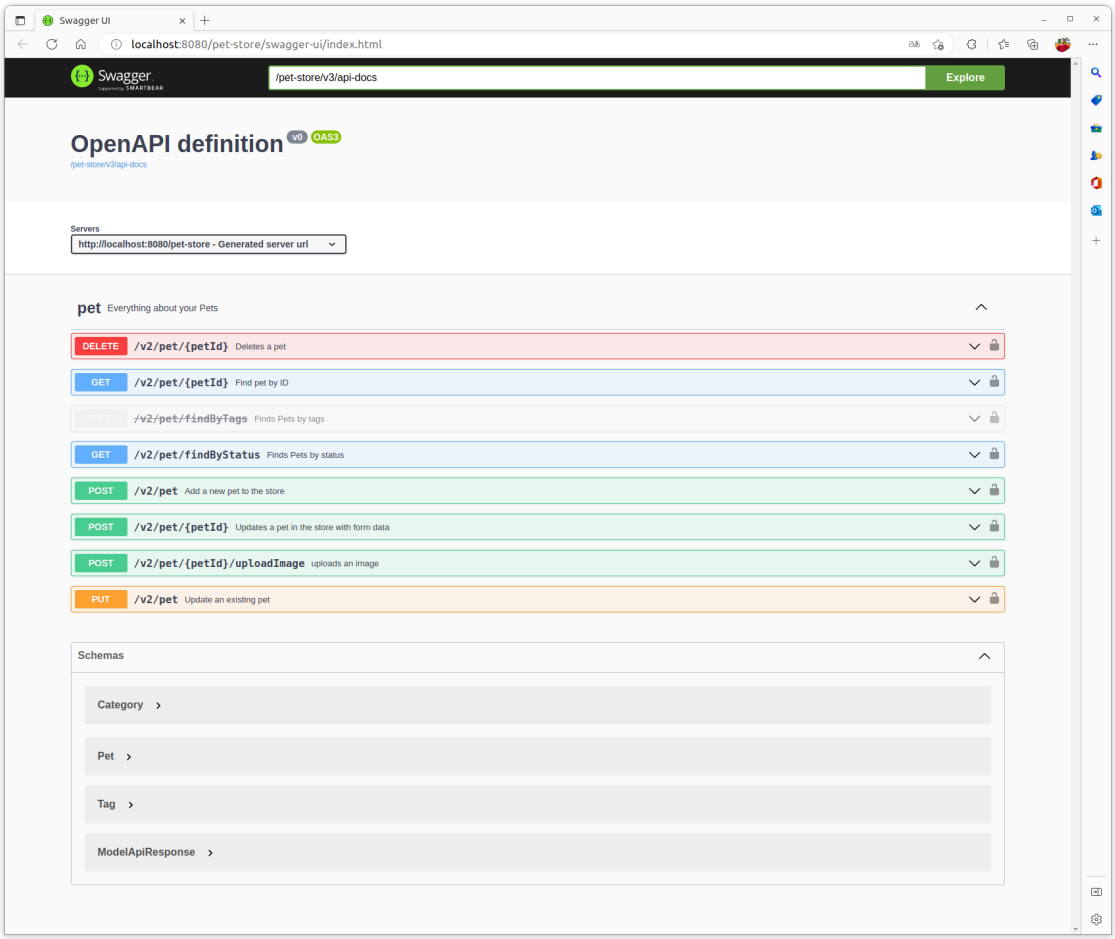


Figura 20: API del módulo de mascotas

Una de las posibilidades del producto obtenido es la de ejecutar los módulos de manera independiente. En la figura anterior podemos observar como el módulo de mascotas está siendo ejecutado de manera aislada al resto de módulos del dominio.

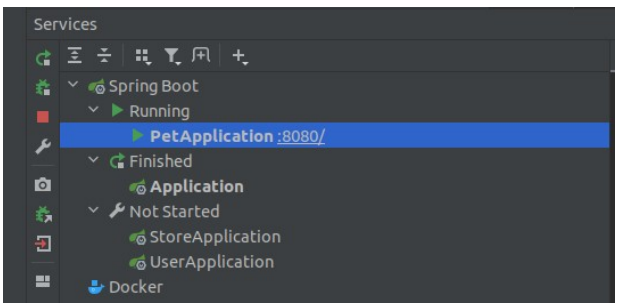


Figura 21: Lista de módulos ejecutables disponibles

En la figura anterior observamos la lista de módulos ejecutables disponibles.

3. API Gateway

3.1. Introducción

Una vez creado el API Modular, procedemos con la protección de la misma mediante Kong, el API Gateway escogido definido en la Planificación del Trabajo.

Esta herramienta gestionará los servicios y nos ofrecerá ciertas características que exploraremos. Cabe mencionar que la versión de pago de esta herramienta, Kong Enterprise [Error: no se encontró el origen de la referencia], ofrece características destacables a las que no tendremos acceso.

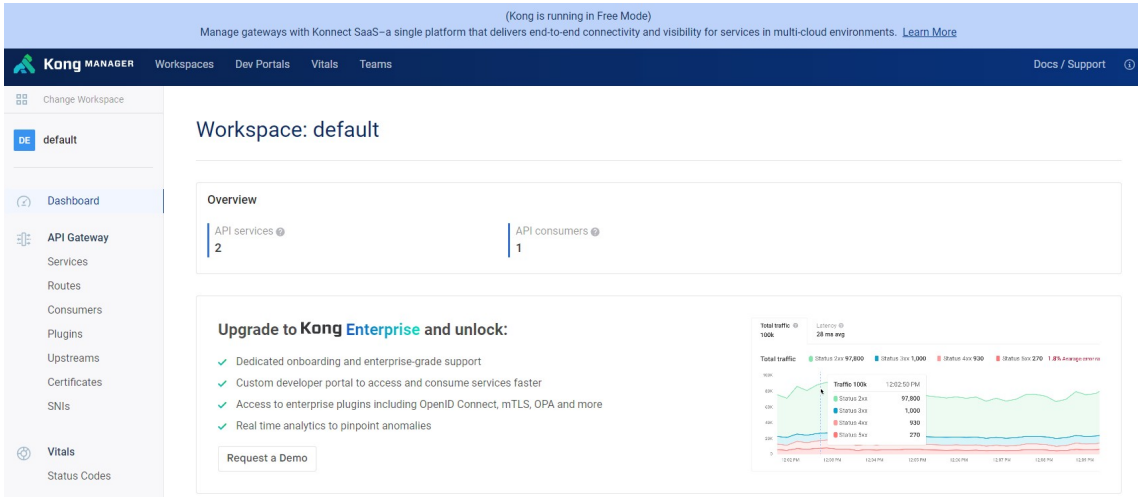


Figura 22: Interfaz web del menú principal de Kong

3.2. Configuración de servicios

Los servicios son las abstracciones que creamos para hacer referencia aquellas entidades que van a satisfacer determinadas peticiones. Creamos las siguientes:

- Petstore: Emplea el protocolo «http», el host «api», el puerto «8080» y el path «/api/v3». Este último parámetro es para evitar tener que indicar el path base en cada una de la rutas. En definitiva, la localización donde tenemos el API modular.

- Petstore-documentation: Emplea el protocolo «http», el host «api», el puerto «8080» y el path «/». En definitiva, la localización donde tenemos la documentación del api y la interfaz de swagger.

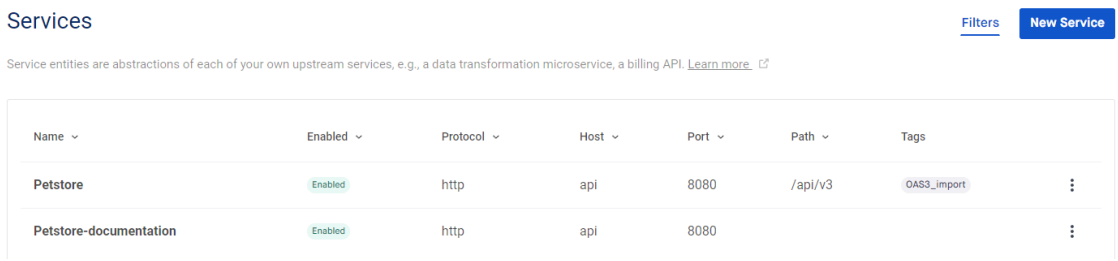


Figura 23: Configuración de los servicios en la interfaz web

3.3. Composición de rutas

Las rutas son las abstracciones que creamos para hacer referencia a las peticiones que hacen los clientes. Todas la rutas pertenecen a un servicio al cuál estas van a ser redirigidas. Se han definido todas las que están establecidas en el OpenAPI. El total de las rutas de los dos servicios definidos es 20 aproximadamente. En la siguiente figura se puede observar un ejemplo de 4 de ellas.

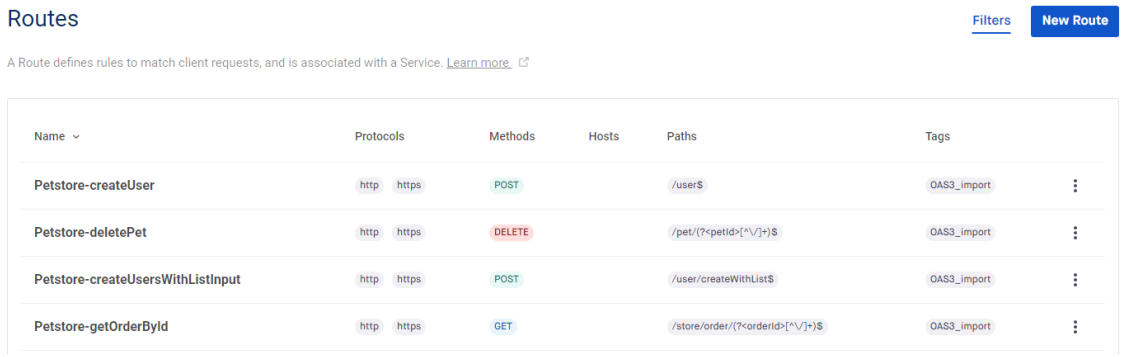


Figura 24: Configuración de las rutas en la interfaz web

Los campos más importantes que se definen son los siguientes:

- El nombre de la ruta.
- El protocolo http admitido.
- El método http correspondiente.
- El path que especificado mediante una expresión regular.

3.4. Configuración de consumidores

Los consumidores son las abstracciones que creamos para representar a los usuarios de los servicios. Se ha el usuario «Applicatrion1» para representar a la aplicación autenticada.

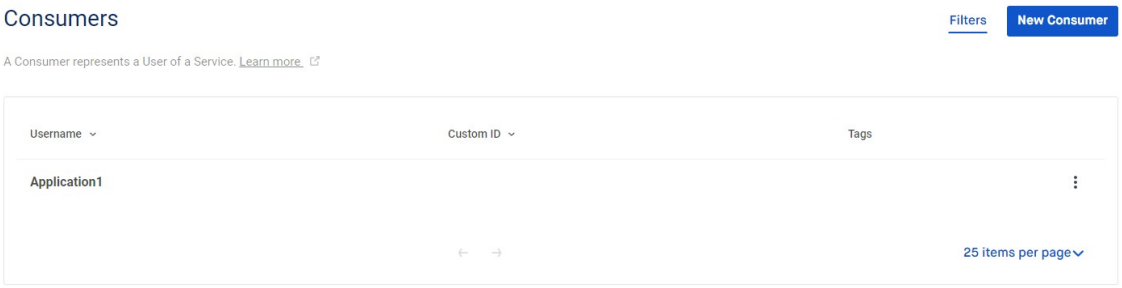


Figura 25: Configuración de los consumidores

3.5. Configuración de complementos

A los recursos que se han definido hasta este momento, tales como servicios, rutas y consumidores, se le aplican una serie de complementos para aumentar la seguridad del sitio web.

3.5.1. Key authentication

El primer complemento que configuraremos es el de la autenticación porque es uno de los más importantes, dado que nos permitirá identificar el consumidor que hace la petición y, por lo tanto, los permisos asociados a este.

La autenticación es una abstracción que puede ser implementada de diferentes maneras. Los siguientes tipos de autenticación son un ejemplo de ellas:

- Básica [27]: El nombre y la contraseña del usuario son requeridos en cada petición. Estos se deben enviar en la cabecera «*Authorization: Basic <credentials>*» donde las credenciales, compuesta por la unión del usuario y la contraseña mediante el carácter «;», están codificadas en base64.

- HMAC [28]: Se lleva a cabo la validación de la firma digital que son depositadas en la cabecera «Proxy-*Authorization*» y «*Authorization*» de las peticiones.

- Key [29]: Se requiere una clave válida asociada a un consumidor en cualquier zona habilitada por la configuración del complemento, como puede ser un parámetro, una cabecera o bien en el cuerpo de la petición.

- LDAP [30]: La autenticación se delega a un agente externo conocido como protocolo ligero de acceso a directorios. El consumidor debe indicar las cabeceras «Proxy-*Authorization*» y «*Authorization*». Este complemento solo se puede usar bajo la versión Enterprise.

- OAuth 2.0 Introspection [31]: La autenticación se delega a un agente externo conocido como servidor de autenticación abierta. El consumidor debe enviar un token en cada una de las peticiones que será validado en

el servidor externo. Este complemento solo se puede usar bajo la versión Enterprise.

Teniendo en cuenta estas ejemplos disponibles y otras opciones, se elige el método Key como la implementación que se emplea para la autenticación de los consumidores.

Este método de autenticación es adecuado para el caso de uso de la aplicación porque lo que se pretende es identificar aplicaciones y/o proyectos que hacen uso de los servicios que se ofrecen.

Por lo tanto, con ella se pretende bloquear el tráfico anónimo, controlar la cantidad de llamadas, identificar posibles patrones en el tráfico o filtrar los registros por clave de la API.

De esta manera, descartamos el uso de esta autenticación para identificar usuarios individuales, autorizaciones granulares seguras y/o identificar a los creadores de un proyecto [32]. La configuración del complemento es la siguiente:

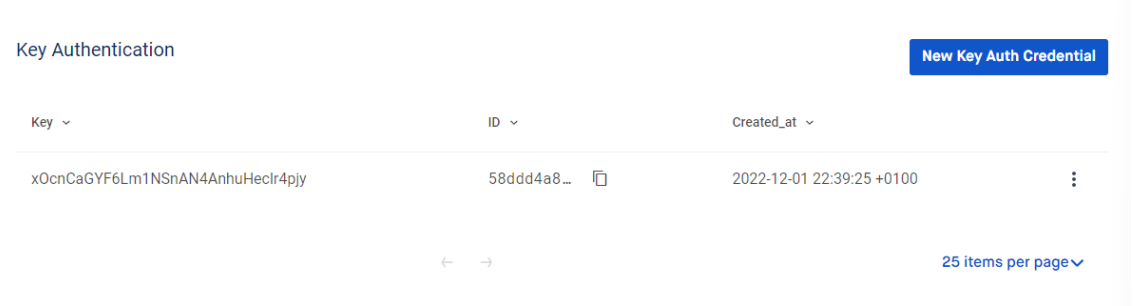


Figura 26: Configuración del complemento de autenticación mediante clave

3.5.2. Lista de control de acceso

La lista de control de acceso es una abstracción que nos permite especificar la capacidad de interactuar de grupos de consumidores sobre los recursos. Este complemento requiere que al menos una implementación de autenticación esté activa.

Se estima óptimo que el servicio de la documentación quede libre a cualquiera pero que, sin embargo, el servicio del API modular requiere, además de estar autenticado, pertenecer al grupo «petstore-application». La configuración en Kong es la siguiente:

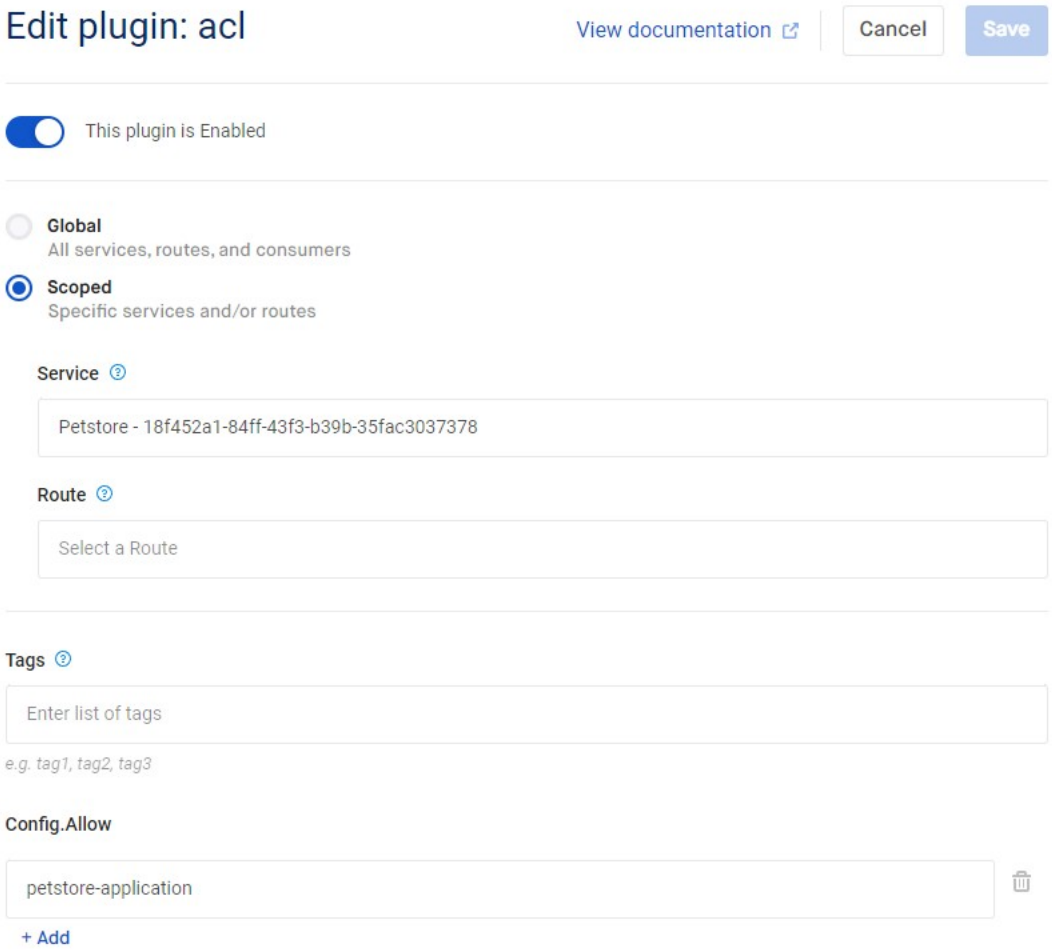


Figura 27: Configuración del complemento ACL

3.5.3. Frecuencia límite

El complemento frecuencia límite de consulta es un mecanismo que permite establecer la cantidad de veces que un recurso puede ser interactuado en un período de tiempo. Se puede establecer a nivel de servicio, ruta o bien por consumidor. La configuración por ACL solo está disponible en la versión Enterprise.

Se estima oportuno, proteger las llamadas que no pueden ser cacheadas en Kong ya que son las que suelen tener el coste más alto. Por este motivo, se configura el complemento a nivel del servicio del API Modular.

Por otra parte, este complemento permite la configuración en diferentes rangos de tiempo a elección del administrador, ya sea de años o bien a nivel meses, días, horas o segundos.

Es importante mencionar que se ha configurado que la información sobre el consumo de los consumidores se almacene en memoria por lo que ante un reinicio del servidor los consumidos estimados a un rango de período de tiempo elevado, como lo son los meses o los años, perderían su efectividad.

También está disponible la opción de persistir esta información en una base de datos. Consideramos que esta es la elección más adecuada de cara a la monetización de los servicios entre otros casos.

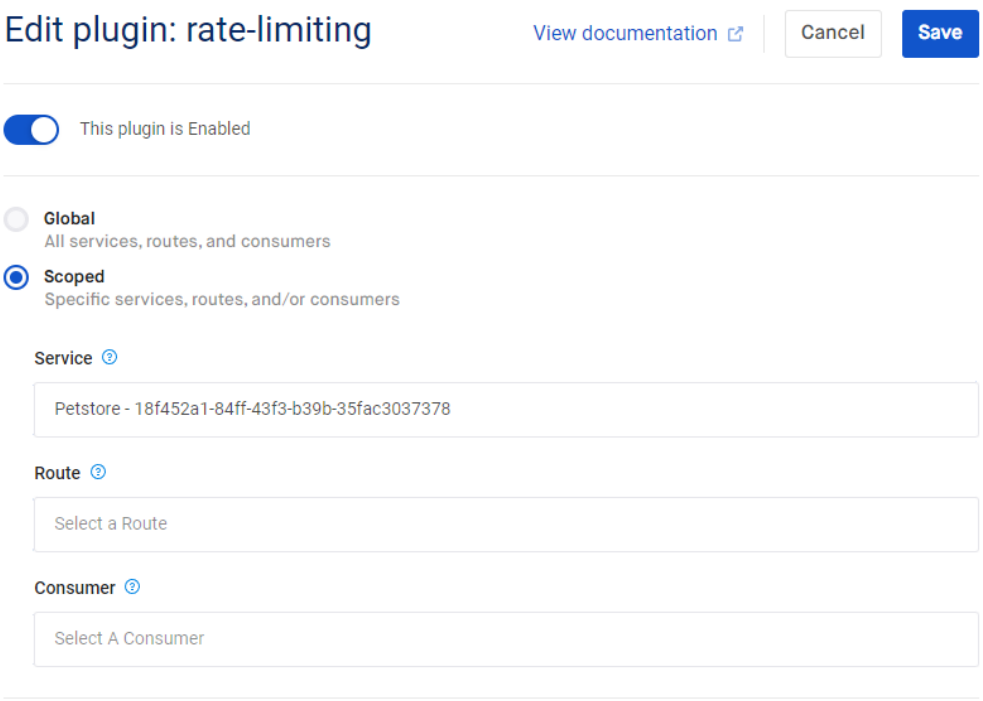


Figura 28: Configuración del complemento de frecuencia de límite

Como límite se establecen 1000 peticiones por hora y 5 cada segundo.

3.5.4. Detección de bots

El complemento de detección de bots es una herramienta que permite proteger los recursos frente a los robots más comunes, tales como Pingdom, Facebook, Google plus o Twitter, entre otros. Se procede con la instalación global:

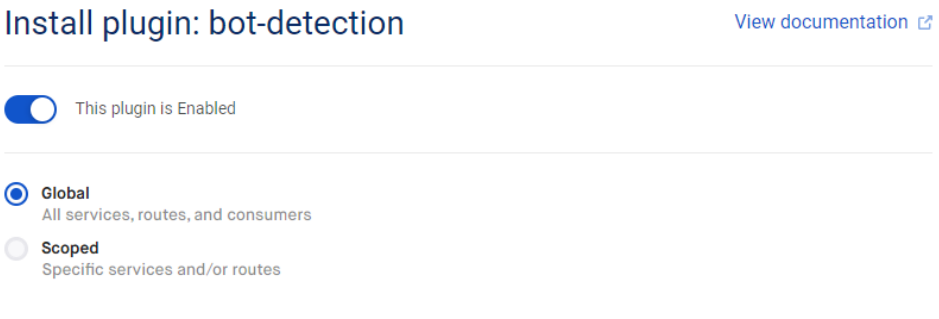


Figura 29: Configuración del complemento de detección de bots

3.5.5. Límite del tamaño de la petición

El complemento de límite de tamaño de la petición es un mecanismo que permite establecer el tamaño en mega-bytes de las peticiones que hacen los consumidores a los recursos que se ofrecen.

Por razones de seguridad, es recomendable habilitar este complemento de manera global dado que previene los ataques de denegación de servicios (DOS).

Por este motivo, configuramos el complemento de manera global. Se establece la propiedad «AllowedPayloadSize» con un valor de 128 para establecer el límite. El resultado final se observa en la siguiente figura:

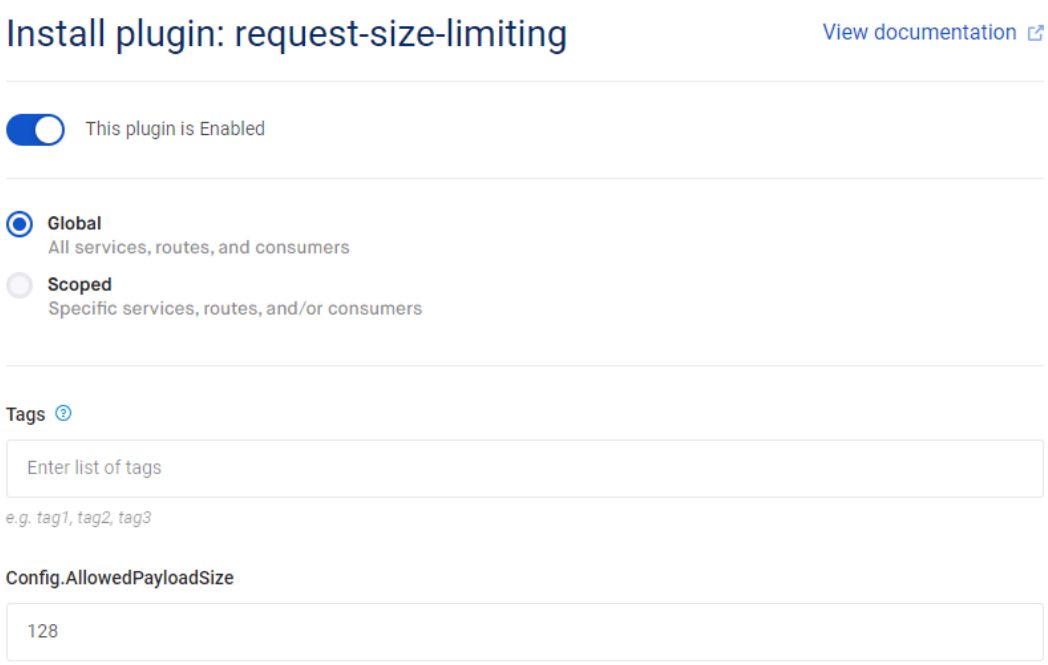


Figura 30: Configuración del complemento de tamaño lítime de petición

3.5.6. Correlación de identificación

El complemento de correlación de identificación permite la trazabilidad entre peticiones y respuestas de los consumidores mediante el uso de un único ID que viaja a través de la cabecera especificada, que por defecto es «Kong-Request-ID».

Dispone de tres maneras de generar el identificador:

- UUID: Genera una hexadecimal UUID en cada petición.
- Contador: Genera un único hexadecimal UUID para cada primera petición y al final de este identificador adjunta el número de la petición asociado.
- Tracker: Genera un identificador formado por la IP y puerto del servidor que acepta la petición, el id del proceso que tiene el servidor, el número de serie de la conexión, el número actual de la petición realizada a través de esa conexión y, por último, la fecha.

Se opta por escoger el generador de identificador en modo contador.

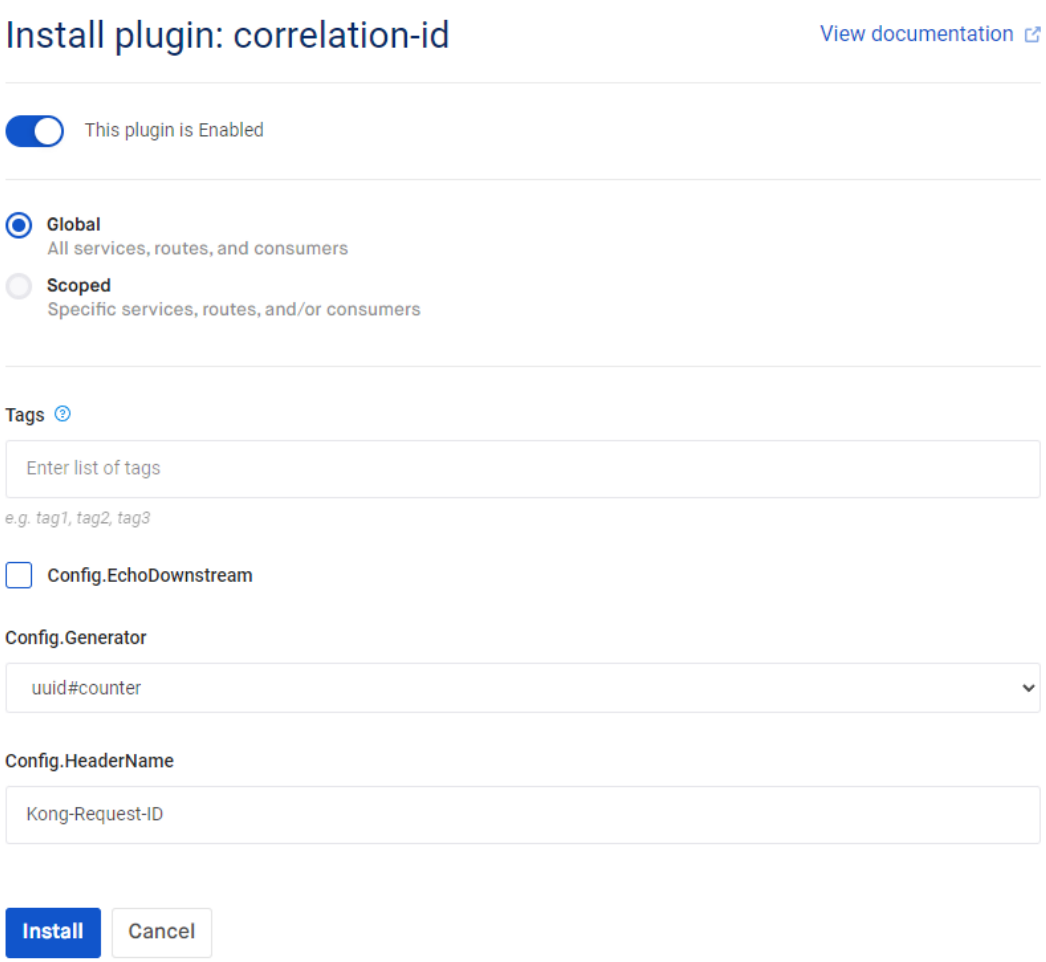


Figura 31: Configuraación del complemento de correlación de identificación

3.6. Configuración de certificado y SNI

3.6.1. Introducción

Las peticiones al API deben realizarse mediante el protocolo HTTPS ya que emplea cifrado TLS en la capa de transporte para las solicitudes. Esto es porque el este protocolo aporta la confidencialidad e integridad a la comunicación entre consumidor y API.

La confidencialidad y la integridad forman parte de la Triada CID [33] en las comunicaciones. La primera garantiza que la información solo pueda ser leída por el destinatario. La segunda garantiza que la información no pueda ser modificada desde que se emite.

Ambas propiedades se consiguen gracias a que el mensaje se envía encriptado mediante clave pública del destinatario, por lo que a su vez el mensaje solo podrá ser leído por él, puesto que dispone también de su clave privada.

Ahora bien, a estas dos propiedades es necesario añadirle la autenticidad para garantizar que los usuarios se comuniquen con entidades que sean las que realmente dicen ser. Para ello se cuenta con organismos que proveen una colección de claves públicas certificadas asociadas a la representación de un sujeto, que son las que se conocen como las autoridades de certificación.

Antes de realizar la configuración partimos con que el certificado publicado que entrega el servidor para esta dominio es un certificado firmado por sí mismo por lo que al no poder garantizar la autenticidad del servidor la petición es bloqueada por motivos de seguridad.

```
webster@webster-pc:~/petstore/ca$ curl https://host.docker.internal:8443 -v && echo
*   Trying 192.168.50.4:8443...
* Connected to host.docker.internal (192.168.50.4) port 8443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  CAfile: /etc/ssl/certs/ca-certificates.crt
*  CApath: /etc/ssl/certs
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS header, Unknown (21):
* TLSv1.3 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: self-signed certificate
* Closing connection 0
curl: (60) SSL certificate problem: self-signed certificate
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
webster@webster-pc:~/petstore/ca$
```

Figura 32: Petición bloqueda por motivos de seguridad. Self-signed

3.6.2. Creación de claves

Procedemos con la creación del certificado siguiendo la guía de Kong sobre creación de certificados [34].

1. Se crea la CA.

```
openssl genrsa -out ca.key 4096
openssl req -new -x509 -days 3650 -key ca.key -out ca.pem
```

2. Se crea la key del host deseado.

```
openssl genrsa -out host.docker.internal.key 2048
```

3. Se crea el certificado de firma de petición (CSR) con la clave del cliente.

```
openssl req -new -key host.docker.internal.key -out kong.lan.csr
```

4. Se firma el certificado con la CA creada.

   a) Se determina la configuración de la firma en el fichero «host.docker.internal.ext».

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage=digitalSignature, nonRepudiation, keyEncipherment,
         dataEncipherment
subjectAltName=@alt_names

[alt_names]
DNS.1 =  host.docker.internal
```

   b) Se crea el certificado del host firmado.

```
openssl x509 -req -in host.docker.internal.csr -CA ca.pem -CAkey \
ca.key -CAcreateserial -out host.docker.internal.pem -days 1825 \
-sha256 -extfile host.docker.internal.ext
```

3.6.3. Configuración de Kong

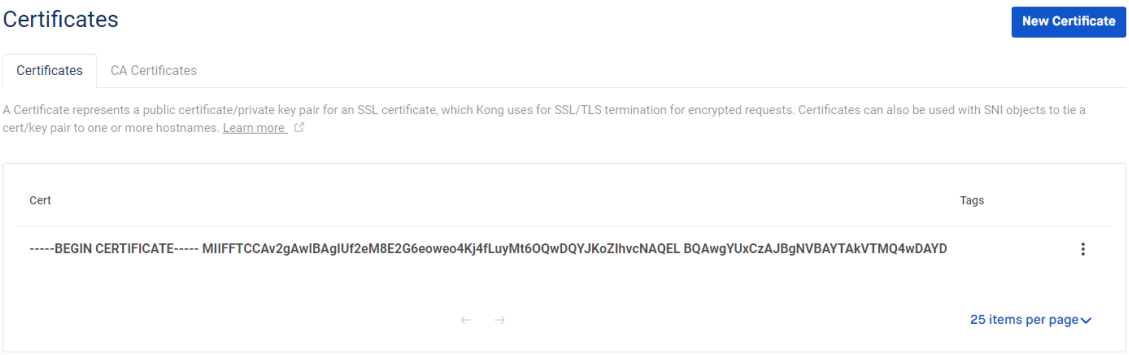


Figura 33: Configuración del certificado del host

Ahora que Kong ya sabe de la existencia del certificado que contiene al host que se ha determinado, le indicamos mediante la configuración de SNI que asocie este host con el certificado creado:

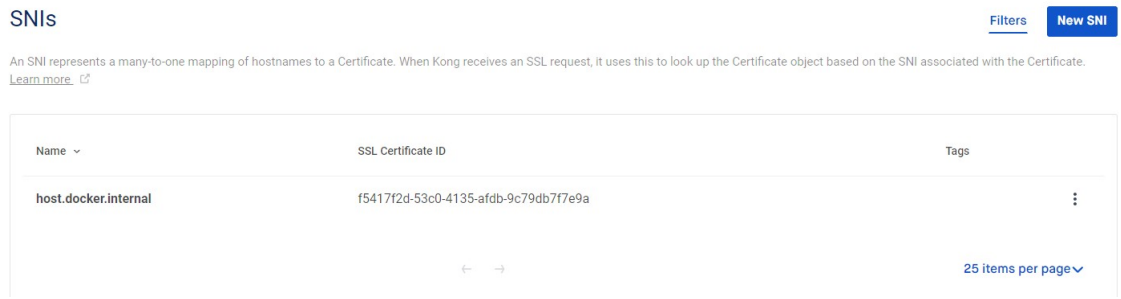


Figura 34: Configuración de la asignación del certificado al host

3.6.4. Configuración de los clientes

Ahora se intenta desde el cliente realizar una petición al API y nos encontramos con que obtenemos un error similar al de la Figura 32, pero esta vez en lugar de obtener un error por certificado auto firmado, se obtiene un error que indica que el certificado de autoridad no es reconocido.

Como veremos, será necesario incluir el certificado de autoridad dentro de los reconocidos por el cliente. Esta tarea se hará de diferentes maneras en función del sistema operativo y del cliente que se esté utilizando para interactuar con el API.

Lo normal, es que este paso adicional no se tenga que realizar dado que los dominios se validan con autoridades de certificación reconocidas tales como Digicert, GlobalSign o Sectigo de tal manera que el certificado público del servidor se acepte sin que se cancele la petición por motivos de seguridad.

```
webster@webster-pc:~/petstore/ca$ curl https://host.docker.internal:8443 -v && echo
*   Trying 192.168.50.4:8443...
* Connected to host.docker.internal (192.168.50.4) port 8443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  CAfile: /etc/ssl/certs/ca-certificates.crt
*  CApath: /etc/ssl/certs
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS header, Unknown (21):
* TLSv1.3 (OUT), TLS alert, certificate unknown (558):
* SSL certificate problem: authority and subject key identifier mismatch
* Closing connection 0
curl: (60) SSL certificate problem: authority and subject key identifier mismatch
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

Figura 35: Petición bloqueda por motivos de seguridad. CA no reconocida.

La instalación de la CA en Ubuntu 22.04 es realmente sencilla:

1. Se convierte servidor del formato «pem» al «crt».

```
openssl x509 -in ca.pem -inform PEM -out ca.crt
```

2. Se crea la carpeta «extra» en el directorio «/usr/local/share/ca-certificates, si es que no existe.

```
sudo mkdir /usr/local/share/ca-certificates/extra
```

3. Se copia la CA en formato «crt» al directorio creado en el paso anterior.

```
sudo cp ca.crt /usr/local/share/ca-certificates/extra/ca.crt
```

4. Se actualizan los certificados del sistema operativo.

```
sudo update-ca-certificates
```

En la siguiente figura se observa la petición exitosa al API mediante el protocolo HTTPS.

```
webster@webster-pc:~/petstore/ca$ curl https://host.docker.internal:8443 -v && echo
*   Trying 192.168.50.4:8443...
* Connected to host.docker.internal (192.168.50.4) port 8443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  CAfile: /etc/ssl/certs/ca-certificates.crt
*  CApath: /etc/ssl/certs
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
*  subject: C=ES; ST=Spain; L=Madrid; O=uoc; OU=University; CN=Webster; emailAddress=wcosmedlr@uoc.edu
*  start date: Nov 26 16:22:11 2022 GMT
*  expire date: Nov 25 16:22:11 2027 GMT
*  subjectAltName: host "host.docker.internal" matched cert's "host.docker.internal"
*  issuer: C=ES; ST=Spain; L=Madrid; O=uoc; OU=University; CN=Webster; emailAddress=wcosmedlr@uoc.edu
*  SSL certificate verify ok.
* Using HTTP2, server supports multiplexing
```

Figura 36: Petición por HTTPS exitosa

En el navegador el certificado también es reconocido como válido:

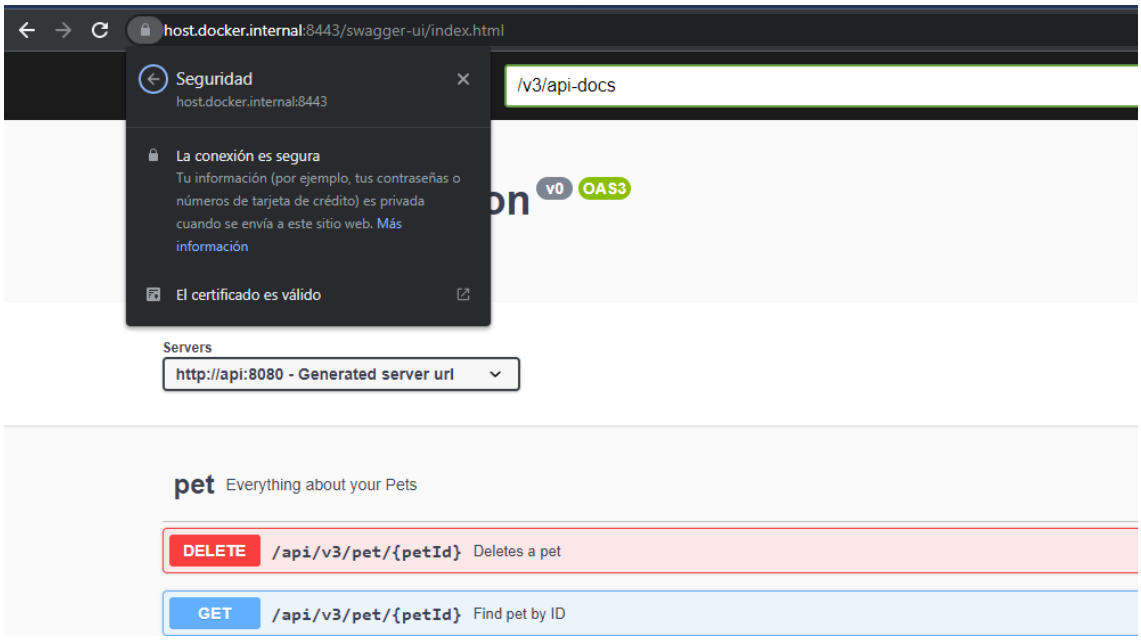


Figura 37: Petición por HTTPS y navegador exitoso

4. Gestión de contenedores

4.1. Introducción

En esta sección, se explica cómo se han gestionado el producto del API tanto como del API modular como el del Gateway. Como se avanzaba en el apartado de Enfoque y método seguido, la infraestructura de la aplicación se basa en el uso de Docker.

El modelo de despliegue que hemos diseñado para trabajar con los productos consiste en la creación de un docker-compose que emplea dos imágenes, una para cada producto.

La imagen del API modular, dado que es un producto diseñado exclusivamente en este proyecto, se creará desde cero. Además se pretende que lo único que se necesite en el equipo host sea la herramienta Docker, evitando el uso de dependencias tales como Maven o Java.

Por otra parte, Kong es una herramienta popular, con un gran soporte a un entorno cloud muestra de ello es que la que imagen del API Gateway ya se encuentra disponible en el repositorio oficial de Docker, por lo que no es necesario que se cree.

En los dos siguientes apartados se detalla cómo se van creando cada una de los servicios que conforman el fichero «docker-compose.yml», el encargado de declarar la infraestructura de este proyecto.

4.2. Configuración del servicio del API Modular

En el directorio «iac/api» sobre la raíz del proyecto se encuentra el fichero Dockerfile, donde definimos el proceso de creación de la imagen y este está compuesto por lo siguiente:

```
FROM eclipse-temurin:17-jdk-jammy
WORKDIR /app/
COPY app/ .
RUN ./mvnw install
CMD ["./mvnw", "spring-boot:run", "-pl", "application"]
```

Se puede observar cómo se copia el contenido de la «app», localización donde se encuentra el código fuente del api modular. A continuación se ejecuta el comando «mvnw install» para generar el ejecutable. Finalmente se declara la manera de inicialización de la imagen que, con el comando «spring-boot:run -pl application», se le indica que ejecuta el módulo de aplicación para disponer de los servicios de usuario, mascotas y pedidos.

Ahora creamos la imagen con el siguiente comando:

```
docker build api/ -t wcosmedlr/api
```

Una vez definida la imagen del API modular entonces se procede con la declaración del servicio en fichero «docker-compose.yml». El resultado obtenido se puede ver en la siguiente tabla.

```
api:
  container_name: api
  network_mode: pet-store-net
  image: 'wcosmedlr/api:latest'
```

En este fragmento, se crea un servicio «api» que coincide con el del nombre del contenedor. La red sobre la cual se ejecutará será «pet-store-net». La imagen que usa es la de «wcosmedlr/api», que es el nombre que le damos a la imagen creada en los pasos indicados anteriormente con con el comando Dockerfile.

Es importante destacar que esta contenedor no ofrece ningún puerto mediante el cual podamos conectarnos. Estos es así a propósito, dado que el acceso a esta imagen queda exclusivamente regulador por el servicio de Kong.

4.3. Configuración del servicio de Kong

Como se comentaba en la introducción de esta sección, tenemos acceso a una imagen del Api Gateway de Kong, por lo que lo único que es necesario es configurar los parámetros adecuados para crear el contenedor.

```
kong-gateway:
  container_name: kong-dbless
  network_mode: pet-store-net
  volumes:
    - ./kong:/kong/declarative/
  environment:
    - KONG_DATABASE=off
    - KONG_DECLARATIVE_CONFIG=/kong/declarative/kong.yml
    - KONG_PROXY_ACCESS_LOG=/dev/stdout
    - KONG_ADMIN_ACCESS_LOG=/dev/stdout
    - KONG_PROXY_ERROR_LOG=/dev/stderr
    - KONG_ADMIN_ERROR_LOG=/dev/stderr
    - 'KONG_ADMIN_LISTEN=0.0.0.0:8001'
    - 'KONG_ADMIN_GUI_URL=*'
  ports:
    - '8000:8000'
    - '8443:8443'
    - '8001:8001'
    - '8444:8444'
    - '8002:8002'
    - '8445:8445'
```

```
    - '8003:8003'
    - '8004:8004'
  image: 'kong/kong-gateway:3.0.1.0'
```

En este fragmento, se crea un servicio «kong-gatway» tiene «kong-dbless» como nombre del contenedor. La red sobre la cual se ejecutará será «pet-store-net» al igual que el servicio del API modular. La imagen que usa es «kong/kong-gateway», obtenida del repositorio oficial de Kong.

Esta contenedor, a diferencia del API modular, sí que expone puertos. De todos ellos los más relevantes son el puerto 8444 que es el que expone los recursos mediante el protocolo HTTPS y el puerto 8002 que es desde donde se puede acceder a la interfaz web de Kong.

Es importante destacar que el contenedor tiene este nombre porque consideramos oportuno que por razones de seguridad este contenedor no tenga base de datos. La información queda almacenada en el fichero «kong.yml» y se carga cada vez que se inicia el contenedor. Este hecho, facilita en gran medida la escalabilidad y por lo tanto la disponibilidad, que es la tercera y última propiedad de la Triada CID que faltaba por hacer referencia en esta memoria.

4.4. Análisis de vulnerabilidad de las imágenes

Es importante asegurarse de que el producto no desplegado no cuenta con vulnerabilidades que pongan en riesgo los servicios que se ofrecen y aún más cuando llega la hora de desplegar el producto en producción.

Por este motivo, es necesario analizar las vulnerabilidades de las imágenes. Es importante que no se confíe en las imágenes que crea el equipo de desarrollo pero aún menos de las imágenes creadas por terceros.

4.4.1. Análisis de la imagen de Kong

Mediante la ejecución del comando «docker scan kong/kong-gateway» analizamos las posibles vulnerabilidades que tendríamos que asumir en caso de hacer uso de esta.

En la siguiente figura, se observa que la imagen de Kong cuenta con dos vulnerabilidades críticas heredades de la imagen «debian:10.13-slim». El comando, en este caso, también nos indica que la imagen «debian::bullseye-20221004-slim» tiene estas vulnerabilidades resuelta.

Por lo tanto, de cara a producción sería muy recomendable basarnos en la imagen propuesta que, entre otras cuestiones, no es más que una versión más reciente del sistema operativo de Debian.

```
✗ High severity vulnerability found in glibc/libc-bin
  Description: Out-of-bounds Write
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN10-GLIBC-559488
  Introduced through: glibc/libc-bin@2.28-10+deb10u2, kong-enterprise-edition@3.1.0.0, meta-common-packages@meta
  From: glibc/libc-bin@2.28-10+deb10u2
  From: kong-enterprise-edition@3.1.0.0 > zlib/zlib1g-dev@1:1.2.11.dfsg-1+deb10u2 > glibc/libc6-dev@2.28-10+deb10u2 > glibc/libc-dev-bin@2.28-10+deb10u2
  From: kong-enterprise-edition@3.1.0.0 > zlib/zlib1g-dev@1:1.2.11.dfsg-1+deb10u2 > glibc/libc6-dev@2.28-10+deb10u2
  and 1 more...
  Image layer: Introduced by your base image (debian:10.13-slim)

✗ High severity vulnerability found in gcc-8/libstdc++6
  Description: Information Exposure
  Info: https://security.snyk.io/vuln/SNYK-DEBIAN10-GCC8-347558
  Introduced through: gcc-8/libstdc++6@8.3.0-6, apt@1.8.2.3, meta-common-packages@meta
  From: gcc-8/libstdc++6@8.3.0-6
  From: apt@1.8.2.3 > gcc-8/libstdc++6@8.3.0-6
  From: apt@1.8.2.3 > apt/libapt-pkg5.0@1.8.2.3 > gcc-8/libstdc++6@8.3.0-6
  and 2 more...
  Image layer: Introduced by your base image (debian:10.13-slim)


Organization:      wcosmedlr-1ch
Package manager:   deb
Project name:      docker-image|kong/kong-gateway
Docker image:      kong/kong-gateway
Platform:          linux/amd64
Base image:        debian:10.13-slim
Licenses:          enabled

Tested 118 dependencies for known issues, found 74 issues.

Base Image         Vulnerabilities  Severity
debian:10.13-slim  57               0 critical, 2 high, 0 medium, 55 low

Recommendations for base image upgrade:

Major upgrades
Base Image                    Vulnerabilities  Severity
debian:bullseye-20221004-slim  44               0 critical, 0 high, 0 medium, 44 low


webster@webster-pc:~/petstore$
```

Figura 38: Vulnerabilidades críticas en la imagen de Kong

4.4.2. Análisis de la imagen del API modular

Al aplicar el mismo comando a la imagen «wcosmedlr/api» que se ha creado encontramos con vulnerabilidades en algunas librerías java de las que se hace uso. El camino a proceder es simular al indicado en el anterior apartado.

```
Testing wcosmedlr/api...

Tested 71 dependencies for known issues, found 9 issues.


Issues to fix by upgrading:

  Upgrade com.fasterxml.jackson.core:jackson-databind@2.13.4 to com.fasterxml.jackson.core:jackson-databind@2.13.4.2 to fix
  ✗ Denial of Service (DoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMFASTERXMLJACKSONCORE-3038426] in com.fasterxml.jackson.core:jackson-databind@2.
13.4
    introduced by com.fasterxml.jackson.core:jackson-databind@2.13.4

  Upgrade org.apache.tomcat.embed:tomcat-embed-core@9.0.65 to org.apache.tomcat.embed:tomcat-embed-core@9.0.68 to fix
  ✗ HTTP Request Smuggling [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGAPACHETOMCATEMBED-3097829] in org.apache.tomcat.embed:tomcat-embed-core@9.0.65
    introduced by org.apache.tomcat.embed:tomcat-embed-core@9.0.65

  Upgrade org.yaml:snakeyaml@1.30 to org.yaml:snakeyaml@1.32 to fix
  ✗ Stack-based Buffer Overflow [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3016888] in org.yaml:snakeyaml@1.30
    introduced by org.yaml:snakeyaml@1.30
  ✗ Stack-based Buffer Overflow (new) [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3113851] in org.yaml:snakeyaml@1.30
    introduced by org.yaml:snakeyaml@1.30
  ✗ Stack-based Buffer Overflow [Low Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3016889] in org.yaml:snakeyaml@1.30
    introduced by org.yaml:snakeyaml@1.30
  ✗ Stack-based Buffer Overflow [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-3016891] in org.yaml:snakeyaml@1.30
    introduced by org.yaml:snakeyaml@1.30
  ✗ Denial of Service (DoS) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-ORGYAML-2806360] in org.yaml:snakeyaml@1.30
    introduced by org.yaml:snakeyaml@1.30


Issues with no direct upgrade or patch:
  ✗ Information Exposure [Medium Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMH2DATABASE-3146851] in com.h2database:h2@2.1.214
    introduced by com.h2database:h2@2.1.214
  No upgrade or patch available
  ✗ Remote Code Execution (RCE) [High Severity][https://security.snyk.io/vuln/SNYK-JAVA-COMH2DATABASE-31685] in com.h2database:h2@2.1.214
    introduced by com.h2database:h2@2.1.214
  No upgrade or patch available


Organization:      wcosmedlr-1ch
Package manager:   maven
Target file:       /app/application/target
Project name:      wcosmedlr/api:latest:/app/application/target
Docker image:      wcosmedlr/api
Licenses:          enabled
```

Figura 39: Vulnerabilidades críticas en la imagen del API modular

5. Conclusiones

5.1. Resultados obtenidos

Con el desarrollo de este proyecto se ha obtenido un API modular basada en Spring Framework que ha sido desplegada en Kong API Gateway. Además, se ha seguido el enfoque, método y planificación marcados desde el inicio.

El producto obtenido nos ofrece acceso a diferentes servicios bajo un contexto de confiabilidad, integridad y disponibilidad, marcado por un amplio conjunto de mecanismos de seguridad.

Por lo tanto, los resultados son los esperados y se puede apreciar que se ha puesto en práctica los conocimientos adquiridos durante el máster junto con los adquiridos en la trayectoria profesional para afrontar este reto.

5.2. Consecución de objetivos

El plan de ejecución del proyecto ha sido la de reducir la problemática de negocio para poner foco en el diseño de una arquitectura y metodología seguras que sirvan de referencia a aquellos que se enfrenten en la vida real a un proyecto similar.

Dado que se ha explicado detalladamente el motivo y la manera de ejecución en cada una de las tareas que se han llevado a cabo durante el desarrollo del proyecto se puede considerar que se ha alcanzado el objetivo principal del proyecto.

5.3. Metodología y planificación

Las claves de la obtención del producto deseado son, en primer lugar, el fiel seguimiento de la metodología y planificación y, en segundo lugar, el contacto cercano con el tutor de este proyecto dado que ha sido fundamental en la obtención de los requisitos y en la resolución de las consultas.

Cabe mencionar que finalmente no se ha optado por emplear el firewall de aplicaciones web sWAF planeado dado que Kong API Gateway ha satisfecho de sobremanera los requisitos del proyecto aún a pesar de que sus funciones estaban limitadas por utilizar la versión gratuita en lugar de la enterprise.

5.4. Impactos previstos

Se espera que reduzca el impacto digital en futuros proyectos dado que siguiendo el enfoque y metodología aquí especificados puede ayudar a mejorar la sostenibilidad.

La seguridad está presente en el crecimiento sostenido y, por ello, cuando se desarrolla un producto hay que tenerla para así no comprometer las necesidades de todos los implicados.

5.5. Líneas de trabajo a futuro

- Admitir múltiples métodos de autenticación: por lo general cada escenario tiene un tipo de autenticación recomendada; en este proyecto se optó por API Token aunque en función del caso se pueden valorar otras como: Autenticación QR, Biometría o Certificado digital.

  El uso de la autenticación multifactor es recomendable. Esta consiste en el empleo de dos medios de autenticación diferentes para hacerla más segura, de manera que cuando una de estas sea superada por un posible atacante no tenga éxito puesto que el segundo factor protege al usuario.

- Resolución de vulnerabilidades detectadas durante el Análisis de vulnerabilidad de las imágenes. En un entorno de producción se deben corregir las vulnerabilidades detectadas con el fin de reducir el potencial impacto de las amenazas.

- Análisis estático de código. Durante el desarrollo del proyecto se ha utilizado escáneres de código con el fin de detectar vulnerabilidades. Este tipo de herramientas analiza la semántica del código mediante reglas predefinidas de tal manera que cuando se detecta una no recomendada lo notifica.

  Ahora bien, este análisis se puede incluir en el flujo de construcción del código de manera que si esta no se supera entonces el código no se despliegue en el entorno de producción.

- Arquitectura Zero Trust. Esta arquitectura consiste en el concepto de seguridad centrado en la creencia de que las organizaciones no deben confiar automáticamente en nada dentro o fuera de sus perímetros. Durante el desarrollo de la práctica se ha implementado esta arquitectura en diferentes áreas, en especial en el API Gateway.

Figura 40: Arquitectura Zero Trust - Píldoras de conocimiento [3]

6. Glosario

- OAS: OpenAPI Specification.
- CRUD: Operaciones de crear, seleccionar, actualizar y eliminar.
- DB: Base de datos.
- WAF: Cortafuegos de aplicaciones web.
- JVM: Java virtual machine.
- API: Interfaz de programación de la aplicación.
- Gateway: Puerta de enlace.
- REST: Transferencia de estado representacional.

7. Bibliografía

7.1. Libros

1. DES RIVIÈRES, Jim. API First. 2005.

7.2. Vídeos

2. INCIBE. 6 de Enero de 2023. Tu Ayuda en Ciberseguridad. https://www.youtube.com/watch?v=TWKvYnz6mL0
3. Autentia. 8 de Enero de 2023. Arquitectura Zero Trust. https://www.youtube.com/watch?v=j25jzpScdkw

7.3. Artículo de revista

4. MUHAMAD, Wardani, et al. Designing semantic web service based on OAS 3.0 through relational database. En *2020 International Conference on Information Technology Systems and Innovation (ICITSI)*. IEEE, 2020. p. 306-311.

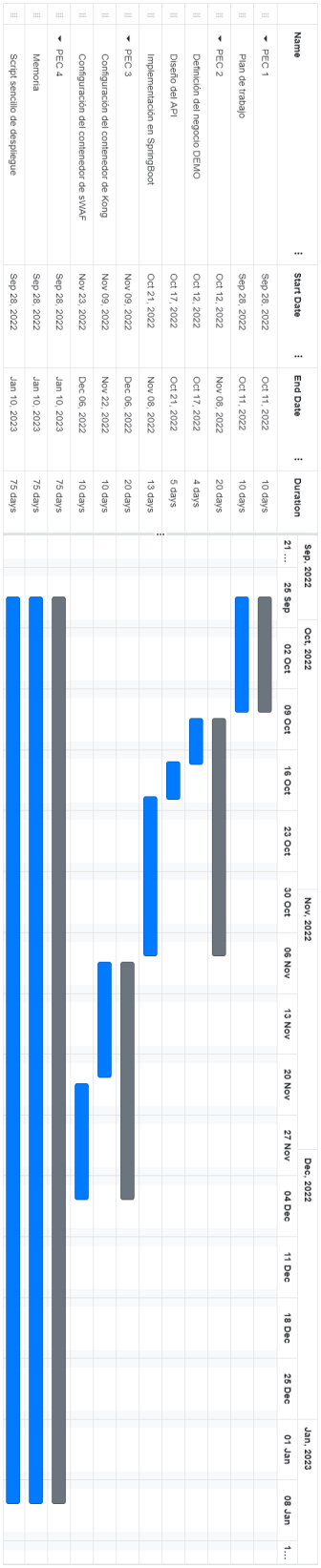
7.4. Páginas web

5. Martin Fowler. 10 de octubre 2022. MonolithFirst. https://martinfowler.com/bliki/MonolithFirst.html
6. Martin Fowler. 10 de octubre 2022. Microservices. https://martinfowler.com/articles/microservices.html
7. OWASP. 10 de octubre 2022. OWASP Top Ten. https://owasp.org/www-project-top-ten/
8. Rezable. 10 de octubre 2022. What is an API Gateway? https://www.reblaze.com/wiki/api-security/what-is-an-api-gateway/
9. Wikipedia. 10 de octubre de 2022. Web application Firewall. https://es.wikipedia.org/wiki/Web_application_firewall
10. UOC. Septiembre de 2022. ¿Cómo incorporar la competencia ”Comportamiento ético y global” al trabajo Final (TF)”?
11. RGPD. 10 de octubre de 2022. Reglamento General de Protección de Datos. https://rgpd.es/
12. AWS. 10 de octubre de 2022. AWS Educate. https://aws.amazon.com/es/education/awseducate/
13. Docker. 10 de octubre de 2022. https://www.docker.com/
14. Spring Framework. 10 de octubre de 2022. https://spring.io/
15. Kong. 10 de octubre de 2022. Community. https://konghq.com/community
16. sWaf. 10 de octubre de 2022. https://swaf-project.github.io/ https://en.wikipedia.org/wiki/Hexagonal_architecture_(software)
17. Wikipedia. 10 de octubre de 2022. https://swaf-project.github.io/ https://en.wikipedia.org/wiki/Hexagonal_architecture_(software)
18. Arnaud Lauret. 5 de noviembre. OpenAPI Map. https://apihandyman.io/toolbox/openapi-map/

19. Apache Maven Project. 6 de noviembre de 2022. Guide to Working with Multiple Modules. https://maven.apache.org/guides/mini/guide-multiple-modules.html
20. Swagger. 6 de noviembre de 2022. PetStore OpenAPI. https://petstore3.swagger.io/api/v3/openapi.json
21. OpenAPI Generator. 6 de noviembre de 2022. complementos.https://openapi-generator.tech/docs/complementos
22. Wikipedia. 6 de noviembre de 2022. Delegation pattern. https://en.wikipedia.org/wiki/Delegation_pattern
23. H2. 6 de noviembre de noviembre de 2022.
24. Spring. 6 de noviembre de 2022. Spring Data JPA. https://spring.io/projects/spring-data-jpa
25. Springdoc. 6 de noviembre de 2022. springdoc-openapi. https://springdoc.org/
26. Kong. 6 de enero de 2023. Enterprise: https://konghq.com/products/api-gateway-platform
27. Kong. 30 de noviembre de 2022. Basic Auth. https://docs.konghq.com/hub/kong-inc/basic-auth/
28. Kong. 30 de noviembre de 2022. HMAC Auth. https://docs.konghq.com/hub/kong-inc/hmac-auth/
29. Kong. 6 de enero de 2023. Api Key. https://docs.konghq.com/hub/kong-inc/key-auth/
30. Kong. LDAP Advanced. https://docs.konghq.com/hub/kong-inc/ldap-auth-advanced/
31. Kong. OAuth 2.0 Introspection. https://docs.konghq.com/hub/kong-inc/oauth2-introspection/
32. Google. 1 de diciembre de 2022 .When to use API Key. https://cloud.google.com/endpoints/docs/openapi/when-why-api-key
33. Wikipedia. 2 de diciembre de 2022. Seguridad de la información. https://es.wikipedia.org/wiki/Seguridad_de_la_informaci%C3%B3n
34. Kong. 2 de diciembre de 2022. How to setup Kong to serve an SSL certificate for API requests. https://support.konghq.com/support/s/article/How-to-setup-Kong-to-serve-an-SSL-certificate-for-API-requests

8. Anexos

8.1. Plan de trabajo completo



Name	Start Date	End Date	Duration
PEC 1	Sep 28, 2022	Oct 11, 2022	10 days
Plan de trabajo	Sep 28, 2022	Oct 11, 2022	10 days
PEC 2	Oct 12, 2022	Nov 08, 2022	20 days
Definición del negocio DEMO	Oct 12, 2022	Oct 17, 2022	4 days
Diseño del API	Oct 17, 2022	Oct 21, 2022	5 days
Implementación en SpringBoot	Oct 21, 2022	Nov 08, 2022	13 days
PEC 3	Nov 09, 2022	Dec 06, 2022	20 days
Configuración del contenedor de Kong	Nov 09, 2022	Nov 22, 2022	10 days
Configuración del contenedor de sWAF	Nov 23, 2022	Dec 06, 2022	10 days
PEC 4	Sep 28, 2022	Jan 10, 2023	75 days
Memoria	Sep 28, 2022	Jan 10, 2023	75 days
Script sencillo de despliegue	Sep 28, 2022	Jan 10, 2023	75 days