ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

DESARROLLO DE GATEWAY DE SEGURIDAD EN PYTHON CON FASTAPI

(DEVELOPMENT OF SECURITY GATEWAY WITH FASTAPI)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Virginia Fernández Palacio

Mayo-2022

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Virginia Fernández Palacio
Director del TFG: Iñaki Goitia de Gea
Título: "Desarrollo de Gateway de Seguridad en Python con FastAPI"
Title: "Development of Security Gateway with FastAPI**"**

Presentado a examen el día: 11-05-2022

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Mª Angeles Quintela Incera
Secretario (Apellidos, Nombre): Luis Francisco Díez Fernández
Vocal (Apellidos, Nombre): Jesús Mirapeix Serrano

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Lista de Acrónimos

ALM: Application Lifecycle Management

API: Application Programming Interface

BKS: BankSphere

DRY: Don`t Repeat Yourself

HTTP: Hypertext Transfer Protocol

IT: Information Technology

JWT: Json Web Token

LDAP: Lightweight Directory Access Protocol

MVC: Modelo-Vista-Controlador

PAAS: Platform as a Service

POC: Proof of Concept

SCG: Spring Cloud Gateway

SOAP: Simple Object Access Protocol

SSO: Single Sign-On

STS: Security Token Service

XSLT: eXtensible Stylesheet Language for Transformations

Índice de Contenidos

Capítulo 1.

Prólogo

Resumen

Este Proyecto de Fin de Grado se desarrolla en un contexto de renovación de la Tecnología en el ámbito financiero, pertenece a un proyecto enorme de migración del Mainframe y su objetivo es actualizar la herramienta que se usa para aportar Seguridad a las peticiones HTTP, modernizándola para su adaptación a la Arquitectura de Microservicios y añadiendo la innovación del uso de Python, que no es el lenguaje con el que se desarrollan estas herramientas de forma habitual.

Como tarea principal, nos encontramos con un profundo análisis de las librerías del desarrollo bajo el Framework Django, para después documentarlo y comenzar con su desarrollo bajo el Framework FastAPI, con el objetivo de analizarlo y tomar la decisión de si es más efectivo y viable que su compañero, de esta manera se incluirá en el Stack Tecnológico de Arquitectura de Microservicios para posteriormente, ser homologado.

Palabras Clave: Gateway, Seguridad, API, Microservicios, Token.

Abstract

This end-of-degree Project has been developed in a context of Technology renewal in financial services. It is part of a larger migration Project and its objectives are: to update the tool to provide security for http Requests, modernise it, adapt it to the Microservices architecture and introduce the use of Python, which is a novel Programming Language for these types of Projects.

The main task deeply analyses and documents the libraries of the existing Project under Django Framework and compares it against the same Project under the FastAPI Framework. This analysis leads to determine which solution is more effective and suitable for the Microservices architecture with the hope for an eventual inclusión in the internal approved frameworks.

Key Words: Gateway, Security, API, Microservices, Token.

Capítulo 2.

Introducción

2.1 Contexto

Nos situamos en el área de las Telecomunicaciones, dentro de los Protocolos utilizados para Comunicaciones de datos en las Aplicaciones Web.

Actualmente, se han popularizado estas comunicaciones por medio de API Rest y han dejado un poco de lado el protocolo SOAP. En este proyecto estudiaremos como se implementa la Seguridad en este tipo de comunicaciones, utilizando una puerta de enlace y enmarcando todo en el desarrollo de un Proyecto innovador a gran escala del que hablaremos en el siguiente apartado.

2.2. Marco Tecnológico

2.2.1 Proyecto central

Antes de meternos de lleno a explicar el desarrollo del Gateway de Seguridad, vamos a contextualizarlo. Todo comienza con un gran proyecto de renovación del Core Banking llamado Gravity, la revolución en la tecnología usada en el ámbito financiero en Grupo Santander, lo que sugiere un proyecto gigante donde entran en juego muchas infraestructuras y herramientas tecnológicas.

Su objetivo es salir de una solución monolítica para ir hacia tecnologías abiertas e intercambiables, de manera que los datos puedan ser accesibles para cualquier aplicación distribuida. Es por ello que se pretende incrementar la capacidad de recuperarse ante cualquier problema, separando modos de ejecución e instaurando herramientas de inteligencia artificial que puedan tener un control total sobre todo lo que ocurre en los sistemas.

Para llevar a cabo esta renovación se necesitan dos componentes: los Partners Tecnológicos y un equipo líder.

Los Partners Tecnológicos, con la potencia que proveen sus infraestructuras, ayudarán al desarrollo y mantenimiento de las nuevas tecnologías. Por otro lado, el equipo líder consistirá en un grupo de personas con un conocimiento profundo en tecnología bancaria.

¿Por qué esta transformación?

- Porque ayudará a reducir los costes anuales hasta un 60 %
- Resiliencia, gracias a los nodos distribuidos.
- Traer los datos al Front, para acelerar Deliveries, Agility y Cost Efficiency.
- Escalabilidad, al estar basado en la infraestructura de la nube.
- Para las APIs permite interconexión con nuevas plataformas y terceras partes.
- El desarrollo con DevOps aportará facilidad de mantenimiento.

Sin duda, el reto más desafiante ante el que se encuentra este proyecto es la migración del Core a la nube pero, ¿Cómo se lleva a cabo?

1. Evaluación
2. Estimación
3. Habilitación Técnica
4. Migración
5. Certificación y Ejecución
6. Dual Run
7. Monitoreo

Esta migración del Mainframe a la nube se realiza dentro de cuatro bloques con equipos Agile altamente cualificados:

PLATFORM

Este bloque nos proveerá de un alto rendimiento, alta escalabilidad y una arquitectura totalmente integrada construida sobre arquitecturas técnicas de referencia con soluciones líderes en el mercado.

MIGRATION

A este bloque lo componen un conjunto de herramientas para el desarrollo de Software y tipos de datos con el fin de garantizar la automatización de procesos de migración.

OPERATIONS

Gravity es un nuevo modelo tecnológico que incorpora su propio Modelo Operacional.

CERTIFICATION

Herramientas customizadas para la automatización e implementación de pruebas de desarrollo, con objetivo de replicar, monitorear y sincronizar mecanismos end-to-end.

Este proyecto se está llevando a cabo de manera global, con él se pretende crecer de manera dinámica y gradual, y escribir historia, para dejar la tecnología ya implementada a los que vengan posteriormente.

2.2.2. Gateway de Seguridad

¿Qué lugar ocupa nuestro Gateway de Seguridad dentro de Gravity?

Este proyecto está ubicado dentro del Desarrollo de Software, siendo documentado y utilizado como una pieza de Arquitectura que aporta Seguridad en la capa de Exposición a la hora de recibir peticiones de Servicios Externos, ya que tendremos dos posibles orígenes de las peticiones, los usuarios corporativos, que podrán acceder con sus credenciales pasando por la capa LDAP que les otorgara el acceso, y obteniendo el JWT a partir de su Token corporativo BKS, y por otra parte las peticiones con origen en Servicios Externos, a los que se les proveerá de unos credenciales genéricos que deberán figurar también en la capa LDAP para dar acceso, en este caso se les proveerá directamente de un JWT. De esta manera los usuarios podrán acceder a consumir el Microservicio solicitado.

Capítulo 3.

Arquitectura Actual

3.1 APIs y Microservicios

Como hemos comentado en la introducción, se está dejando de lado el protocolo SOAP para utilizar cada vez más el estilo de Arquitectura de desarrollo web APIRest. Ambos ofrecen funcionalidades para varios casos de uso con API y Servicios Web por lo que vamos a ver una tabla comparativa a continuación pata entender mejor que les diferencia.

SOAP (Simple Object Access Protocol)	**REST (REpresentational State Transfer)**
Protocol	Architecture
Function Driven	Data driven
Requires Advanced Security, but can define it too	Relies on underlying network which can be less secure
Needs more bandwidth	Only needs minimum bandwidth
Stricter rules to follow	Easier for developers to suggest recommendations
Cannot use REST	Can use SOAP
Only works in XML	Works in different data formats such as HTML, JSON, XML, and plain text.
Supports HTTP and SMPT protocols	Only requires HTTP

Figura 1. Tabla comparativa SOAP-REST

Pero no solo se está pasando de utilizar el protocolo SOAP para empezar a implementar APIRest, también se está implementando, cada vez más, la Arquitectura de Microservicios, que trabajan conjuntamente con las API en la mayoría de las implementaciones, como es nuestro caso. Muchas veces se pueden confundir APIs y Microservicios, ¿Cuál es su diferencia?

Microservicios: técnica específica para desarrollar sistemas software.

Permite a las empresas ser más flexibles y escalables, entre sus ventajas están:

- Fáciles de implementar (no se alteran otros servicios)
- Fácil acceso (reutilización)
- Respuesta rápida
- Identificación y resolución rápida ante problemas
- Acceso a soluciones de las API (problema servicios heredados)

API: conjunto de definiciones y protocolos que se utiliza para integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones.

Separa la aplicación principal de la infraestructura que brinda un servicio al cliente, sus ventajas:

- Flexibilidad y funcionalidad entre sistemas
- Medio de comunicación entre Microservicios
- Exposición a terceros

Es por esto que podemos decir que las API son una parte de los Microservicios y que aportan la interacción con los mismos, dejando atrás las antiguas soluciones monolíticas y encaminándose hacia estructuras modernas.

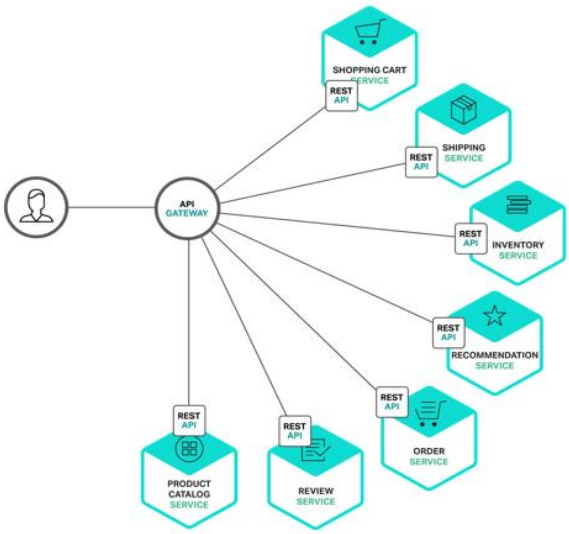


Figura 2. Arquitectura de Microservicios

3.1.1 Gateway de Seguridad

Con este Gateway trabajamos en los complementos de la Arquitectura de Microservicios, implementando nuestra puerta de enlace de seguridad como uno de ellos. Una vez vista su Arquitectura, podemos observar el siguiente diagrama donde se muestra la complementación y forma de trabajo de la puerta de enlace como punto intermedio entre las APIS y Microservicios.

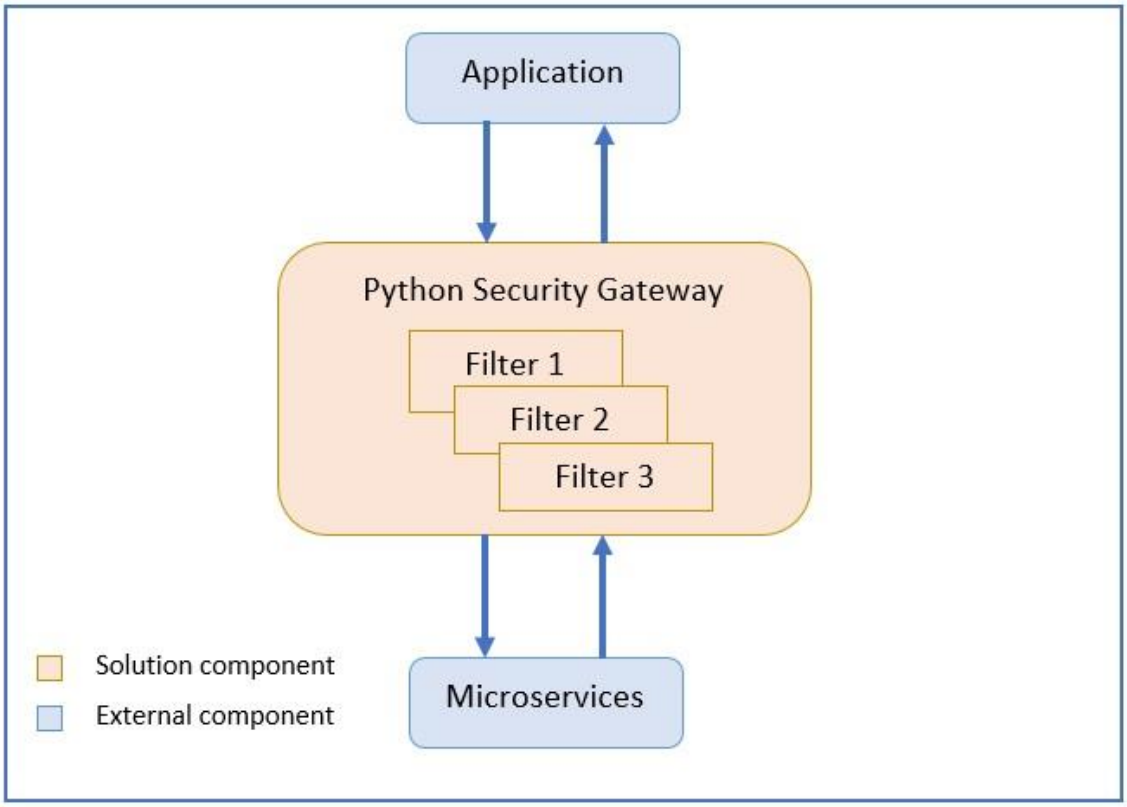


Figura 3. Diagrama Gateway de Seguridad

El proyecto de la puerta de enlace está compuesto por dos librerías programadas en Python que deben ser configuradas para poder usarse en otros proyectos al ser importadas. Estas, consisten en la implementación de filtros de seguridad para poder reenviar la solicitud a los Microservicios además de proveernos del enrutamiento y la estructura de una API Gateway con la herramienta SCG, para así poder consumirlos. Más adelante se explicará detalladamente el desarrollo y funcionalidad de ambas.

A día de hoy, existen infinidad de librerías de diferentes tipos (Machine Learning, Visualización, Inteligencia Artificial, etc.), cada una de acuerdo al trabajo a realizar y con el objetivo de ayudar en el Desarrollo de Proyectos de Programación.

Nuestras librerías, pertenecen al área de Seguridad y su función será integrar soluciones que habían funcionado hasta ahora con estructuras estandarizadas como son los JSON Web Tokens que veremos a continuación, además de ofrecer la posibilidad de trabajar con herramientas adicionales como son Spring Cloud Gateway y sus Fábricas de Predicados.

Capítulo 4

El Token como herramienta principal del Desarrollo

4.1 Introducción a los Token

Si buscamos hoy en día la definición de Token lo primero que encontraremos es que hace referencia a un objeto que representa otra cosa, como otro objeto, físico o virtual, o un concepto abstracto.

En este proyecto no estamos haciendo referencia a ese tipo de Token tan popular en las Criptomonedas, sino a los Token de Seguridad para Autenticación de Usuarios, una cadena de caracteres que provee cierta información y que nos ayuda a trasmitir los datos de forma segura, por lo que no debemos confundirlos con los Token utilizados en la Arquitectura distribuida Blockchain.

En el desarrollo de este Gateway de Seguridad, el filtro más importante y que será ejecutado en cualquier Solicitud HTTP, consiste en un análisis y validación de Tokens JWT para proporcionar el Inicio de sesión único del que habíamos hablado con anterioridad, por lo que haremos especial hincapié en la explicación de los mismos. Hablaremos de dos tipos de Token de autenticación, los JWT, el estándar actual, y de los BKS, la estructura corporativa, ya que nuestro Gateway se ocupará de recibir un BKS y devolvernos un JWT para poder llevar a cabo esa comunicación con los Microservicios, que funcionan con el estándar.

4.2. JWT

JWT (JSON Web Tokens), es un estándar abierto RFC 7519 que define una forma compacta y autónoma de transmitir información de forma segura.

Resuelven el problema de la estandarización al transferir información entre partes y aunque podemos encontrar implementaciones pasadas tanto de forma pública como privada, los JWT nos proporcionan un formato simple y útil.

Utiliza una serie de afirmaciones que pueden definirse como parte de la especificación JWT o por el usuario y además, hace uso de la firma digital para hacer la información confiable y verificable. Para ello, pueden ser firmados usando una clave secreta con el algoritmo HMAC, o usando el par clave pública/ clave privada con el algoritmo RSA o ECDSA. Además, también pueden ser cifrados para proporcionar confidencialidad entre las partes. Hablaremos de estos algoritmos de firma y de cifrado más adelante.

Para trabajar con este estándar, disponemos de herramientas interactivas muy útiles donde podemos codificar y decodificar JWT como es https://jwt.io/ y que nos proporcionará ejemplos para este apartado 3.1.

Estructura

Los JWT, se construyen a partir de tres elementos diferentes: encabezado, carga útil y datos de firma/cifrado.

Los dos primeros elementos son objetos Json con cierta estructura mientras que el tercero depende del algoritmo utilizado para la firma o el cifrado, y en el caso de los no cifrados, se omite.

Utilizando la herramienta jwt.io citada con anterioridad, mostraremos un ejemplo de JWT, codificado y decodificado.

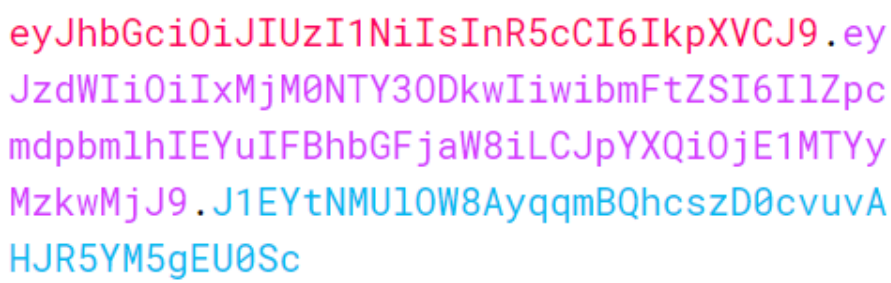


Figura 4. Ejemplo de JWT codificado

Figura 5. Ejemplo de JWT decodificado

Encabezado

En el encabezado, los JWT llevan afirmaciones sobre sí mismos, entre ellas se establecen, los algoritmos utilizados, si el JWT está firmado o encriptado y como se analiza en general el resto del JWT. Según si este JWT se transmite cifrado o no, puede variar el número de campos en la cabecera.

Para un encabezado JWT sin cifrar el único campo obligatorio será 'alg' y llevará el valor 'None'. En caso contrario:

alg: algoritmo principal en uso para firmar y/o descifrar este JWT.

En los campos opcionales se incluyen:

typ: el tipo del propio JWT. Solo debe usarse como ayuda cuando los JWT pueden mezclarse con otros objetos que llevan el mismo tipo de encabezado. Su valor debe establecerse como 'JWT'.

cty: tipo de contenido. La mayoría de los JWT llevan campos específicos más datos arbitrarios como parte de la carga útil, en ese caso no se debe establecer el tipo de contenido.

También es posible añadir campos adicionales definidos por el usuario.

Fijémonos en el ejemplo:


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```


De él podemos saber que el algoritmo utilizado es HS256, lo que es lo mismo que decir que ha sido firmado con HMAC-SHA256; y que es de tipo JWT.

Carga Útil

El Payload es el elemento donde se incluyen los datos del usuario, al igual que el encabezado es un objeto Json. Ningún campo es obligatorio, el estándar JWT especifica que se deben ignorar los campos que no son entendidos por una implementación. Los campos con significados específicos se conocen como Reclamos Registrados y los nombraremos a continuación.

iss: proviene de la palabra emisor y hace referencia a una cadena o URI y que identifica de forma única a la parte que emitió el JWT.

sub: de la palabra subject y que identifica de forma única a la parte sobre la que este JWT transporta información.

aud: de la palabra audience, hace referencia a una cadena o URI o matriz de valores que identifican de forma única a los destinatarios previsto del JWT.

exp: de la palabra expiration, es un número que representa una fecha y hora específicas en el formato “seconds since epoch” definido por POSIX y establece el momento exacto desde que este JWT se considera inválido.

nbf: proviene de “not Before” (tiempo) y es lo contrario que el reclamo exp, se establece el momento exacto a partir del cual se considera válido el JWT.

iat: proviene de “issued at” (tiempo) y hace referencia a una fecha y hora específicas en que se emitió el JWT.

jti: proviene de JWT Id y hace referencia a una cadena que representa un identificador único para este JWT. Se puede usar para diferenciar JWT con contenido similar.

También podemos encontrarnos con otro tipo de reclamos, los privados y los públicos. Los primeros son definidos por los usuarios que consumen y producen los JWT, mientras que los segundos están registrados en el registro de reclamos de Json Web Token de la IANA.

Fijándonos en el ejemplo:

```
{
  "sub": "1234567890",
  "name": "Virginia F. Palacio",
  "iat": 1516239022
}
```

Podemos observar dos reclamos registrados y otro público registrado en la IANA.

Firma/cifrado

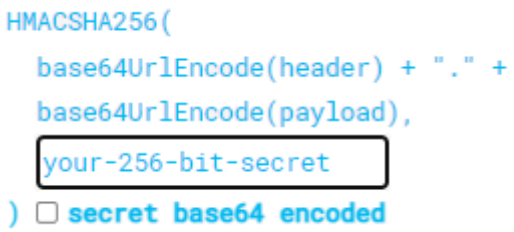
Las firmas web Json son probablemente la característica más útil de los JWT. Al combinar un formato de datos simple con una serie bien definida de algoritmos de firma, se están convirtiendo rápidamente en el formato ideal para compartir datos de forma segura entre clientes e intermediarios.

Su propósito es permitir que una o más partes establezcan la autenticidad del JWT, que, en este contexto, significa que los datos contenidos en el JWT no han sido alterados. Esto quiere decir que cualquier parte que pueda hacer una verificación de firma, puede confiar en los contenidos proporcionados por el JWT. Es importante añadir, que una firma no impide que otras partes lean el contenido dentro del JWT, que es lo que se supone que debe hacer el cifrado.

El proceso de verificar la firma de un JWT se conoce como validación. Un Token se considera válido cuando se cumplen todas las restricciones especificadas en su encabezado y carga útil.

Un JWT puede considerarse válido si carece de firma (“alg”=None), y también puede considerarse no válido llevando firma por otros aspectos, como haber caducado.

Fijándonos en el ejemplo:



En este caso el Token se firma con el algoritmo HS256 (HMAC-SHA256), que es el más común y vamos a explicarle con más detenimiento.

HMAC-SHA256

HMAC, es un algoritmo que combina una determinada carga útil con una clave mediante una función hash criptográfica. El resultado es un código que se puede usar para verificar un mensaje solo si tanto la parte que lo genera como la que lo verifica conocen la clave, es decir, los HMAC permiten que los mensajes se verifiquen a través de claves compartidas.

La función hash criptográfica utilizada en HS256 es SHA-256 que vamos a ver con detenimiento enseguida pero antes hagamos una breve introducción a lo que es la función hash.

Esta función toma un mensaje de longitud arbitraria y produce una salida de longitud fija. El mismo mensaje siempre producirá el mismo resultado, además la parte criptográfica de una función hash asegura que sea matemáticamente inviable recuperar el mensaje original a la salida de la función, por lo que son funciones unidireccionales que se pueden usar para identificar mensajes sin compartir el mensaje.

Algoritmos Generales

Base64-URL

Este algoritmo soluciona el problema de los dos caracteres problemáticos en el algoritmo Base64, los caracteres “+” y “/” se reemplazan por “-“ y “_”. Se trata de un algoritmo de codificación de binario a texto, su estándar es RFC 46481. Para comprender mejor como puede convertir los octetos de bits en texto podemos observar la siguiente imagen.

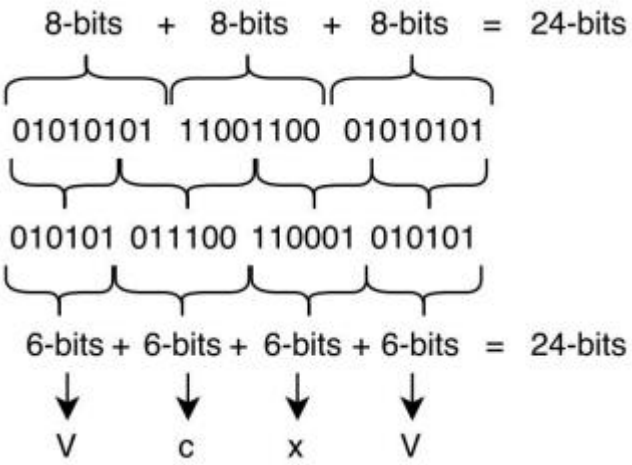


Figura 6. Codificación Base64

Si el número de octetos en los datos de entrada no es divisible por tres, entonces la última parte de datos a codificar tendrá menos de 24 bits de datos. Cuando este es el caso, se agregan ceros a los datos de entrada concatenados para formar un número entero de grupos de 6 bits. Hay tres posibilidades:

1. Que los 24 bits completos están disponibles como entrada: no se realiza ningún procesamiento especial.
2. Que haya 16 bits de entrada disponibles: se forman tres valores de 6 bits y al último valor de 6 bits se le agregan ceros adicionales a la derecha. La cadena codificada resultante se rellena con un carácter = extra para que quede explícito que faltaban 8 bits de entrada.
3. Que haya 8 bits de entrada disponibles: se forman dos valores de 6 bits y al último valor de 6 bits se le agregan ceros adicionales a la derecha. La cadena codificada resultante se rellena con dos caracteres = adicionales para que quede explícito que faltaban 16 bits de entrada.

SHA

Es el algoritmo de hash seguro utilizado en las especificaciones JWT, siendo importantes SHA-256 y SHA-512.

Este algoritmo funciona procesando la entrada en fragmentos de tamaño fijo, aplicando una serie de operaciones matemáticas y luego acumula el resultado realizando una operación con los resultados de la iteración anterior. Una vez procesados todos los fragmentos, se dice que se calcula el resumen.

Esta familia de algoritmos se diseñó para evitar colisiones y producir salidas radicalmente diferentes, incluso cuando la entrada solo cambia ligeramente, es por ello que se consideran seguros.

Este algoritmo requiere de una serie de funciones predefinidas, que veremos a continuación, pero se definen en la especificación.

```
function rotr(x, n) { return (x
      >>> n) | (x << (32 - n));
}

function ch(x, y, z) { return (x
      & y) ^ ((~x) & z);
}

función maj (x, y, z) {
      volver (x & y) ^ (x & z) ^ (y & z);
}

function bsig0(x) { return
      rotr(x, 2) ^ rotr(x, 13) ^ rotr(x, 22);
}

function bsig1(x) { return
      rotr(x, 6) ^ rotr(x, 11) ^ rotr(x, 25);
}

function ssig0(x) { return
      rotr(x, 7) ^ rotr(x, 18) ^ (x >>> 3);
}

function ssig1(x) { return
      rotr(x, 17) ^ rotr(x, 19) ^ (x >>> 10);
}
```

Figura 7. Funciones del Algoritmo RSA

Algoritmos de Firma

HMAC

Un esquema de autenticación basado en HMAC toma una función hash, un mensaje y una clave secreta como entradas y produce un código de autenticación como salida. El mensaje no se puede modificar sin la clave secreta.

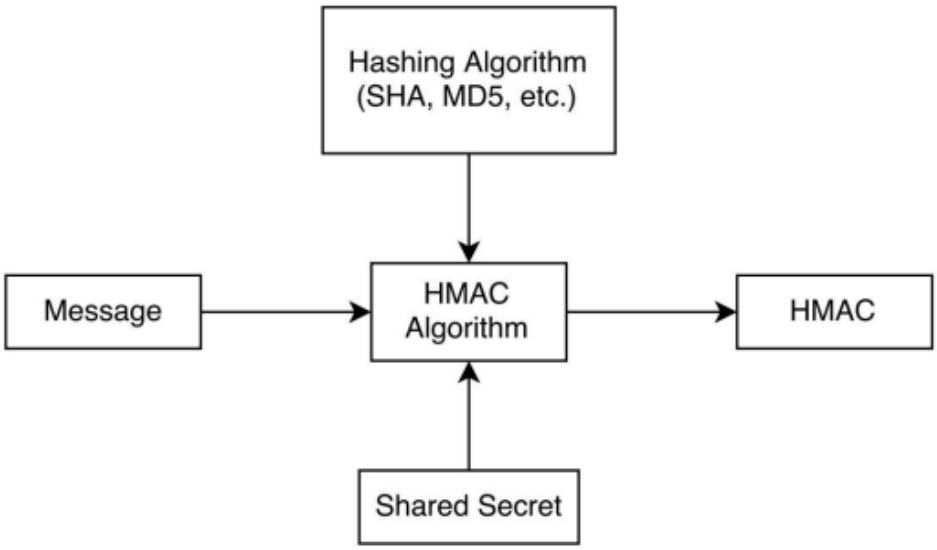


Figura 8. HMAC

HMAC junto con SHA256 proporciona el algoritmo de firma HS256.

RSA

Es un algoritmo de los más utilizados en la actualidad, su aspecto clave es su asimetría, es decir, la clave utilizada para cifrar algo no es utilizada para descifrarlo. Se conoce como cifrado de clave pública (PKI), donde la clave pública es la clave de cifrado y la clave privada la de descifrado, pero cuando se trata de firmas, la clave privada se usa para firmar y la pública para verificar que fue firmada por una clave específica.

Existen variaciones de este algoritmo tanto para la firma como para el cifrado.

Curva Elíptica

Los algoritmos de curva elíptica, al igual que RSA, se basan en un conjunto de problemas matemáticos intratables, mientras que RSA se basa en la dificultad del problema de factorización24, encontrar los factores primos de un gran número coprimo, los algoritmos de curva elíptica se basan en la dificultad del problema del logaritmo discreto de curva elíptica. Las curvas elípticas se describen mediante la siguiente ecuación:

$$y^2 = x^3 + ax + b$$

Funcionamiento de los JWT

En la fase de la autenticación, trabajaremos con un flujo básico en Oauth Server, donde el usuario inicia sesión correctamente con sus credenciales en un Front y Oauth Server devuelve un JWT. Con este JWT accederemos a la API y de ahí al Microservicio.

Puesto que los Tokens no son credenciales, se debe tener mucho cuidado para evitar problemas de seguridad, por lo que no debemos conservar los Tokens más tiempo del necesario.

Cada vez que el usuario desee acceder a una ruta o recurso protegido, el agente de usuario debe enviar el JWT, generalmente en el encabezado de Autorización utilizando el prefijo Bearer. Para ello, se debe tener en cuenta que si enviamos Tokens JWT a través de encabezados HTTP, hay que evitar que crezcan demasiado, ya que algunos servidores no aceptan más de 8 KB en encabezados, en ese caso es posible buscar una solución alternativa como Auth0.

4.3. BKS

Los Token BKS tienen su origen en Banksphere, que es un entorno completo usado en el desarrollo y ejecución de aplicaciones distribuidas para el mundo web, basado en Java Enterprise Edition o Java EE.

Sigue un modelo de tres niveles que diferencia claramente las capas de presentación, lógica de negocio y modelo de datos. Este entorno participa en todas las fases del ciclo de vida de un proyecto de desarrollo, proporcionando:

- Una metodología de trabajo.
- Un conjunto de herramientas para el diseño, creación y mantenimiento de los diferentes componentes, permitiendo su gestión desde el concepto hasta la producción.
- Un entorno de ejecución gestionado, basado en servicios.

Pero, ¿Cómo se crean los Token BKS?

Se utiliza XSLT (Lenguaje que permite transformar un documento XML a otros formatos), tiene varios módulos diseñados específicamente para generar, cifrar, descifrar, firmar y verificar BKS. El código, accede a los archivos de configuración para obtener algunas propiedades necesarias para generar el Token BKS correctamente.

En este archivo de configuración configFile_TokenBKS.xml se pueden seleccionar las siguientes propiedades:

KeyObject: contiene el nombre del KeyObject involucrado que contiene la clave privada, para firmar el Token.

CertObject: contiene la clave pública para verificar la firma del Token.

ET: contiene el emisor del Token usado para la generación del Token.

Ejemplo de BKS

"MEE3NTFGRTQxNEYyNTAwREIxNkRCOUZ-
DIzE4MC4xMDEuMTE2LjkjMTYxNzg4OTc0NTMzNyNQRDk0Yld3Z2RtVnljM
mx2YmowaU1TNHdJaUJsYm1Od-
lpHbHVaejBpU1ZOUExUZzROVGt0TVNJL1BqeDBiMnRsYmtSbFptbHVhWFJ
wYjI0K1BHeHZZMkZzUlcxcGRIUmxjajVKYm5SeVlWTkRRa0ZzWlcxa-
GJtbGhQQzlzYjJOaGJFVnRhWFIwWlhJK1BIVnpaWEpKUkQ1dU56QXd-
NekE4TDNWelpYSkpSRDQ4Ym1GdFpUNUZjbWxyWVNCU2IyMWhiaUJO-
Wlc1a2Iz-
cGhJRHd2Ym1GdFpUNDhZV3hwWVhNK2JqY3dNRE13UEM5aGJHbGhjejQ4
ZFhObGNrTnZjbkErYmpjd01ETXdQQzkxYzJWeVEyOXljRDQ4TDNS-

dmEyVnVSR1ZtYVc1cGRHbHZiajQ9I0RFU2VkZS9DQkMvUEtDUzVQYWR-
kaW5nI3YxI0NvcnBJbnRyYW5ldCNOT1QgVVNFRCNTSEExd2l0aFJTQS-
NiNm4rV3FnaUUrcFRDTFlZNDNpM240c1Bibjg1T2krcExDQUJMK0NBUD-
ZUbXFPY09pSkVEclhxTnJCeTFudXpJN0hsT-
VlhcVcvdDRucmV5cE90b0M4UU1sUFZna1h0NTljUTRLTGhVYzY2Q1VLQXh-
sOXhpM3RvNlpQMmxENUU5UWZLS2JaK2NlZFhwY1VBdFJMe-
jNISzMyOEhKUzQzLy9vaWc2RHptRjVHcmc9"

Este ejemplo, como podemos observar, nos muestra una estructura muy parecida a la que hemos visto con anterioridad del JWT.

¿Cómo podemos dividir está estructura de Token BKS?

Está dividido en diez partes:

1. Id de Seguridad
2. IP usuario
3. Fecha de expiración del Token
4. XML con los Datos de usuario
5. Método de cifrado
6. Versión del Token
7. Emisor del Token
8. XML cifrado
9. Método de firma
10. Firma

Capítulo 5.

Librerías

Al trabajar en un Proyecto real, en una empresa real y con una funcionalidad real, este proyecto llevará una estructura marcada por la modularización y la paquetización, una buena práctica que actualmente es raro no llevar a cabo, pero que cabe explicar.

Esta estructura del Proyecto, o más individualmente de las librerías, nos proveerá de organización, simplificación de código y accesibilidad, lo que proporciona un mejor entendimiento del código para usuarios externos que trabajen con él posteriormente al realizar su actualización o mantenimiento.

Además, también cabe destacar, cómo se debe realizar el desarrollo de una librería para poder llevarla después a producción. Teniendo en cuenta que seguimos la estructura por paquetes y módulos, necesitaremos tener la siguiente jerarquía de ficheros y directorios:

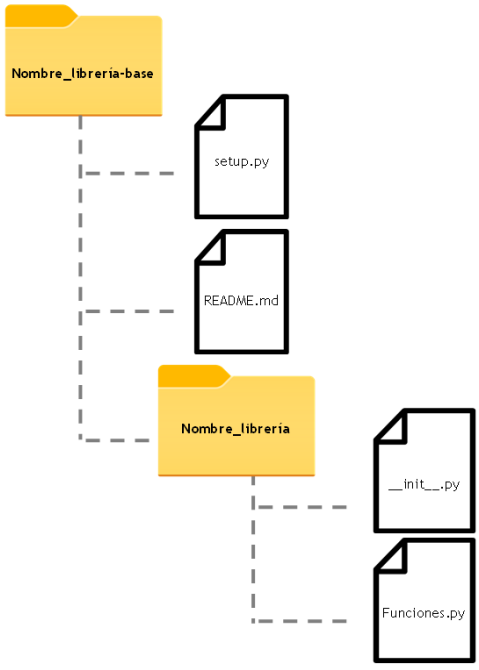


Figura 9. Jerarquía de ficheros y directorios librería Python

Como podemos observar en la Figura 9, debemos añadir el fichero setup.py y el archivo README.md para que nuestro proyecto pueda utilizarse posteriormente como una librería. A continuación, hablamos de ambos.

Fichero setup.py

Este fichero deberá contener instrucciones para poder instalar la librería, como, por ejemplo, requisitos previos de instalación de otras librerías.

Archivo README.md

Fichero en formato Markdown en el que incluiremos información sobre nuestra librería y que, aunque incluirlo es una buena práctica, es opcional.

Seguidamente, entramos de lleno a explicar las dos librerías que componen nuestro Proyecto.

5.1. Librería 1: Seguridad con JWT

5.1.1. Paquetes

5.1.1.1. Estructura

La estructura de esta librería la formaremos de la siguiente manera:

- En el paquete bks_credential podemos acceder al análisis del formato de un credencial Token BKS o Cookie BKS, guardando sus parámetros en diferentes variables para trabajar con ellas posteriormente, además de realizar operaciones de codificación y decodificación.
- En Corporate tenemos una clase que se encargara de hacer las llamadas correspondientes al STS y se asegura de llevar a cabo la conversión BKS a JWT validando las diferentes partes del Token: cabecera, datos y firma.
- En Services, nos encontramos con el servicio STS y la clase correspondiente a la memoria Cache. El servicio STS, comprueba que los parámetros que recibe de entrada son correctos y realiza las llamadas a la Cache para comprobar si hay una coincidencia de credenciales.
- En Type, nos encontramos con dos clases que tienen como funcionalidad diferenciar entre Cookie y Token BKS, con motivo de prepararlos en formato para ser enviados al servicio STS.
- Utils, es un paquete utilizado para guardar constantes o métodos que usaremos con regularidad en nuestro código.
- En excepciones, manejaremos los errores y excepciones que podemos tener en nuestro código para tener un control de errores.
- Por último, tendremos un paquete Test, donde se llevaran a cabo pruebas de código simulando un caso real, para comprobar que el código funciona como se espera.

5.1.1.2. Funcionalidad

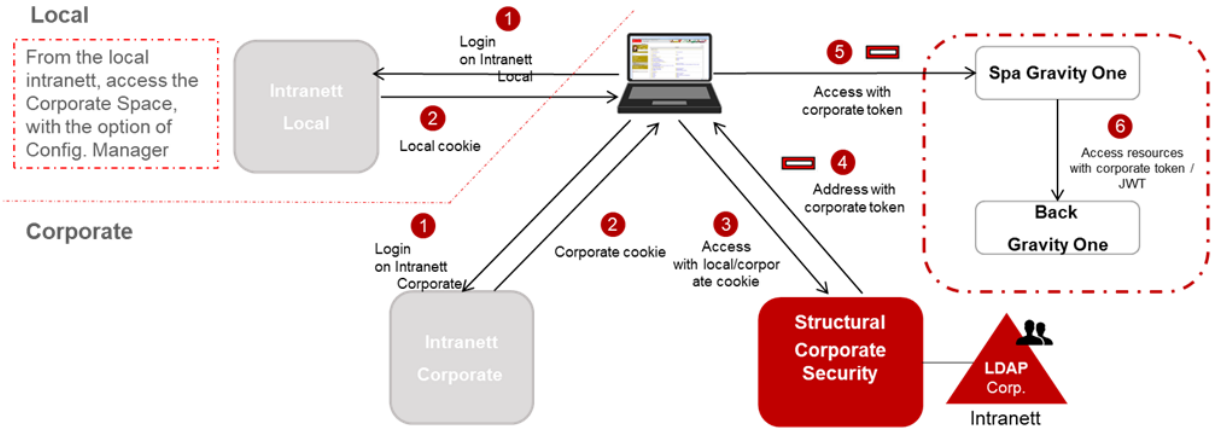


Figura 10. Diagrama de flujo Acceso a Microservicios con Token

En primer lugar, un usuario solicitará consumir un Microservicio. Como explicamos al principio del documento, este usuario puede ser Corporativo y tener sus propias credenciales o puede ser Externo, en ese caso se le proveerá de unos credenciales genéricas. Por otra parte, como podemos ver en la figura de arriba que puede pedir acceso desde el servidor local o desde el corporativo.

Los Token corporativos se puede crear, basándose en un Token BKS o en una Cookie BKS. Al hacer login con las credenciales, usuario y password, los servidores devuelven el Token o Cookie BKS, que en este último caso será transformada a Token por el conjunto Structural Corporate Security y capa LDAP cuando han sido verificados los credenciales en su base de datos. De aquí se redirigirá la petición HTTP tipo POST para solicitar al STS que valide y transforme el Token BKS a JWT.

Una vez recibido el JWT, se podrá acceder al Microservicio.

¿Cómo se obtiene el Token?

En el caso del Token corporativo BKS, su creación es interna, sin embargo en la creación de un JWT, es el sistema STS quién provee de dicho estándar, además de llevar a cabo el cambio de BKS a JWT. Para más información sobre los Token BKS o JWT, puede consultar el Capítulo 4. A continuación hablaremos de este servicio STS y de cómo funciona.

5.1.2. Security Token Service

La tarea principal de este servicio alojado en un Endpoint es la validación y generación de Tokens para mantener un SSO ("Inicio de Sesión Único") entre Aplicaciones. Su funcionalidad es cerrar la brecha que existe en un SSO entre aplicaciones heredadas y aplicaciones basadas en una arquitectura moderna ya que, mientras las primeras usan Tokens corporativos BKS para proporcionarlo, las modernas se han estandarizado para usar JWT.

Echemos un ojo a la Arquitectura de este Servicio:

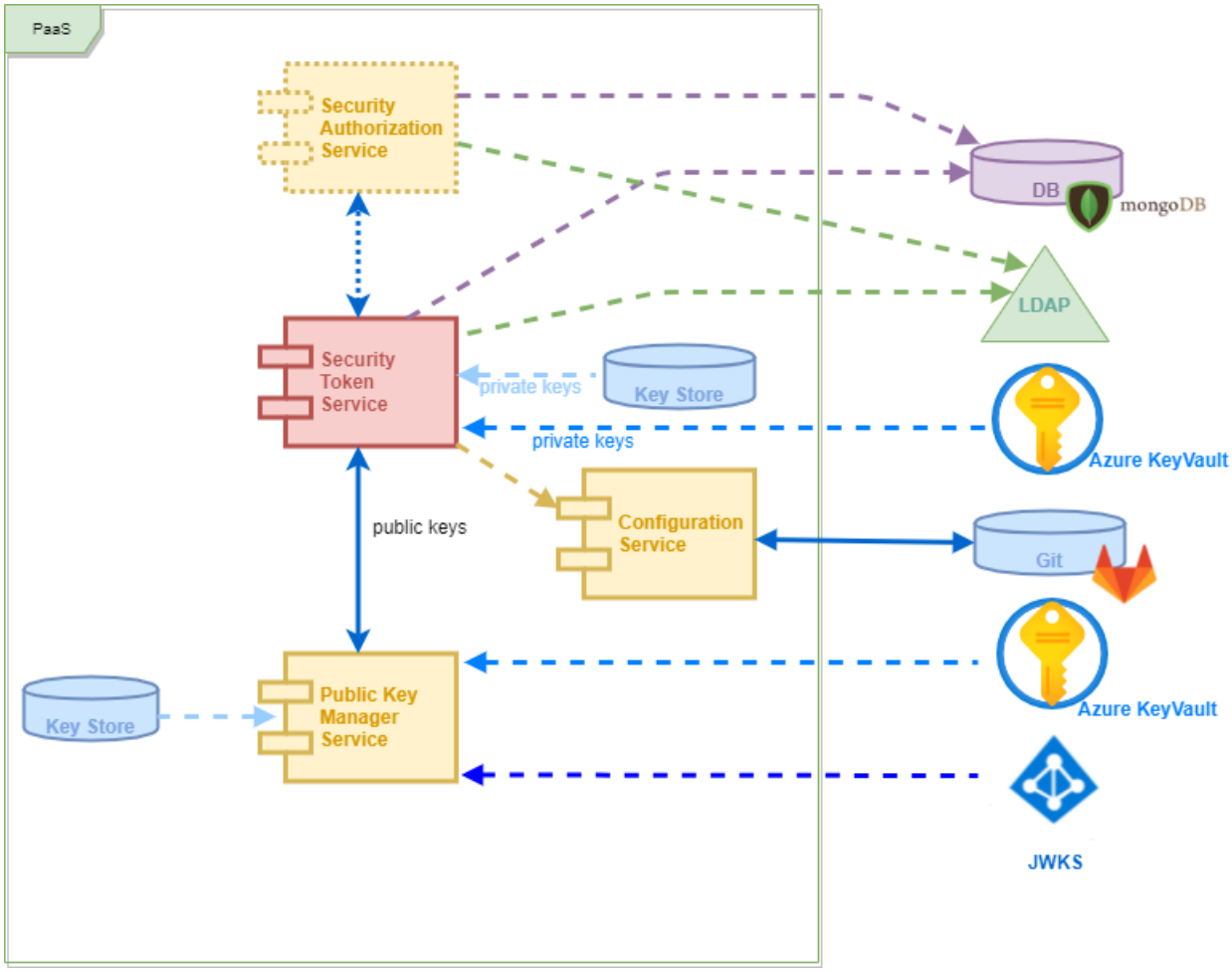


Figura 11. Diagrama Estructura Security Token Service

En primer lugar, nos encontramos con un modelo de Cloud Computing basado en Plataforma como Servicio (PaaS), donde un tercero brindará el sistema de hardware y una plataforma de software de aplicaciones, un entorno ideal para desarrolladores y programadores.

Podemos ver diferentes componentes que trabajan con el servicio STS, por lo que haremos una breve descripción de ellos.

- **Security Token Service (STS):** como hemos explicado anteriormente, es la pieza principal con la que trabajamos y que se encarga de validar e intercambiar Tokens.
- **Key Store:**
  - Private Keys: el servicio puede usar un fichero JKS, repositorio o Azure Key vault para obtenerlas.
  - Public Keys: el servicio puede usar un fichero JKS o el servicio remoto Public Key Manager Service como repositorio, cuya localización se especifica en la configuración del servicio.
- **Public Key Manager Service:** servicio remoto utilizado únicamente para obtener las claves públicas.
- **Configuration Service:** proporciona y gestiona la configuración de los Microservicios desplegados, está configuración se encuentra alojada en un repositorio de Git.
- **Database:** se usa para almacenamiento de aplicaciones y usuarios para un uso de punto final de usuario genérico. Este componente es opcional.
- **LDAP:** Protocolo Ligero de Acceso a Directorios, necesario para autenticar a los usuarios administradores de STS que usan los puntos finales de administración.
- **Security Authorization Service:** solo será necesario cuando los usuarios deban estar autorizados para acceder a los recursos de los Microservicios. Proporciona permisos de usuario en función de los roles y grupos LDAP, estos permisos se agregan a los Tokens generados.
- **JWKS:** Json Web Keys Server, ofrece claves públicas o certificados en formato JWK.
- **AZURE:** ofrece claves privadas desde Azure. La compañía guarda estas claves con una jerarquía de claves de cifrado, las cifra cuando las guarda y las descifra al leerlas, el cifrado es transparente para el usuario.

Aunque en nuestro filtro la funcionalidad del servicio STS será la de proporcionarnos un JTW a partir de un Token BKS, su diseño tiene más funcionalidades, con operaciones diferenciadas según la misma:

i. Conversión de Token JWT a Token BKS.
ii. Conversión de Token BKS (Token o Cookie) a JWT.
iii. Token BKS con usuario genérico.
iv. API para usuarios administradores en LDAP.

Esta funcionalidad se puede ejecutar a través de los siguientes puntos finales, divididos en dos grupos: API de servicio y Administración, utilizando códigos de estado http para la gestión de errores, donde nos encontraremos con códigos 1xx, que pertenecen a respuestas informativas, los 2xx, que hacen referencia a respuestas correctas, los 3xx, que serán redirecciones, 4xx, que corresponden a errores por parte del cliente y las 5xx, que tendrán que ver con errores por parte del servidor.

- **API de servicio**: permite la conversión de credenciales de acuerdo a los diferentes casos.

  - Endpoint de conversión de JWT Serenity Token a BKS Token

    Parámetros de la solicitud:
    Token – string

    Respuesta:
    Token BKS codificado

    Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The server could not verify input credentials.
401	Realm not found in configuration for specified issuer in *JWT token*.
401	The credential provided has expired.
405	Request method not supported.
500	Internal server error. Server logs may have additional information.

Tabla 1. JWT to BKS

  - Endpoint actualizado de JWT Serenity Token a BKS Token

    Parámetros de la solicitud:
    Token – string
    Audience – List<String>

    Respuesta:
    Token JWT codificado

    Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The server could not verify input credentials.

401	Realm not found in configuration for specified issuer in *JWT token*.
401	The credential provided has expired.
405	Request method not supported.
500	Internal server error. Server logs may have additional information.

Tabla 2. JWT to BKS actualizado

- Endpoint de conversión BKS Credential a JWT Serenity Token

  Parámetros de la solicitud:
  credentialType - String
  credential - String

  Respuesta:
  Token JWT codificado

  Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The server could not verify input credentials.
401	Realm not found in configuration for specified issuer in *JWT token*.
401	The credential provided has expired.
405	Request method not supported.
500	Internal server error. Server logs may have additional information.

Tabla 3. BKS to JWT

- Generic user Endpoint para generar BKS Token

  Parámetros de la solicitud:
  sessionId - String
  Respuesta:
  Token JWT codificado

  Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	User not authorized. Either credentials are expired or are not valid.

401	Realm not found in configuration for specified issuer in *JWT token*.
401	The credential provided has expired.
405	Request method not supported.
409	The client is not registered, it's necessary do it previously.
500	Internal server error. Server logs may have additional information.

Tabla 4. Generación BKS

- Generic user Endpoint para generar JWT

  Parámetros de la solicitud:
  Username, audience and extra claims
  Respuesta:
  Token JWT codificado

  Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The server could not verify input credentials.
401	Realm not found in configuration for specified issuer in *JWT token*.
401	The credential provided has expired.
405	Request method not supported.
409	The client is not registered, it's necessary do it previously.
500	Internal server error. Server logs may have additional information.

Tabla 5. Generación JWT

- Get Corporate Credential Expiration

  Parámetros de la solicitud:
  Cookie – String
  Authorization (Bearer) - String
  Respuesta:
  Expiration
  expirationRelative

  Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The credential provided is invalid.
401	Realm not found in configuration for specified issuer.
401	The credential provided has expired.
405	Request method not supported.
500	Internal server error. Server logs may have additional information.

Tabla 6. Credential Expiration

- Refresh Corporate Credential

  Parámetros de la solicitud:
  Cookie – String
  Authorization (Bearer) - String
  Respuesta:
  Token
  Expiration
  expirationRelative

  Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The credential provided is invalid.
401	Realm not found in configuration for specified issuer.
401	The credential provided has expired.
405	Request method not supported.
500	Internal server error. Server logs may have additional information.

Tabla 7. Actualización credenciales

- Get Token from Cookie

  Parámetros de la solicitud:
  Cookie – String
  Respuesta:
  Token
  Expiration
  expirationRelative

  Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The credential provided is invalid.
401	Realm not found in configuration for specified issuer.
401	The credential provided has expired.
405	Request method not supported.
500	Internal server error. Server logs may have additional information.

Tabla 8. Cookie to Token

- BKS Cookie logout

  Parámetros de la solicitud:
  Cookie – String
  Respuesta:
  Set-cookie

  Gestión de errores:

400	Request is malformed or there are missing mandatory parameters.
401	The credential provided is invalid.
401	Realm not found in configuration for specified issuer.
401	The credential provided has expired.
405	Request method not supported.
500	Internal server error. Server logs may have additional information.

Tabla 9. Cookie logout

- **Administración**: permite la gestión de las aplicaciones que requieren del usuario genérico (al que se le dan unos credenciales genéricos), accediendo a la operativa que está en el Host.

    - Create an application Endpoint.
    - Register a generic user to an application Endpoint.
    - Get info about an application Endpoint.
    - Reset password for an application.
    - Delete an application.

Para ejecutar esta funcionalidad en el Endpoint donde está alojado el servicio STS, mandaremos una Solicitud HTTP de tipo POST, incluyendo unos parámetros característicos para cada caso. El Endpoint contestará con una Response donde se incluirá la información solicitada además de un código de estado Http, confirmando que todo ha ido bien o informándonos de que ha podido suceder (Gestión de errores).

En cuanto a la seguridad de este servicio y no a la que el mismo proporciona con la creación y validación de Tokens, podemos decir que cumple con el Estándar de Seguridad Criptográfica, declarados como conexiones seguras tanto para peticiones entrantes como salientes, HTTPS, LDAPS, DDBB, etc.

5.1.3. Single Sign On

El inicio de Sesión único es un esquema que permite a los usuarios acceder a múltiples sistemas iniciando sesión una vez. Al aplicar este esquema, las aplicaciones crean una relación de confianza con un Servicio Estructural de Seguridad para delegar la autenticación y generar las credenciales apropiadas.

Por otro lado, las aplicaciones actúan como los Proveedores de Identidad (un sistema que crea, mantiene y administra la identidad de los usuarios). Aunque existen múltiples herramientas para ofrecer las capacidades de SSO, en nuestro Proyecto solo utilizaremos el Inicio de Sesión único basado en la tecnología de Token BKS, que gracias al servicio STS, se podrá llevar a cabo con la tecnología JWT.

5.2. Librería 2: Predicate and Filter Factories con SCG

Esta librería nos proveerá de la infraestructura de un API Gateway para enrutar todas las peticiones que después pasaremos por un filtro de validación de Tokens como filtro básico, para después poder añadir monitorización de otros parámetros si es necesario.

Es una herramienta de gestión de API que se encuentra ente el cliente y un conjunto de micros de backend y se encarga de enrutar las peticiones http que le llegan.

Funciona como un proxy inverso que acepta todas las llamadas a la interfaz de programación de la aplicación, agrega los servicios necesarios para satisfacer las solicitudes y devuelve el resultado.

En este caso, la librería constará de dos filtros, uno en el que se hará uso de la plataforma Spring Cloud Gateway y otro donde podremos usar las Route Predicate Factories y los Filter Gateway Factories que nos proporciona la misma, profundizaremos en esta plataforma para después ver cómo hacemos uso de ella.

Para poder entender mejor como funciona esta librería debemos entender bien los siguientes tres conceptos:

- **Ruta**: componente básico de SCG. Definido por un ID, URI de destino, colección de predicados y colección de filtros. Una ruta coincidirá si el predicado que agregamos a la solicitud es verdadero.
- **Predicado**: es una interfaz funcional de Java que devuelve True o False dependiendo de si el predicado coincide con la entrada de Spring Framework ServerWebExchange.
- **Filtro**: son instancias de Spring Framework GatewayFilter, en ellas se pueden modificar solicitudes y respuestas.

5.2.1. Spring Cloud Gateway

Este proyecto proporciona una API Gateway construida sobre el ecosistema Spring. Tiene como objetivo proporcionar una forma simple pero efectiva de enrutar las API en la Arquitectura de Microservicios. Proporciona funciones básicas de puerta de enlace basadas en la cadena de filtro: seguridad, monitoreo, limitación de corriente,… Es una solución para reemplazar a Zuul.

Cómo funciona:

Los clientes realizan solicitudes a Spring Cloud Gateway. Si la asignación del controlador determina que una solicitud coincide con una ruta, se envía al controlador web de puerta de enlace. Este controlador ejecuta la solicitud a través de una cadena de filtros que es específica de la solicitud, los filtros pueden ejecutar la lógica tanto antes como después de que se envíe la solicitud de proxy.

En el siguiente Diagrama podemos observar gráficamente el funcionamiento de la puerta de enlace SCG:

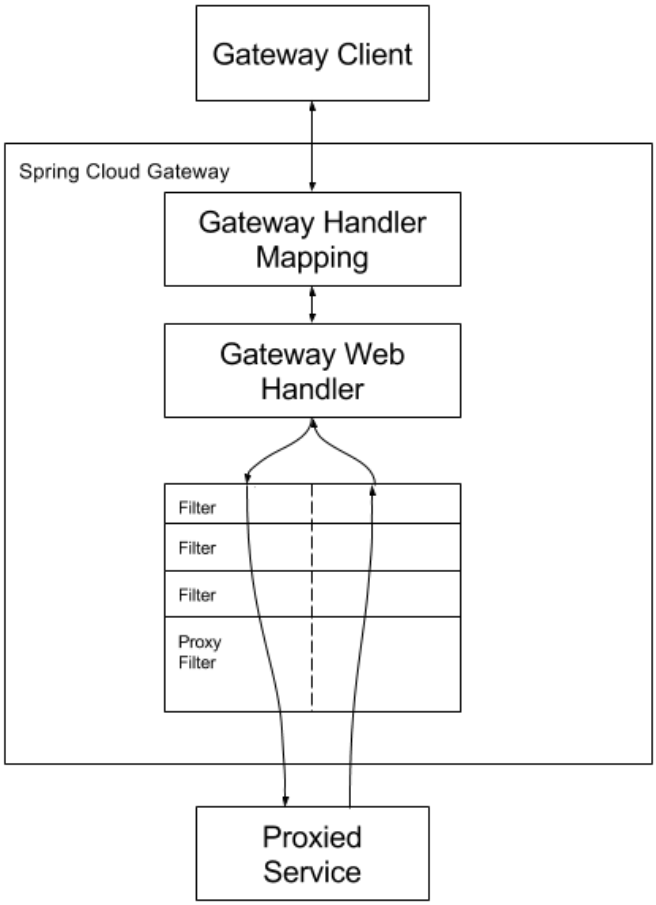


Figura 12. Diagrama Estructura Spring Cloud Gateway

Para la ejecución de este Gateway, necesitaremos de unas condiciones llamadas predicados, que podrán solicitarse de manera individual o agrupándose mediante operaciones lógicas “and”, y de unos filtros que podrán modificar las solicitudes Http. Todo ello forma nuestro filtro que serán las Fábricas de rutas de predicados. Profundicemos en ello un poco más.

5.2.2. Predicate and Filter Factories

Fábricas de predicados de ruta

Spring Cloud Gateway determina coincidencias de ruta como parte de la infraestructura Spring WebFlux HandlerMapping (Interfaz que define un mapeo entre solicitudes y objetos, proporcionando una gran personalización de este mapeo, lo que lo hace una capacidad inusual y poderosa). SCG, incluye muchas fábricas de predicados de ruta integradas que coinciden con diferentes atributos de la solicitud HTTP y que pueden combinar predicados de ruta con la sentencia lógica “and” como hemos comentado anteriormente.

Fábricas de predicados de ruta disponibles en SCG

- **After**

  Esta fábrica toma como parámetro "datetime" y coincide con las solicitudes que ocurren después de la fecha y hora especificada.

  Ejemplo: esta ruta coincide con cualquier solicitud realizada después del 20 de enero de 2017 a las 17:42, hora de la montaña (Denver).

```
spring:
  cloud:
    gateway:
      routes:
      - id: after_route
        uri: https://example.org
        predicates:
        - After=2017-01-20T17:42:47.789-07:00[America/Denver]
```

- **Before**

  Esta fábrica toma como parámetro "datetime" y coincide con las solicitudes que ocurren antes de la fecha y hora especificada.

  Ejemplo: esta ruta coincide con cualquier solicitud realizada antes del 20 de enero de 2017 a las 17:42, hora de la montaña (Denver).

```
spring:
  cloud:
    gateway:
      routes:
      - id: before_route
        uri: https://example.org
        predicates:
        - Before=2017-01-20T17:42:47.789-07:00[America/Denver]
```

- **Between**

  Esta fábrica toma dos parámetros: datetime1 y datetime2, y coincide con las solicitudes que ocurren después de datetime1 y antes de datetime2. El parámetro datetime2 debe ser posterior a datetime1.

  Ejemplo: esta ruta coincide con cualquier solicitud realizada después del 20 de enero de 2017 a las 17:42 hora de la montaña (Denver) y antes del 21 de enero de 2017 a las 17:42 hora de la montaña (Denver). Esto podría ser útil para las ventanas de mantenimiento.

```
spring:
  cloud:
    gateway:
      routes:
      - id: between_route
        uri: https://example.org
        predicates:
        - Between=2017-01-20T17:42:47.789-07:00[America/Denver], 2017-01-21T17:42:47.789-07:00[America/Denver]
```

- **Cookie**

  Esta fábrica toma dos parámetros: cookie name y regexp (una expresión regular de Java). El predicado coincide con las solicitudes cuyas cookies tienen el nombre dado y cuyos valores coinciden con la expresión regular.

  Ejemplo: esta ruta coincide con las solicitudes que tienen un nombre de cookie chocolate cuyo valor coincide con la ch.p (expresión regular).

```
spring:
  cloud:
    gateway:
      routes:
      - id: cookie_route
        uri: https://example.org
        predicates:
        - Cookie=chocolate, ch.p
```

- **Header**

  Esta fábrica toma dos parámetros, header name y regexp (expresión regular de Java). El predicado coincide con un encabezado que tiene el mismo nombre y cuyos valores coinciden con la expresión regular.

Ejemplo: esta ruta coincide si la solicitud tiene un encabezado llamado X-Request-Id cuyo valor coincide con la expresión regular \d+ (es decir, si tiene un valor de uno o más dígitos).

```
spring:
  cloud:
    gateway:
      routes:
      - id: header_route
        uri: https://example.org
        predicates:
        - Header=X-Request-Id, \d+
```

- **Host Route**

Esta fábrica toma un parámetro, una lista de nombres de host, patterns. El predicado coincide con el host.
Ejemplo: esta ruta coincide si la solicitud tiene un host de encabezado con un valor de www.somehost.org o beta.somehost.org o www.anotherhost.org.

```
spring:
  cloud:
    gateway:
      routes:
      - id: host_route
        uri: https://example.org
        predicates:
        - Host=**.somehost.org,**.anotherhost.org
```

- **Method Route**

Está fábrica toma como argumento methods, el que incluye uno o más parámetros que serán los métodos http para hacer coincidir.

Ejemplo: esta ruta coincide si el método de solicitud fue GET o POST.

```
spring:
  cloud:
    gateway:
      routes:
      - id: method_route
        uri: https://example.org
        predicates:
        - Method=GET,POST
```

- **Path Route**

  Esta fábrica toma dos parámetros: una lista de Spring PathMatcher patterns y una bandera opcional llamada matchOptionalTrailingSeparator.

  Ejemplo: esta ruta coincide si la ruta de solicitud fue, por ejemplo, /red/1 o /red/blue o /blue/Green.

```
spring:
  cloud:
    gateway:
      routes:
      - id: path_route
        uri: https://example.org
        predicates:
        - Path=/red/{segment},/blue/{segment}
```

- **Query Route**

  Esta fábrica toma dos parámetros, uno requerido, param, y otro opcional, regexp.

  Ejemplo: la ruta anterior coincide si la solicitud contenía un parámetro de consulta green.

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: https://example.org
        predicates:
        - Query=green
```

- **Remote Addr**

  Esta fábrica toma una lista (de tamaño mínimo 1) de sources, que son cadenas de notación CIDR en IPv4 o IPv6.

  Ejemplo: esta ruta coincide si la dirección remota de la solicitud fue, por, ejemplo, 192.168.1.10.

```
spring:
  cloud:
    gateway:
      routes:
      - id: remoteaddr_route
        uri: https://example.org
        predicates:
        - RemoteAddr=192.168.1.1/24
```

Fábricas de Filtros Gateway

Los filtros de ruta permiten la modificación de la solicitud HTTP entrante o la respuesta HTTP saliente y se limitan a una ruta en particular. Spring Cloud Gateway incluye muchas fábricas GatewayFilter integradas, las vemos a continuación.

Fábricas de filtros disponibles en SCG

Figura 13. GatewayFilter Factories

5.2.3. Funcionalidad

Estructuraremos esta librería de la siguiente forma:

Esta librería cuenta tan solo con dos clases. La primera, Views, haciendo referencia al modelo de vistas de Django y manteniendo su formato para FastAPI, consiste en un código que forma la estructura Spring Cloud Gateway, llevando a cabo el enrutamiento y el procesado de los predicados. Para realizar el enrutamiento, se tiene un archivo de configuración con las rutas que debe manejar este Gateway.

Por otra parte, tenemos la clase apps donde se complementan la configuración proporcionada por nuestro framework con la configuración de SCG.

Capítulo 6.

Django

6.1. Introducción a Django

Django es un framework de desarrollo web de código abierto en Python que sigue el diseño conocido como MVC (Modelo-Vista-Controlador). Creado en Julio de 2005 por Adrian Holovaty y Simon Willison, su meta fundamental es facilitar la creación de sitios web complejos. Pone énfasis en el reuso, conectividad y extensibilidad de componentes, el desarrollo rápido y el principio DRY (Don`t Repeat Yourself). Podemos encontrar su documentación en https://www.djangoproject.com/.

6.2. Pros/Cons

Pros

Es un framework de uso general que es:

- Completo, provee casi todo lo que los desarrolladores quisieran que tenga de fábrica, sigue principios de diseño consistentes y tiene una amplia y actualizada documentación.
- Versátil, usado para construir casi cualquier tipo de sitio web.
- Seguro, ayuda a los desarrolladores a evitar varios errores comunes de seguridad al ser diseñado para hacer lo correcto en cuanto a proteger un sitio web automáticamente.
- Escalable, usa un componente basado en la arquitectura “shared-nothing” (cada parte de la arquitectura es independiente de las otras y puede ser reemplazado o cambiado).
- Mantenible, por utilizar el principio DRY y seguir el diseño MVC que agrupa código relacionado en módulos.
- Portable, está escrito en Python, el cual se ejecuta en muchas plataformas.

Cons

- A veces, Django no es apropiado para proyectos pequeños con pocas funciones porque la funcionalidad del marco puede confundir al desarrollador.
- El menor número de dependencias hace que el desarrollo se complete con una gran cantidad de código.
- Presenta muchas características y configuraciones, por lo que el usuario no puede aprenderlo rápidamente.
- La solicitud de cada proceso individual hace que el proceso de desarrollo de Django sea más lento.

Capítulo 7.

FastAPI

7.1. Introducción a FastAPI

FastAPI es un Framework para el desarrollo de API RESTful en Python. Está basado en Pydantic y escribe sugerencias para validar, serializar/deserializar datos y generar automáticamente documentos OpenAPI.

Es un Framework moderno, rápido que se creó teniendo en cuenta la asincronía y se puede usar con Python a partir de la versión 3.6. Es desarrollado por Sebastián Ramírez en diciembre de 2018 y su documentación la podemos encontrar en https://fastapi.tiangolo.com/.

7.2. Pros/Cons

Pros

Aunque en nuestro caso la mayor ventaja ha sido la documentación automática que nos provee este framework a través del endpoint \docs de la cual mostraremos ejemplos en el Capítulo 8, cuando FastAPI es destinado a desarrollar APIs nos encontramos con características como las siguientes.

- Es uno de los frameworks más rápidos disponibles, gracias a Starlette y Pydantic.
- Menos errores, reduce alrededor de un 40 %.
- Intuitivo, gran soporte de editor.
- Fácil, para usar y aprender.
- Corto, minimiza la duplicación de código.
- Robusto, código listo para producción.
- Basado en estándares y compatible con OpenAPI y JSON Schema.

Cons

Probablemente no era la mejor solución para este proyecto al no poder complementar las librerías que podemos encontrar en su documentación con los accesos a otros servicios corporativos con los que debía conectarse. Además, es un marco relativamente nuevo por lo que la comunidad es pequeña.

Capítulo 8.

Implementación

8.1. Requerimientos

En todos los proyectos necesitamos de un fichero donde podamos recoger las versiones de los módulos que utilizamos, además de un listado de los mismos. Este fichero suele llevar el nombre de requirements, se guarda en formato .txt y puede generarse de manera automática en la ventana de comandos.

Cuando hacemos un desarrollo, esto se vuelve algo fundamental. Lo habitual es crear un Entorno Virtual para nuestro Proyecto donde podamos manejar las librerías que necesitamos exclusivamente para ese desarrollo con sus respectivas versiones en vez de instalarlas para todo el sistema, todo ello de forma aislada, lo que nos facilitará la posterior exportación de nuestro desarrollo y evita las incompatibilidades con otras aplicaciones.

Gracias a este fichero requirements.txt, otros desarrolladores pueden probar nuestro proyecto, instalando las versiones oportunas en su entorno virtual y ejecutando el código sin problemas.

También es habitual tener diferentes versiones de un desarrollo, ya que una primera versión puede tener fallos o problemas que se irán corrigiendo, para ello es necesario llevar un seguimiento de versiones con sus respectivos logs de errores.

8.2. Ejemplos Activos

8.2.1. Django

Ejemplo 1. Gateway de Seguridad en Django para consumir sobre Postman

A continuación, veremos una petición http de tipo GET extraída de Postman para solicitar un JWT a partir de un BKS.

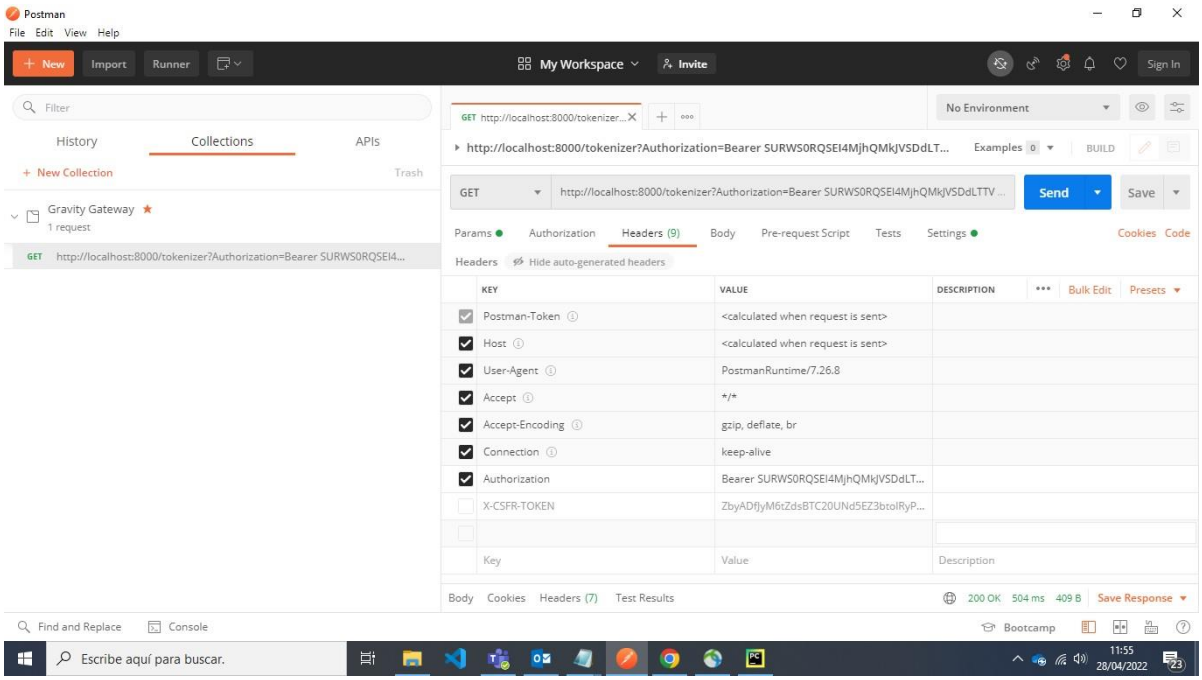


Figura 14. Request BKS to JWT en Postman

Una vez finalizado el desarrollo sobre Django, después de haber superado con éxito los tests y haber comprobado que el Gateway funciona como esperamos, el siguiente paso es probarlo en una aplicación real. Para ello se utilizará la Aplicación Postman, donde a modo de ejemplo lanzaremos la petición que más utilidad y con más frecuencia se dará en nuestro desarrollo: petición http de tipo GET para obtener un Token JWT a partir de un Token BKS dado.

¿Qué campos necesitamos rellenar en la petición Http?

URL a la que hacemos la petición:

http://localhost:8000/tokenizer?Authorization=Bearer SURWS0RQSEI4MjhQMkJVSDdLTTVKNzZKI1VOS05PV05fSVAjNTE5MjI5NTgwOTUxO-CNQRDk0Yld3Z2RtVnljMmx2YmowaU1TNHdJaUJsYm1Od-lpHbHVaejBpU1ZOUExUZzROVGt0TVNJL1BqeDBiMnRsYmtSbFptbHVhWFJwYjI0K1BIVnpa-WEpKUkQ1MWMyVnlTV1E4TD

En este caso, lanzamos la petición desde nuestro servidor local, por el puerto 8000.

Después de configurar nuestra Request en Postman, y darle a send, recibiremos la correspondiente Response, que nos devuelve el Token JWT en formato Json que vemos en la siguiente figura.

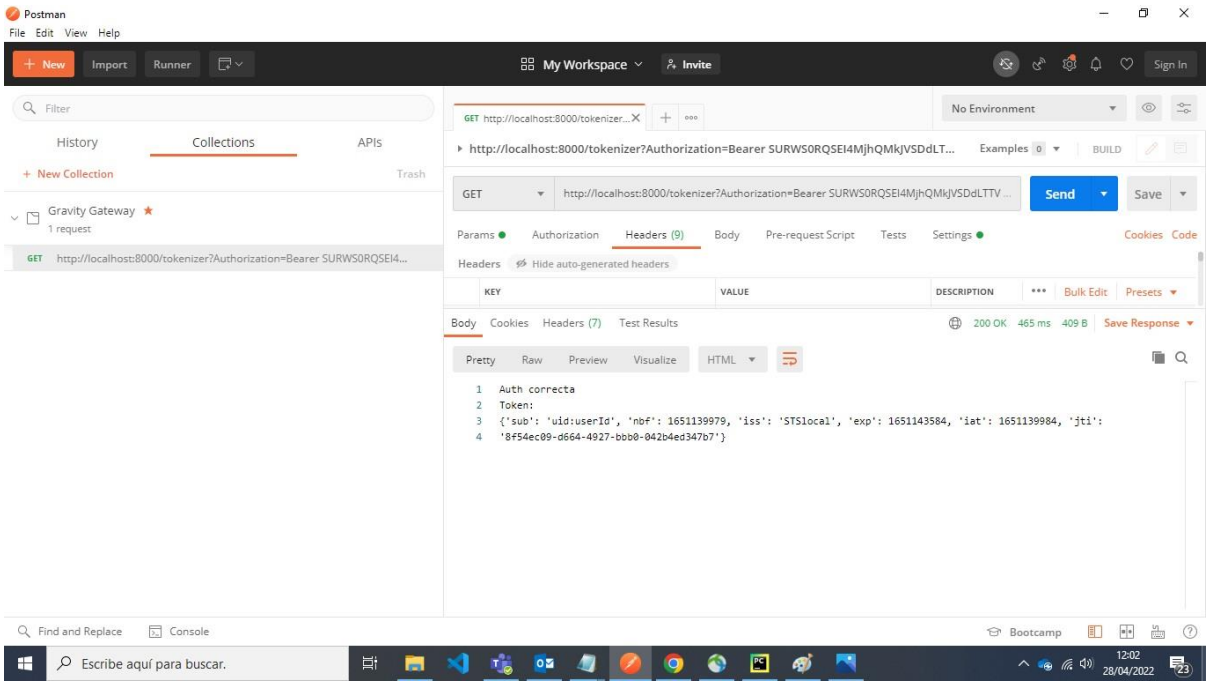


Figura 15. Response con el JWT

En Postman no podemos ver la cadena de caracteres que conforman nuestro JWT, pero si podemos hacerlo en la terminal de nuestro IDE, Pycharm. De esta manera, en la herramienta https://jwt.io/ verificamos que efectivamente se trata de un JWT válido y que corresponde con el Payload recibido en Postman.

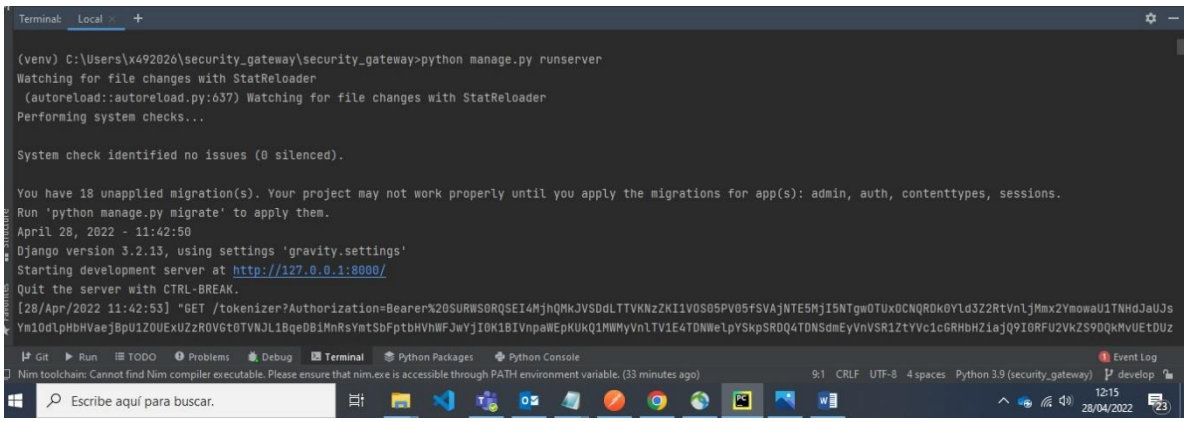


Figura 16. Arranque servidor Django en la Terminal

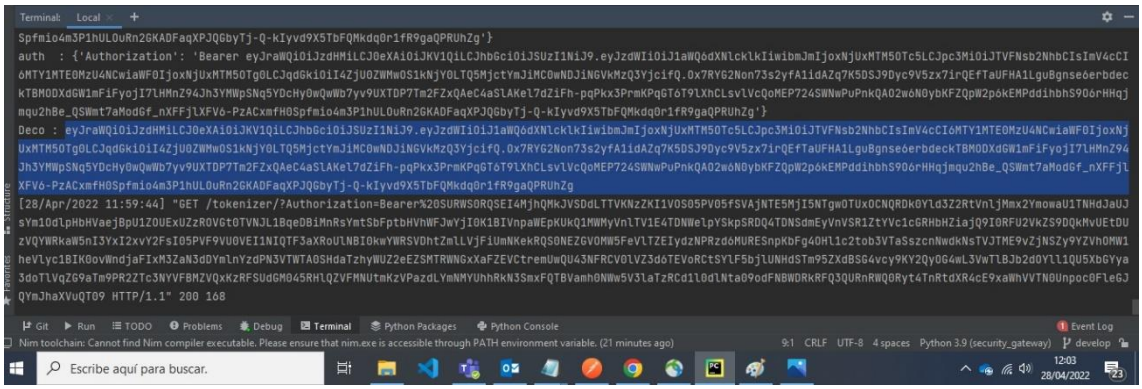


Figura 17. Cadena JWT recibida en la Terminal

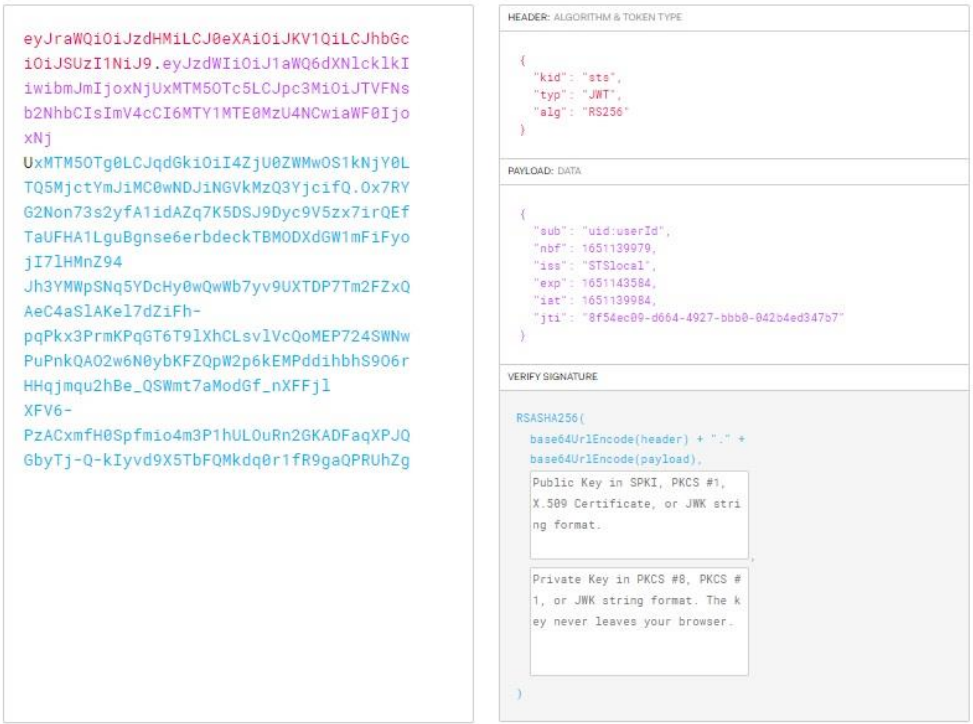


Figura 18. Comprobación jwt.io

8.2.2. FastAPI

Ejemplo 1. Documentación Automática con FastAPI en un "Hello World"

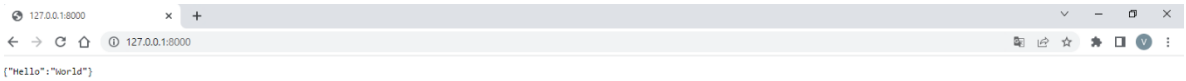


Figura 19. JSON Hello World con FastAPI utilizando Uvicorn

Figura 20. Acceso documentación automática en el endpoint \docs

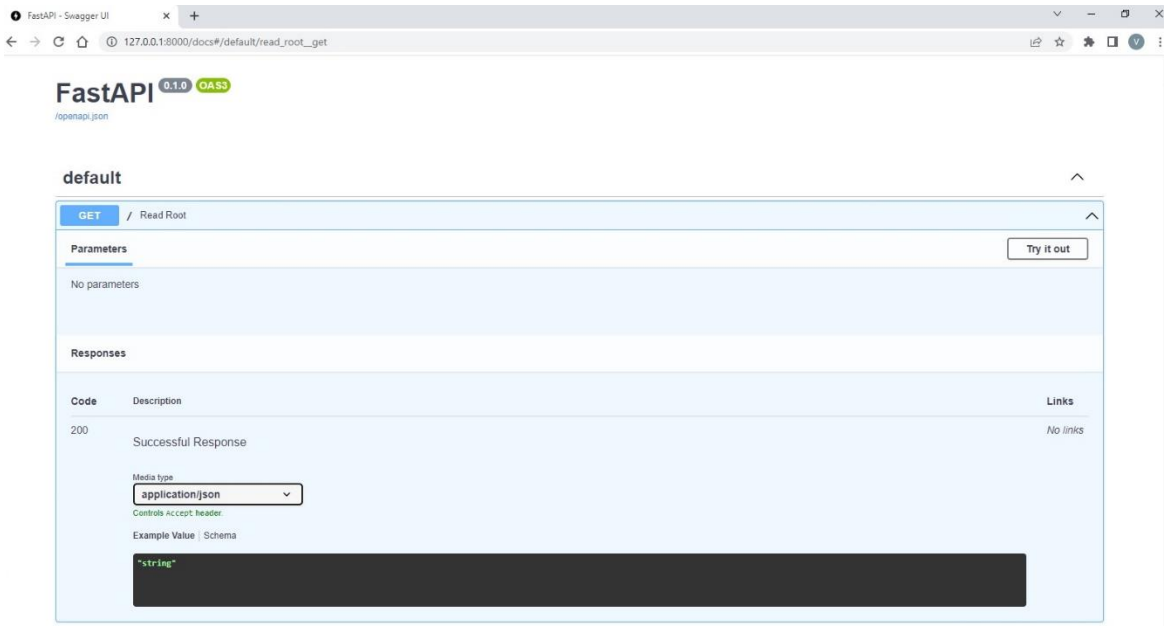


Figura 21. JSON Hello World con FastAPI utilizando Uvicorn

¿Para qué sirve esta documentación automática?

Vamos a empezar explicando el ejemplo más sencillo del mundo de la programación que es el famoso Hello World.

Como podemos ver en las figuras anteriores, se probarán las operaciones de tipo GET, POST, PUT y DELETE que pueden ejecutarse en nuestros puntos finales. En este caso, el "Hello World" consiste básicamente en una operación de tipo GET para imprimir una cadena, algo de lo que nos informan en esta documentación. Además de que tipo de petición es, nos informa de si necesitamos o no pasar algún parámetro o el tipo de respuesta que podemos obtener, esto es algo que ayuda con la validación de datos y acerca a los clientes a saber cómo funcionan las API REST.

Esta documentación es un esquema que se genera automáticamente gracias a la especificación Swagger de OpenAPI y para la que hemos elegido utilizar el servidor web asíncrono Uvicorn, herramientas de las que podemos obtener más información en sus respectivas web.

https://swagger.io/specification/

https://www.uvicorn.org

Una vez visto un ejemplo sencillo como es el Hello World, a continuación, veremos la documentación que nos ofrece FastAPI en el desarrollo de nuestras librerías.

Ejemplo 3. Documentación Automática para la petición de un Token JWT a partir de un Token BKS

En este otro apartado, vamos a ver la Documentación Automática que se genera cuando queremos hacer una Request de tipo GET para pasar un Token BKS y que nos devuelva un Token JWT. Como veremos en las imágenes de a continuación, a la petición http GET le pasamos como parámetro una Request.

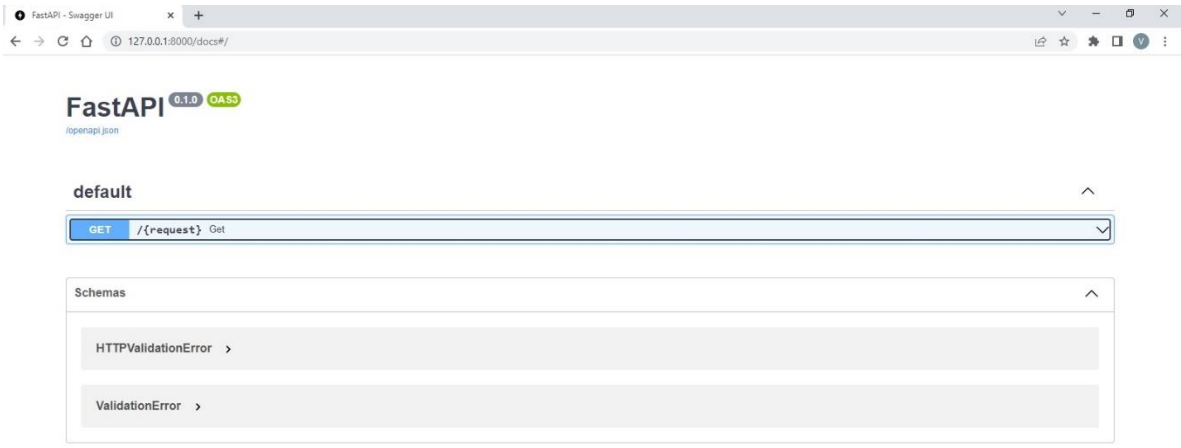


Figura 22. Tipo de peticiones que se pueden generar en el Gateway de Seguridad

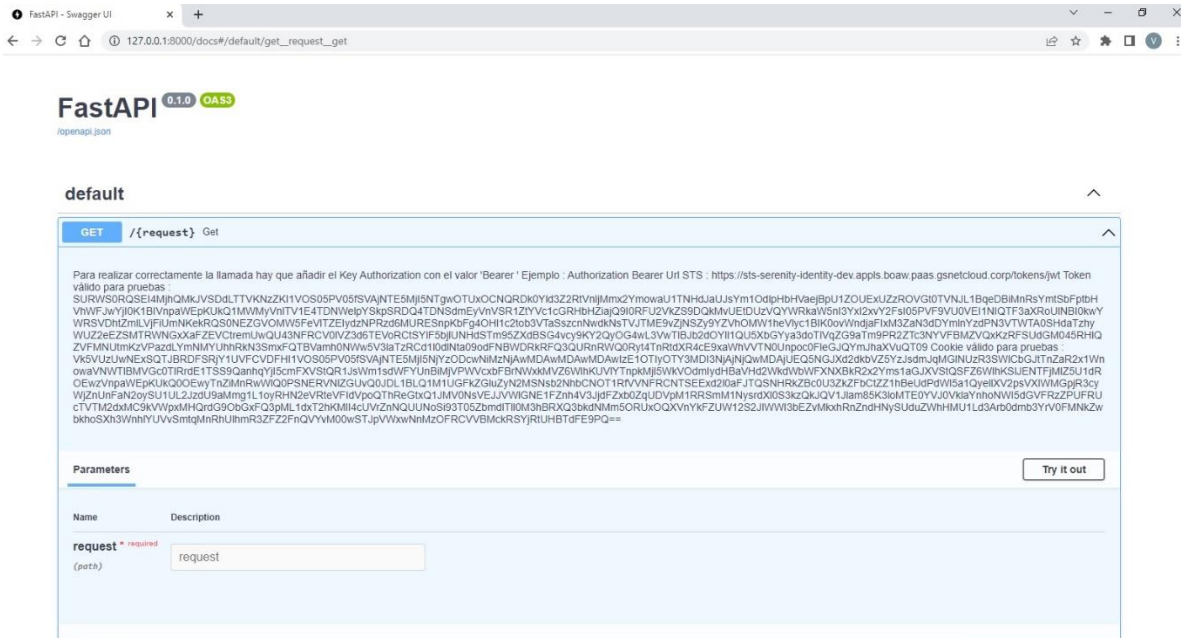


Figura 23. Comentarios acerca de cómo se debe lanzar la petición GET

Nos pararemos en este punto para explicar algo muy interesante que nos crea OpenAPI, y es que la documentación en forma de comentario que podemos ver en el código donde está programado este endpoint, será traspasada a la Documentación Automática como parte de ella. Algo que es de gran utilidad para poder entender que hace un método exactamente.

Por último, podemos ver como nos informa sobre los tipos de respuestas que podemos tener: una exitosa, que nos devolverá un JWT en una cadena String, y otra con error, en caso de que falte algo en nuestra Request. Además, podemos probarlo en tiempo real.

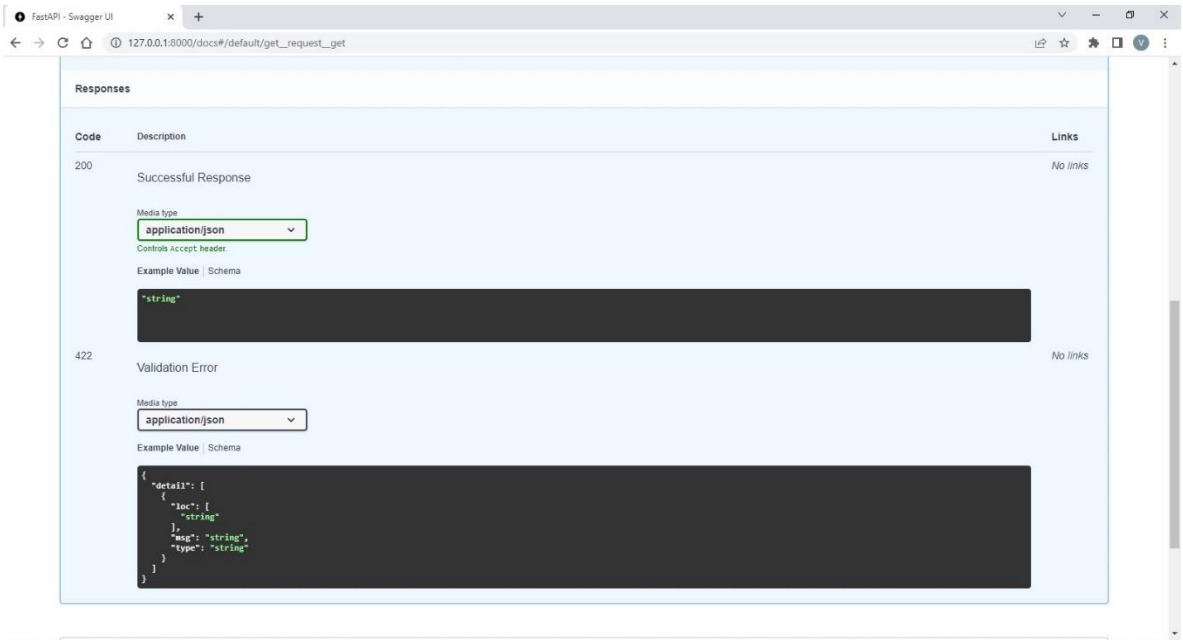


Figura 24. Posibles respuestas a la petición GET

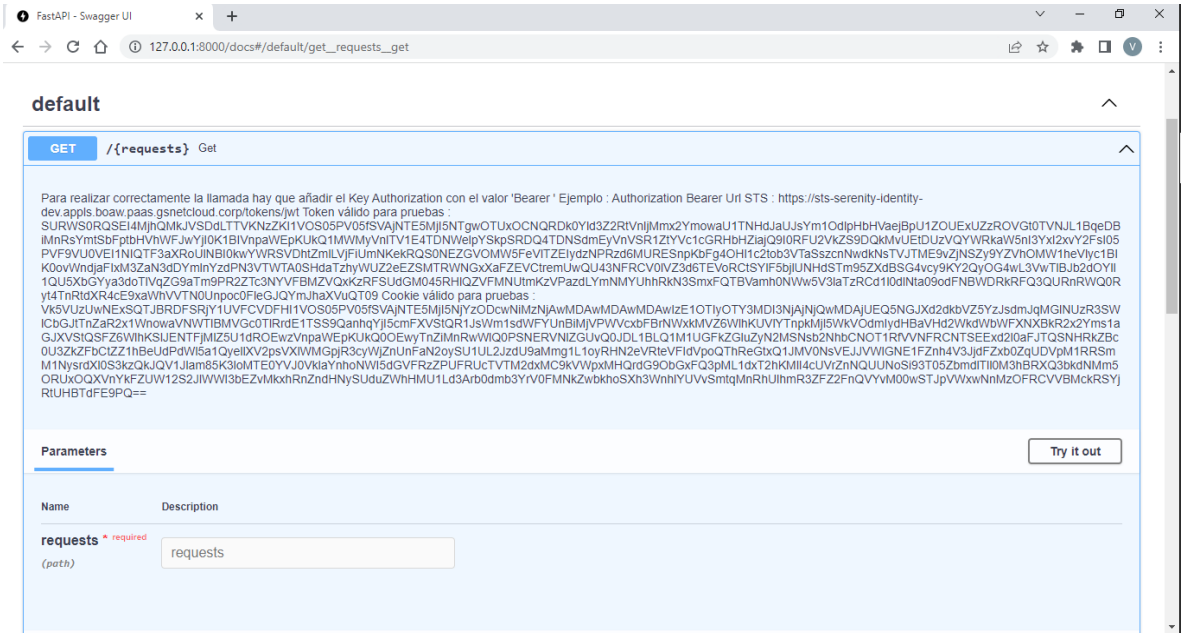


Figura 25. Try it out

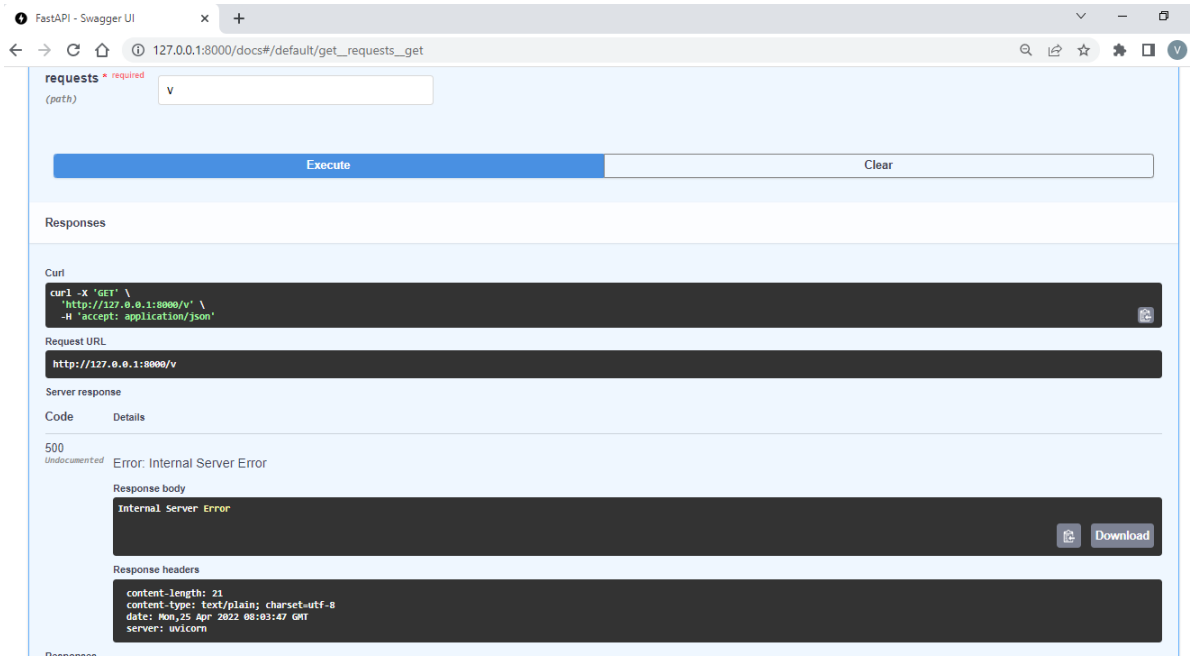


Figura 26. Validación

Ejemplo 4. Gateway de Seguridad en FastAPI para consumir sobre Postman

Para terminar mostramos el resultado del mismo Proyecto en Postman desarrollado con el Framework FastAPI en su versión híbrida usando alguna librería de Django.

Para arrancar FastAPI utilizaremos el servidor web ASGI Uvicorn.

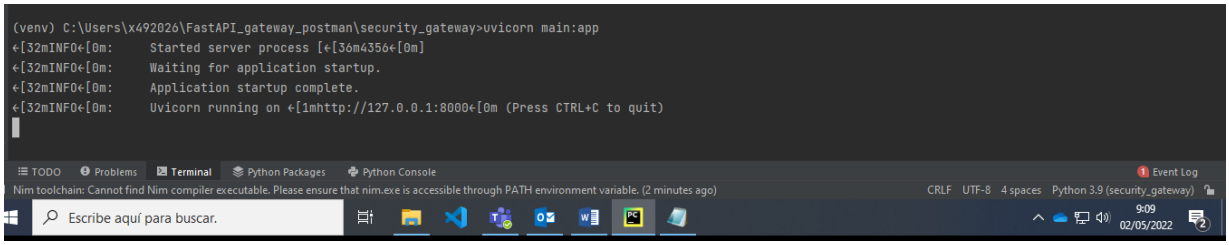


Figura 27. Arranque servidor Uvicorn de FastAPI en la Terminal. Pycharm

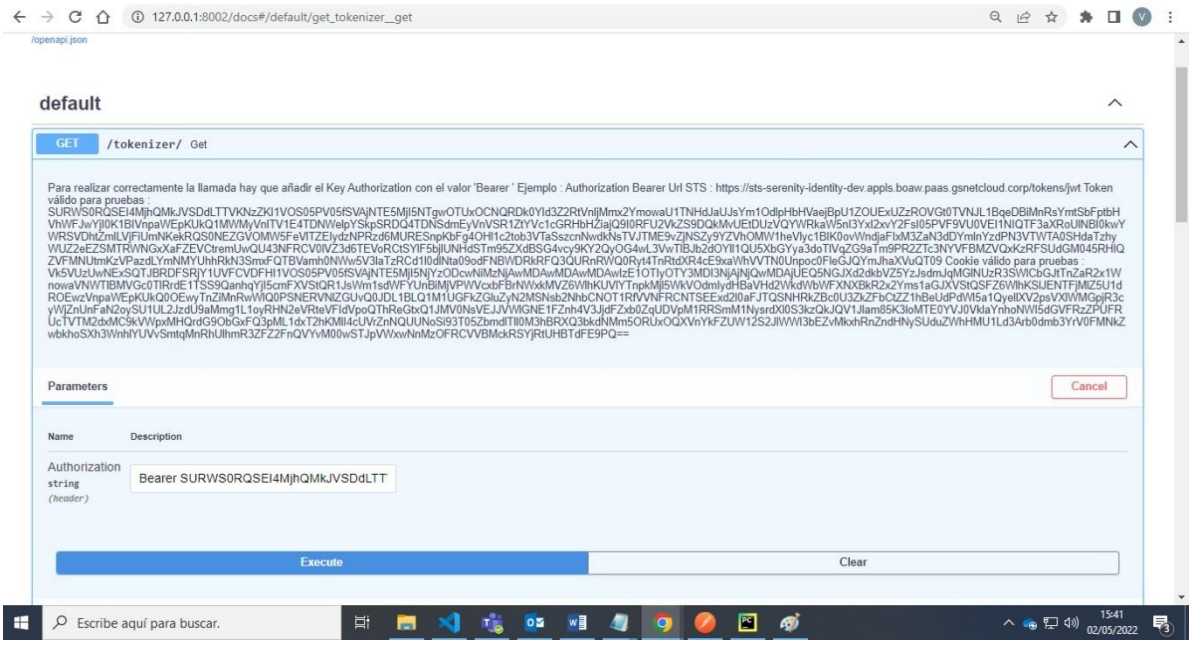


Figura 28. FastAPI Request.

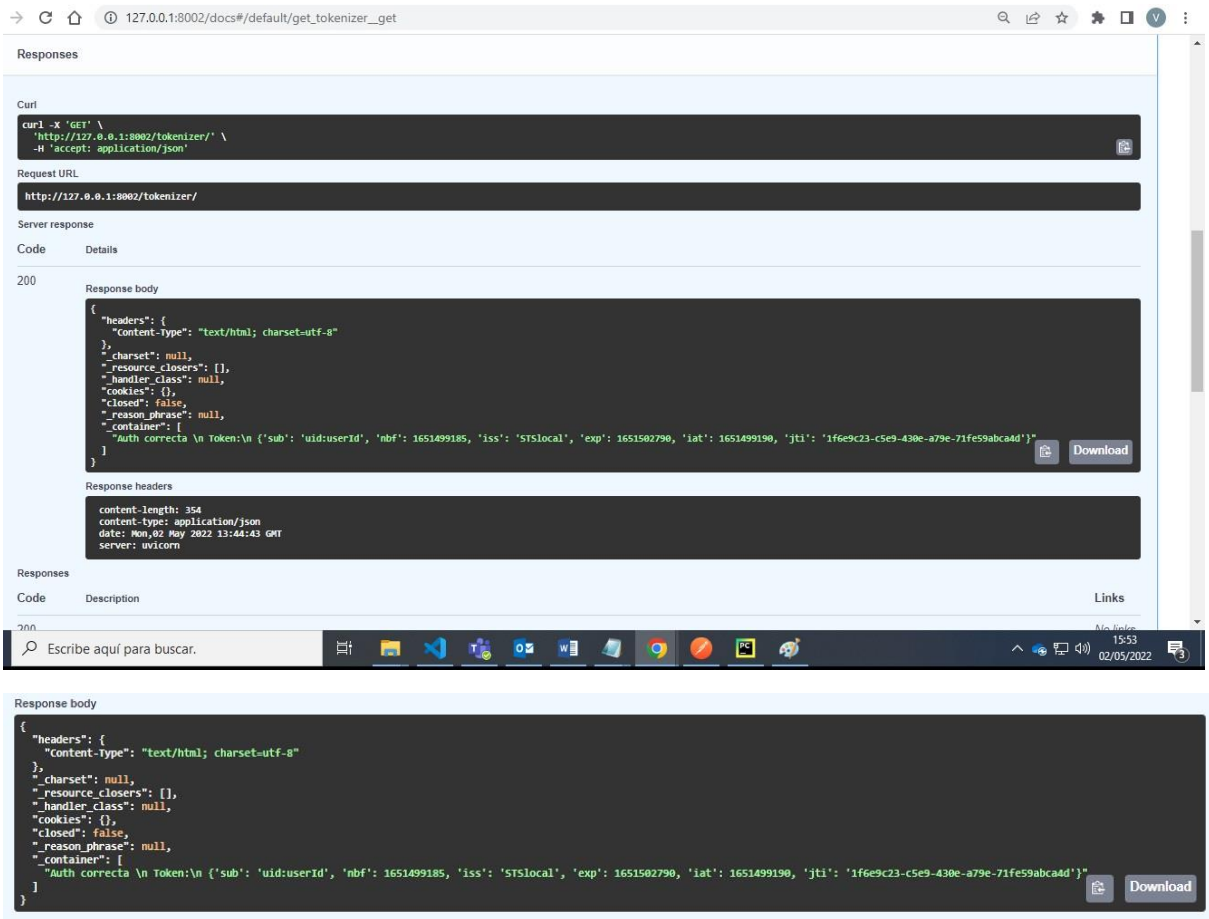


Figura 29. FastAPI Response.

Figura 30. JWT codificado

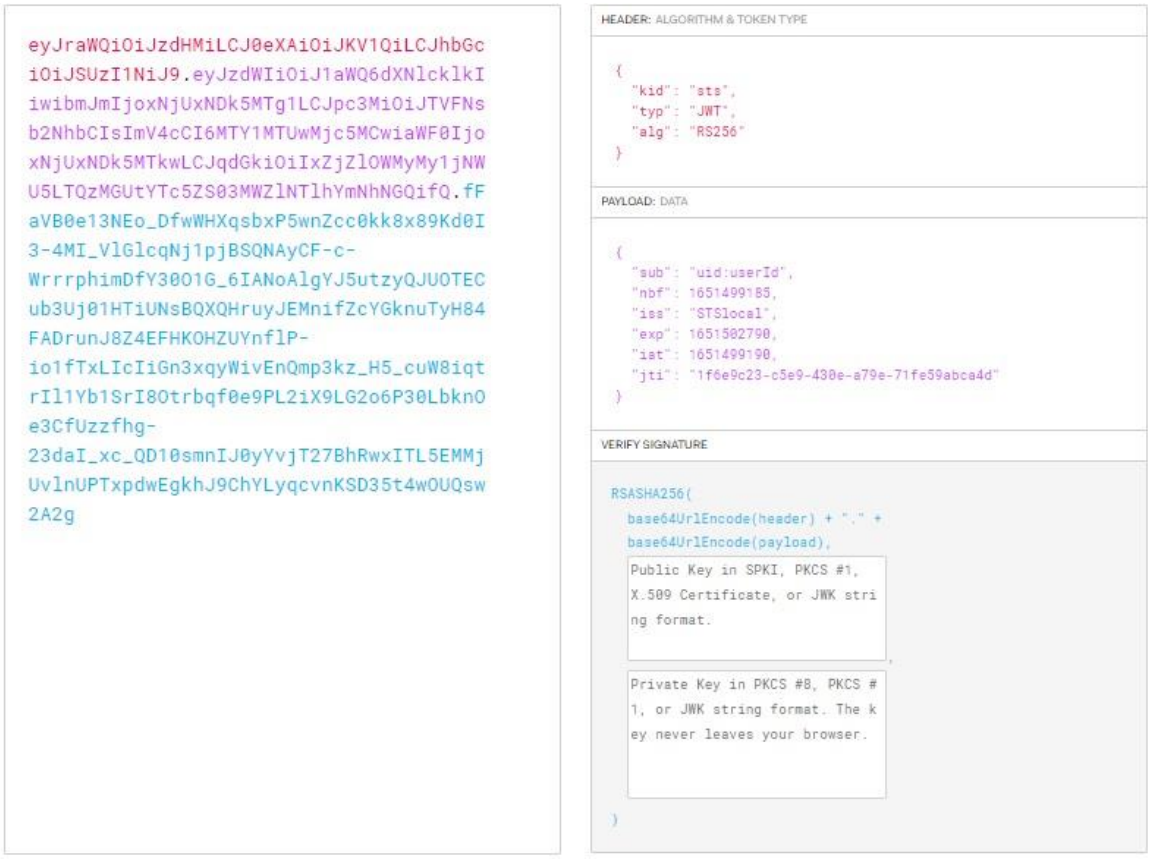


Figura 31. JWT decodificado

Capítulo 9.

Conclusiones

Cuando surgió la idea de realizar el Proyecto para llevar a cabo este TFG, se propuso con motivo de utilizar un Framework más específico para el desarrollo de API Restful como es FastAPI, ya que Django aunque está bien consolidado en el mercado y podemos obtener multitud de información sobre sus desarrollos, al final tiene un carácter general, lo que significaría que podemos encontrarnos con herramientas más potentes.

Una vez que finalizó el análisis y la documentación del Desarrollo del Gateway sobre Django y comenzó la programación del desarrollo sobre FastAPI, comenzaron los problemas, librerías utilizadas con Django que ahora no podíamos utilizar, librerías en FastAPI que aunque nos daban funcionalidades muy parecidas no eran implementables en un desarrollo que debía funcionar junto con otros servicios internos corporativos, etc.

Finalmente, no hemos podido hacer uso de algunas librerías que proporciona FastAPI al no comenzar el desarrollo desde cero, pero hemos podido utilizar la documentación automática que nos proporciona este Framework, algo que sin duda merece mucho la pena y nos ahorra mucho trabajo.

Por todo esto, la mayor conclusión que podemos sacar de este Proyecto es la importancia de la elección de un marco de trabajo adecuado a las necesidades de un Desarrollo, y de las ventajas y desventajas que habrá que valorar, como en este caso, que no hemos podido hacer uso de algunas de las librerías que nos proporcionaba la documentación de FastAPI, pero hemos podido obtener la documentación automática que es tan importante en la pre-producción, que nos da la opción de poder presentar a un cliente la funcionalidad que tiene nuestro proyecto y realizar cambios de forma muy sencilla e intuitiva.

Capítulo 10.

Valoración Personal

Lo primero que me gustaría destacar de este TFG ha sido lo cómoda que me he sentido con el lenguaje de programación Python, un lenguaje que está creciendo exponencialmente y ya es uno de los más populares del mundo. Es un lenguaje de propósito general que puede ser usado para casi todo, lo que da bastante valor a poder haberlo aprendido en este Proyecto.

Respecto a la parte del Desarrollo, el tener que adaptar el nuevo código a un desarrollo que ya estaba prácticamente preparado, me ha dado la visión de lo que significa adaptar lo nuevo a lo que ya está funcionando, algo que a priori puede parecer más sencillo pero con lo que sin duda te encuentras con más problemas por el camino y que al final es la realidad que nos encontramos en cualquier trabajo.

Sin duda ha sido una experiencia enriquecedora donde no he tenido que luchar con un código yo sola, pero he podido ver como todo tiene que unificarse y como no siempre depende de una persona que el desarrollo funcione, ya que pueden estar caídos los servidores, el STS o incluso una base de datos. He vivido de primera mano que puedes tener problemas de incompatibilidad de librerías y aplicaciones, incluso puedes tener que esperar a que el equipo de soporte te de permisos para instalar un complemento en tu PC.

Con todo esto debo decir que he aprendido mucho sobre el modus operandi fuera de la Universidad y he quedado satisfecha y contenta con el trabajo realizado en un campo de mi interés como es la Seguridad.

Capítulo 11.

Referencias Bibliográficas

11. 1. DOCUMENTACIÓN

1. **Antonio Fernández**. Artículo: Como crear una librería en Python. Feb 22, 2021. https://antonio-fernandez-troyano.medium.com/crear-una-libreria-python-4e841fbd154f

2. **Auth0.** Introducción a los Tokens web JSON. https://jwt.io/introduction

3. **Azure.** Arquitectura de Microservicios. https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

4. **Compare Django and FastAPI**. https://codeahoy.com/compare/django-vs-fastapi

5. **Django.** https://www.djangoproject.com/

6. **FastAPI**. https://fastapi.tiangolo.com/

7. **Grupo Santander.** Documentación Software Components. Fuente Interna Corporativa

8. **Grupo Santander.** Gravity Meet Up Conference. Fuente Interna Corporativa

9. **Grupo Santander.** BKS Architecture. Fuente interna corporativa

10. **IANA.** JSON Web Token Claims. https://www.iana.org/assignments/jwt/jwt.xhtml

11. **Kendra Mazara (Linkedin).** Curso Arquitectura y Diseño seguros. Junio 2021

12. **Programmerclick.** Introducción a Spring Cloud Gateway. https://programmerclick.com/article/3247149095/

13. **Python.** Python Projects, librerías FastAPI. https://pypi.org/

14. **SCG.** Artículo Spring Cloud Gateway. https://programmerclick.com/article/1050333987/

15. **Sebastián Peyrott**. JWT Handbook. 2016-2018.

16. **Spring.** Documentación Spring Cloud Gateway. https://cloud.spring.io/

17. **Swagger.** OpenAPI specification. https://swagger.io/specification/

18. **Uvicorn.** Documentación. https://www.uvicorn.org/

19. **Wikipedia.** Django. https://es.wikipedia.org/wiki/Django_(framework)

20. **Wikipedia.** FastAPI. https://en.wikipedia.org/wiki/FastAPI

21. **Yugesh Verma.** Django vs Flask vs FastAPI: una guía comparativa de los marcos web de Python. https://analyticsindiamag.com/django-vs-flask-vs-fastapi-a-comparative-guide-to-python-web-frameworks/

11.2. FIGURAS

11.3. TABLAS DE GESTIÓN DE ERRORES