

# Seguridad adicional frente a ciber ataques “*Man-In-The-Middle*”

**Víctor Medina Pérez**  
Grado en Ingeniería Informática  
Ingeniería del Software

**Oriol Martí Girona**  
**Santi Caballe Llobet**

06/2022

©Víctor Medina



Esta obra está sujeta a una licencia de  
Reconocimiento-NoComercial-SinObraDerivada [3.0](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)  
[España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

---

*Quisiera agradecer a mis padres Juan y Yolanda, como a mi hermano Alejandro; y a mi pareja Ana, tanto los ánimos, como la paciencia, comprensión, consejos y fuerzas que me han acompañado día a día durante toda esta etapa de gran esfuerzo y dedicación.*

*Gracias de corazón, sin vosotros, no hubiera llegado hasta aquí.*

---

## FICHA DEL TRABAJO FINAL

<b>Título del trabajo:</b>	<i>Seguridad adicional frente a ciber ataques “Man-In-The-Middle”</i>
<b>Nombre del autor:</b>	<i>Víctor Medina Pérez</i>
<b>Nombre del consultor/a:</b>	<i>Oriol Martí Girona</i>
<b>Nombre del PRA:</b>	<i>Santi Caballe Llobet</i>
<b>Fecha de entrega (mm/aaaa):</b>	06/2022
<b>Titulación:</b>	<i>Grado en Ingeniería Informática</i>
<b>Área del Trabajo Final:</b>	<i>Ingeniería del Software</i>
<b>Idioma del trabajo:</b>	<i>Castellano</i>
<b>Palabras clave</b>	<i>Java</i> <i>Man-In-The-Middle</i> <i>Security</i>
<p><b>Resumen del Trabajo (máximo 250 palabras):</b> <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i></p>	
<p>Hoy en día la gran mayoría de entidades, sin importar su naturaleza u origen, tienden a operar, a través de la nube, con sus servicios.</p> <p>Bajo este contexto, la vulnerabilidad ante los ataques deliberados por delincuentes y hackers han incrementado notablemente el esfuerzo de las empresas por mejorar la seguridad de sus comunicaciones en todos sus aspectos. Uno de estos aspectos es la de proteger el intercambio de información sensible a través de la red lo máximo posible.</p> <p>Uno de los mayores interesados son todas aquellas entidades que, a través de servicios como el de identificación y realización de transacciones, manipulan información muy sensible y, por tanto, necesitan ofrecer el mayor grado de seguridad y confianza a sus clientes.</p> <p>En este trabajo me propongo diseñar, implementar y desplegar una arquitectura de microservicios, en el que uno de ellos exponga un servicio de transacciones online seguras, de forma que cualquier comunicación entre un cliente y nuestro servidor sea segura frente a ciber ataques “Man-In-The-Middle”. Para lograrlo, aplicaremos diferentes acciones como pueden ser el uso de sistemas de criptografía, estándares de estructuras de datos idóneas para comunicaciones seguras y corrección de vulnerabilidades de seguridad en código.</p>	

De esta manera, el objetivo es minimizar el riesgo de sufrir ataques cibernéticos con éxito como pueden ser la interceptación, lectura o modificación de los datos que viajan a través del tráfico de red (conocido como un ataque “Man-In-The-Middle”) Es decir, proteger la integridad y origen de toda la información intercambiada entre dos interlocutores.

**Abstract (in English, 250 words or less):**

Today many entities, regardless of their nature or origin, tend to operate, through the cloud, with their services.

In this context, vulnerability to deliberate attacks by criminals and hackers have notably increased the efforts of companies to improve the security of their communications in all its aspects. One of these aspects is to protect the exchange of sensitive information through the network as much as possible.

One of the main stakeholders are all those entities that, through services such as identification and transaction services, handle highly sensitive information and, therefore, need to offer the highest degree of security and trust to their customers.

In this work I intend to design, implement, and deploy a microservice architecture, in which one of them exposes a secure online transaction service, so that any communication between a client and our server is safe against cyber-attacks "Man-In-The-Middle". To achieve this, we will apply different actions such as the use of cryptography systems, data structure standards suitable for secure communications and correction of security vulnerabilities in code.

In this way, the objective is to minimize the risk of suffering successful cyber-attacks such as the interception, reading or modification of the data that travels through network traffic (known as a "MITM" attack ) That is, protect the integrity and origin of all the information exchanged between two interlocutors.

# Índice

<b>1. Introducción</b> .....	1
<b>1.1 Contexto y justificación del Trabajo</b> .....	1
<b>1.2 Objetivos del Trabajo</b> .....	2
<b>1.3 Enfoque y método seguido</b> .....	3
<b>1.4 Planificación del Trabajo</b> .....	6
<b>1.5 Breve resumen de productos obtenidos</b> .....	7
<b>1.6 Breve descripción de los otros capítulos de la memoria</b> .....	7
<b>2. Initial Planning</b> .....	8
<b>2.1 Initial Planning</b> .....	8
<b>2.2 Product Backlog</b> .....	9
<b>3. Sprint 1</b> .....	11
<b>3.1 Sprint Planning</b> .....	11
<b>3.2 Sprint Review</b> .....	12
HdU : TFG-1 : Análisis de plataformas de pago externas y realizar onboarding.....	12
Análisis del servicio externo Donate API de MasterCard.....	12
Análisis del servicio externo de pagos de PayPal.....	13
Conclusión y elección de plataforma de pagos .....	14
HdU : TFG-3 : Buscar librería java : JSON Web Encryption .....	14
HdU : TFG-5 : Diseño de arquitectura software .....	15
HdU : TFG-7 : Modelado de procesos .....	17
Diagrama global de interacciones.....	17
HdU : TFG-4 : Creación de repositorio GitHub .....	18
<b>3.3 Retrospective</b> .....	18
<b>4. Sprint 2</b> .....	19
<b>4.1 Sprint Planning</b> .....	19
<b>4.2 Analysis, design and build</b> .....	20
Etapa de análisis.....	20
Etapa de diseño .....	20
Etapa de construcción .....	21
<b>4.3 Tests</b> .....	22
<b>4.4 Sprint Review</b> .....	23
HdU : TFG-11 : Implementar servidor de microservicios .....	23
HdU : TFG-6 : Diseño del modelo de datos .....	24
Esquema Main.....	24
Esquema Payments.....	25
HdU : TFG-15 : Crear base de datos SQL Postgres y dos esquemas.....	27
DDL - Creación del esquema Main y sus tablas .....	27
DDL - Creación del esquema Payments y sus tablas.....	28
HdU : TFG-2 : Generación de claves para firma y encriptación de JWE .	30
Creación de clave pública x509 y privada PKCS8 (formato *.der) para la firma de datos.....	31
Creación de par de claves RSA pública y privada (certificado PKCS12) para el cifrado de datos .....	31
HdU : TFG-9 : Implementar microservicio generador de JWE .....	32
Creación de clases entidad: Usuario, Rol y Tarjeta .....	32
Implementar repositorio REST para el CRUD .....	32

Decidir Claims del JWE que generaremos .....	34
Implementar servicio generación JWE con librería Nimbus JOSE + JWT .....	34
<b>4.5 Retrospective</b> .....	35
<b>5. Sprint 3</b> .....	36
<b>5.1 Sprint Planning</b> .....	36
<b>5.2 Analysis, design and build</b> .....	37
Etapa de análisis.....	37
Etapa de diseño .....	37
Etapa de construcción .....	37
<b>5.3 Tests</b> .....	39
<b>5.4 Sprint Review</b> .....	40
HdU: TFG-14 : Implementar servidor de configuración.....	40
Creación de un repositorio GitHub auxiliar dedicado a almacenar las propiedades.....	40
Configurar resto de recursos del sistema como clientes del servidor de configuración .....	41
HdU : TFG-13 : Implementar servidor de recursos (Gateway).....	42
Configuración de los recursos del servidor en formato YML .....	43
HdU : TFG-12 : Implementar servidor de autorización con OAuth 2.0.....	44
Crear cliente REST para comunicarse con el Generador JWE .....	44
Implementar servicio para recuperar Usuarios y JWE en el token .....	45
Configurar contexto de seguridad de Spring Security.....	45
Establecer nuevo Claim para el JWE e incluirlo como información adicional en el token de acceso.....	46
Incluir configuración OAuth2 en el Gateway .....	50
<b>5.5 Retrospective</b> .....	52
<b>6. Sprint 4</b> .....	53
<b>6.1 Sprint Planning</b> .....	53
<b>6.2 Analysis, design and build</b> .....	54
Etapa de análisis.....	54
Etapa de diseño .....	54
Etapa de construcción .....	54
<b>6.3 Tests</b> .....	55
<b>6.4 Sprint Review</b> .....	56
HdU : TFG-17 : Mockear un API que sustituya respuestas del API de MasterCard .....	56
HdU : TFG-8 : Implementar microservicio de pagos .....	56
Crear clases entidad: Donor, Card, Charity, Transaction y One-Time. ....	57
Implementar repositorios para las entidades del esquema Payments. ....	57
Implementar servicio de descifrado del JWE y configuración correspondiente .....	57
Implementar API mockeada como una clase servicio @Service.....	58
Implementar servicio para trabajar con Donors .....	58
Implementar servicio para trabajar con Cards .....	58
Implementar servicio para realizar transacciones del tipo One-Time ..	59
Documentar API de pagos.....	60
HdU : TFG-16 : Simular ciber ataque "Man-In-The-Middle" .....	61
Intercepción de llamada y respuesta del servicio de autorización .....	62
Request: .....	62

Response: .....	62
Intercepción de llamada y respuesta del servicio de creación de Donor .....	63
Request: .....	63
Response: .....	63
Intercepción de llamada y respuesta del servicio de creación de Card .....	64
Request: .....	64
Response: .....	64
Intercepción de llamada y respuesta del servicio de transacción One-Time .....	65
Request: .....	65
Response: .....	65
Interpretación de las intercepciones .....	66
<b>6.5 Retrospective</b> .....	66
<b>7. Conclusiones</b> .....	66
<b>8. Glosario</b> .....	67
<b>9. Bibliografía</b> .....	68
<b>10. Anexos</b> .....	68
Colecciones y entornos de Postman .....	68
Colección API Pagos .....	68
Entorno API Pagos .....	72
Scripts SQL .....	73
Creación de esquema Main .....	73
Creación de esquema Payments .....	73
Inserción de datos iniciales en BD .....	74



## Lista de figuras

<i>Ilustración 1 - Flujo metodología ágil Scrum</i>	3
<i>Ilustración 2 - Plantilla Jira para Historia de Usuario</i>	4
<i>Ilustración 3 - Plantilla Jira para Tareas</i>	5
<i>Ilustración 4 - Diagrama Gantt como "Plan de trabajo"</i>	6
<i>Ilustración 5 - Product Backlog</i>	10
<i>Ilustración 6 - Tablero Sprint 1</i>	11
<i>Ilustración 7 - Diseño de arquitectura</i>	15
<i>Ilustración 8 - Flujo genérico del sistema</i>	17
<i>Ilustración 9 - Evidencia creación de repositorio GitHub</i>	18
<i>Ilustración 10 - Tablero Sprint 2</i>	19
<i>Ilustración 11 - Estructura módulo del microservicio servidor Eureka Server</i>	21
<i>Ilustración 12 - Estructura módulo del microservicio Generador JWE</i>	22
<i>Ilustración 13 – Tarea Jira HdU : TFG-11</i>	23
<i>Ilustración 14 – Tarea Jira HdU : TFG-6</i>	24
<i>Ilustración 15 - Diagrama de clases del esquema Main</i>	24
<i>Ilustración 16 - Modelo datos relacional esquema Main</i>	25
<i>Ilustración 17 - Entidades de MasterCard Donate API</i>	25
<i>Ilustración 18 - Diagrama modelo de clases del esquema Payments</i>	26
<i>Ilustración 19 - Modelo datos relacional esquema Payments</i>	26
<i>Ilustración 20 – Tarea Jira HdU : TFG-15</i>	27
<i>Ilustración 21 – Tarea Jira HdU : TFG-2</i>	30
<i>Ilustración 22 – Tarea Jira HdU : TFG-9</i>	32
<i>Ilustración 23 - Dependencia Maven Spring Data REST</i>	33
<i>Ilustración 24 - Tablero Sprint 3</i>	36
<i>Ilustración 25 - Dependencia Maven db-commons</i>	37
<i>Ilustración 26 - Estructura módulo config-server</i>	38
<i>Ilustración 27 - Estructura módulo gateway-server</i>	38
<i>Ilustración 28 - Estructura módulo oauth-server</i>	39
<i>Ilustración 29 - Estructura módulo db-commons</i>	39
<i>Ilustración 30 – Tarea Jira HdU : TFG-14</i>	40
<i>Ilustración 31 - Repositorio GitHub para propiedades</i>	41
<i>Ilustración 32 - Contenido jwe-generator.properties</i>	41
<i>Ilustración 33 - Dependencia Maven Cloud</i>	41
<i>Ilustración 34 – Tarea Jira HdU : TFG-13</i>	42
<i>Ilustración 35 – Tarea Jira HdU : TFG-12</i>	44
<i>Ilustración 36 - Contenido oauth-server.properties</i>	46
<i>Ilustración 37 - Dependencias Maven Cloud/Security/Config</i>	51
<i>Ilustración 38 - Dependencia Maven Nimbus-JOSE JWT</i>	51
<i>Ilustración 39 - Aviso Jira : informe sprint incompleto</i>	52
<i>Ilustración 40 - Tablero Sprint 4</i>	53
<i>Ilustración 41 - Estructura del módulo payment-service</i>	55
<i>Ilustración 42 – Tarea Jira HdU : TFG-17</i>	56
<i>Ilustración 43 – Tarea Jira HdU : TFG-8</i>	56
<i>Ilustración 44 - Contenido payment-service.properties</i>	58
<i>Ilustración 45 - Detalle servicio "Create Donor"</i>	58
<i>Ilustración 46 - Detalle servicio "Create Card"</i>	59
<i>Ilustración 47 - Detalle servicio "Create OneTime Transaction"</i>	59
<i>Ilustración 48 - Dependencia Maven - OpenApi</i>	60
<i>Ilustración 49 - OpenApi definition de microservicio Pagos</i>	60
<i>Ilustración 50 – Tarea Jira HdU : TFG-16</i>	61
<i>Ilustración 51 - Presentación Postman del API Pagos (A través de Gateway)</i>	61
<i>Ilustración 52 - Wireshark intercepción oauth token request</i>	62
<i>Ilustración 53 - Wireshark intercepción oauth token response</i>	62

<i>Ilustración 54 - Wireshark interceptación create Donor request</i>	63
<i>Ilustración 55 Wireshark interceptación create Donor response</i>	63
<i>Ilustración 56 - Wireshark interceptación create Card request</i>	64
<i>Ilustración 57 - Wireshark interceptación create Card response</i>	64
<i>Ilustración 58 - Wireshark interceptación create One-Time request</i>	65
<i>Ilustración 59 - Wireshark interceptación create One-Time response</i>	65

# 1. Introducción

## 1.1 Contexto y justificación del Trabajo

Hoy en día la gran mayoría de entidades, sin importar su naturaleza u origen, tienden a operar a través de la nube con sus servicios.

Bajo este contexto, la vulnerabilidad ante los ataques deliberados por ciberdelincuentes y hackers han incrementado notablemente los esfuerzos de las empresas por mejorar la seguridad de sus comunicaciones en todos sus aspectos. Uno de estos aspectos, entre otros, es la de proteger el intercambio de información a través de la red lo máximo posible. Es decir, el tráfico de datos entre servidores.

Todas aquellas entidades que, a través de sus servicios, manipulen datos sensibles o confidenciales, requieren de sistemas informáticos que sean capaces de asegurar la emisión o recepción de estos datos de forma segura.

Aquí es donde entran en juego los sistemas de encriptación y los estándares actuales de estructuras de datos para la comunicación entre dos sistemas a través de la red. Estos ofrecen una importante aportación para la protección de datos durante su intercambio. Por ejemplo, esto supondría estar mejor protegido frente a hipotéticos ciber ataques como el famoso “Man-In-The-Middle”, una de las principales amenazas en este tipo de escenarios.

“Man-In-The-Middle” es un ciber ataque que consiste en la interceptación, lectura y posiblemente manipulación de información transmitida entre dos o más interlocutores.

Tal y como afirma el Instituto Nacional de Ciberseguridad, este tipo de ataques son muy peligrosos y difíciles de detectar, ya que uno de los principales objetivos del atacante es evitar ser descubierto.<sup>1</sup>

Este trabajo viene motivado precisamente por la dificultad de detectar un ataque MITM, por lo que tiene por objetivo demostrar cómo el aplicar sistemas de criptografía, autenticación y ciertos estándares de estructuras de datos en la comunicación entre dos sistemas, pueden proteger notablemente el intercambio de información sensible frente a estos ataques.

---

<sup>1</sup> URL: <https://www.incibe.es/protege-tu-empresa/blog/el-ataque-del-man-middle-empresa-riesgos-y-formas-evitarlo> Vista: 04/03/2022

## 1.2 Objetivos del Trabajo

Para poder contextualizar nuestra prueba de concepto, he elegido desarrollar un sistema de microservicios que exponga un API REST dedicado a la realización de transacciones online principalmente.

Es decir, el escenario que queremos representar será uno donde se podría dar la amenaza de un ciber ataque MITM. Consideraremos un hipotético cliente *frontend* (como interlocutor A) que se comunica con nuestro sistema para realizar pagos online a través de un API (un microservicio de pagos anteriormente que identificaremos como interlocutor B)

El interlocutor A, deberá poder autenticarse correctamente en el sistema para poder hacer uso de cualquier recurso, principalmente el de pagos.

El interlocutor B, el que presenta una mayor dificultad técnica, deberá ser capaz de recibir información encriptada, tener capacidad para desencriptarla, procesar la información que necesite de esta y, además, integrarse con algún servicio externo para la simulación de transacciones ficticias.

En ningún momento se debe almacenar información sensible mediante un sistema de almacenamiento de datos. Al contrario, toda esta información sensible no deberá tener estado, tan sólo existir en memoria durante el tiempo de ejecución de las operaciones. Sin embargo, toda aquella información no sensible y necesaria para esta prueba de concepto en este sistema, será almacenada en una BD relacional.

También tendremos que generar las claves/certificados necesarios para llevar a cabo la encriptación, desencriptación y firmas de los datos que vayan a manipular nuestros microservicios. Estas claves deberían estar almacenadas en un sistema de almacenamiento de claves seguro, como podría ser *Azure Key Vault*<sup>2</sup>, pero para este trabajo nos limitaremos a almacenarlos en el equipo local evitando así una mayor complejidad para lograr nuestros objetivos. En un sistema real, estas deberían estar bien protegidas.

De cara a cumplir los principales aspectos de seguridad, en éste sistema deberemos tener al menos:

- Un servidor donde se encontrarán todos los componentes del nuestro sistema
- Un componente a modo de servidor de autorización (por ejemplo, basado en OAuth 2.0)
- Un componente que controle y canalice el acceso a todos los recursos del sistema.
- Un componente que se encargue de proteger los datos que se intercambiarán haciendo uso de un sistema de criptografía y estructuras de datos encriptadas.
- Un microservicio que implemente los servicios de pagos.

---

<sup>2</sup> URL: <https://azure.microsoft.com/es-es/services/key-vault/#product-overview> Vista: 04/03/2022

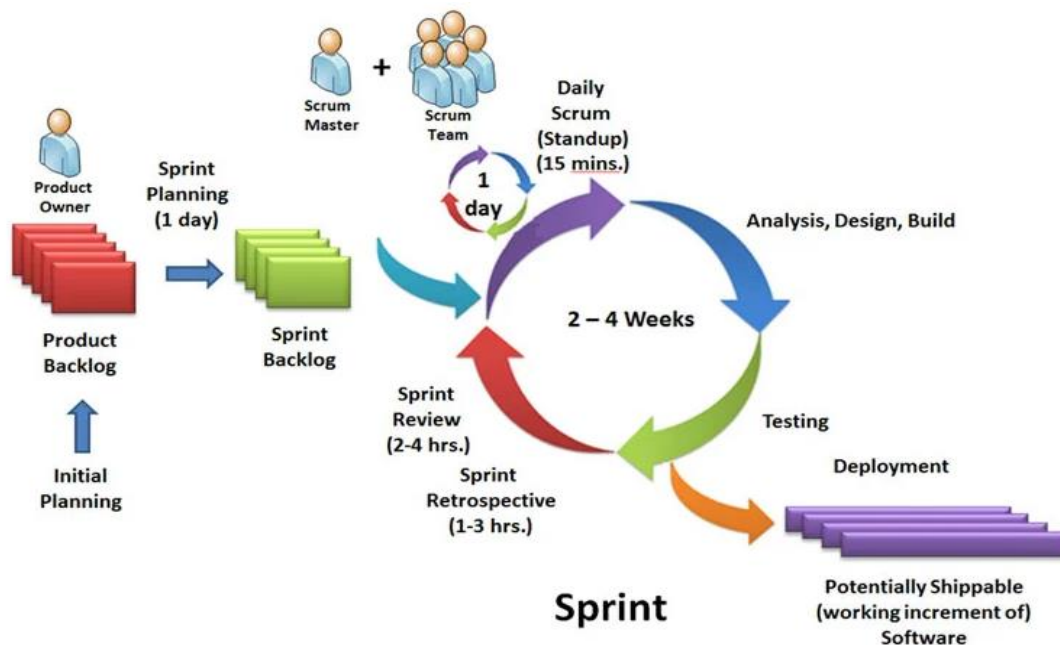
### 1.3 Enfoque y método seguido

La metodología ágil será la que gobernará el desarrollo de todo este proyecto durante todo su ciclo de vida, concretamente la metodología Scrum<sup>3</sup>. Elegimos este enfoque ya que lo que se pretende es tener, desde el principio, un prototipo funcional y entregable en cada iteración.

Debemos tener en cuenta que la metodología Scrum contempla diferentes roles de los cuales al menos tres de ellos son indispensables: *Product Owner*, *Scrum Master* y miembros del equipo de desarrollo. Este trabajo, al ser individual, adquiriremos la responsabilidad de los tres a la vez. Es decir, desarrollaré las funciones de representación del cliente, construcción/seguimiento/priorización de las tareas del *backlog*, liderazgo del equipo de desarrollo y el propio desarrollo del código.

Así pues, dividiremos este trabajo en cinco *sprints*, en el que en cada uno de ellos se subdividirá generalmente en las siguientes fases:

1. Sprint planning
2. Análisis, diseño e implementación
3. Pruebas unitarias
4. Sprint Review
5. Retrospectiva



Copyright © 2011, William B. Heys

Ilustración 1 - Flujo metodología ágil Scrum

<sup>3</sup> URL: [https://platzi.com/blog/metodologia-scrum-fases/?utm\\_source=google&utm\\_medium=paid&utm\\_campaign=14603491644&utm\\_adgroup=&utm\\_content=&gclid=EAlalQobChMIzcqq7te59gIVWYxoCR0AgAOJEAAAYASAAEgJOWvD\\_BwE&gclsrc=aw.ds](https://platzi.com/blog/metodologia-scrum-fases/?utm_source=google&utm_medium=paid&utm_campaign=14603491644&utm_adgroup=&utm_content=&gclid=EAlalQobChMIzcqq7te59gIVWYxoCR0AgAOJEAAAYASAAEgJOWvD_BwE&gclsrc=aw.ds) Vista: 09/03/2022

Por otro lado, utilizaremos dos herramientas de Atlassian, llamada Jira<sup>4</sup>, para poder llevar una gestión organizada de las tareas del backlog y las que formarán parte de nuestros Sprints; junto con Bitbucket<sup>5</sup> para la gestión de nuestro código a través de un repositorio. Ambas en las versiones *standard* gratuitas que ofrecen, ateniéndonos a sus limitaciones.

Las plantillas que utilizaremos para documentar nuestras tareas serán las mismas que ofrece Jira en su plataforma a través de uno de sus servicios que nos permitirá personalizar los tipos de elementos que aparecerán en nuestro tablero.

La siguiente se utilizará para documentar las historias de usuario:

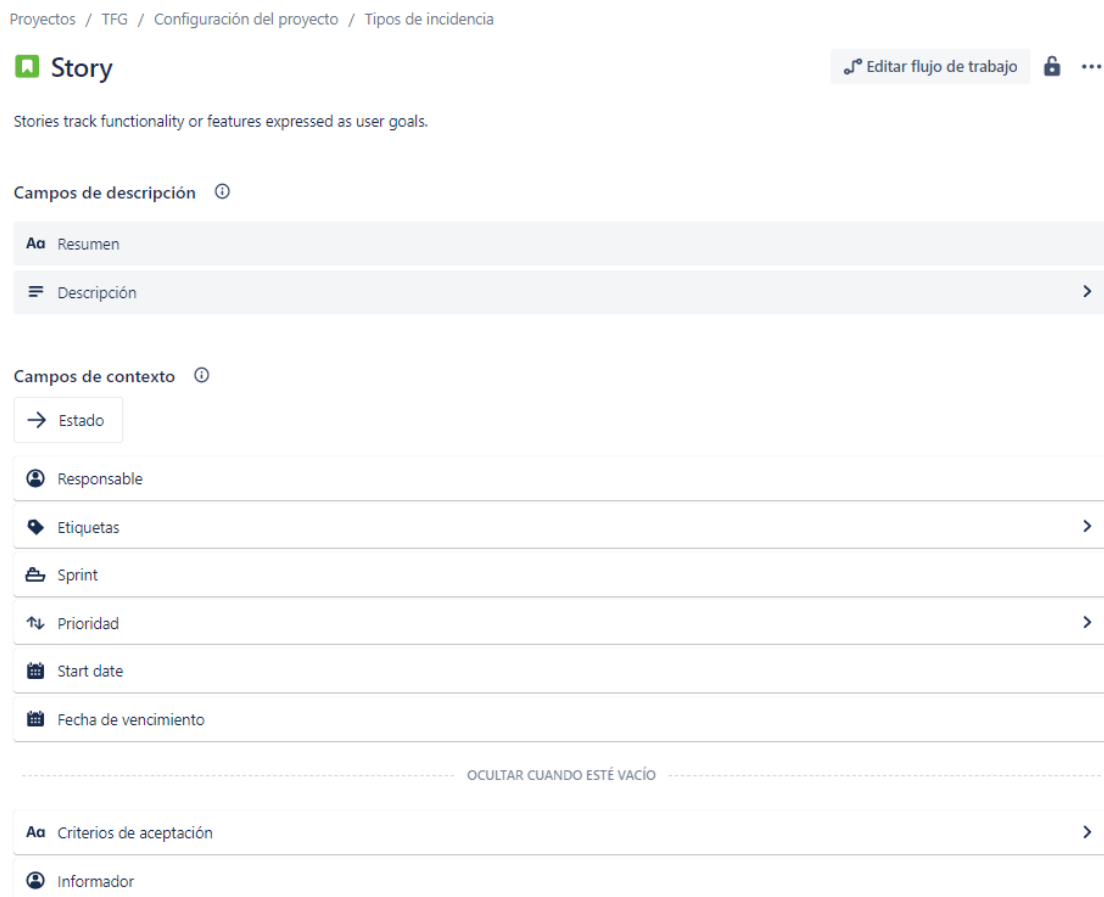


Ilustración 2 - Plantilla Jira para Historia de Usuario

<sup>4</sup> URL: <https://www.atlassian.com/software/jira> Vista: 11/03/2022

<sup>5</sup> URL: <https://bitbucket.org/> Vista: 11/03/2022

Y, por otro lado, las tareas que derivarán de las anteriores historias de usuario:

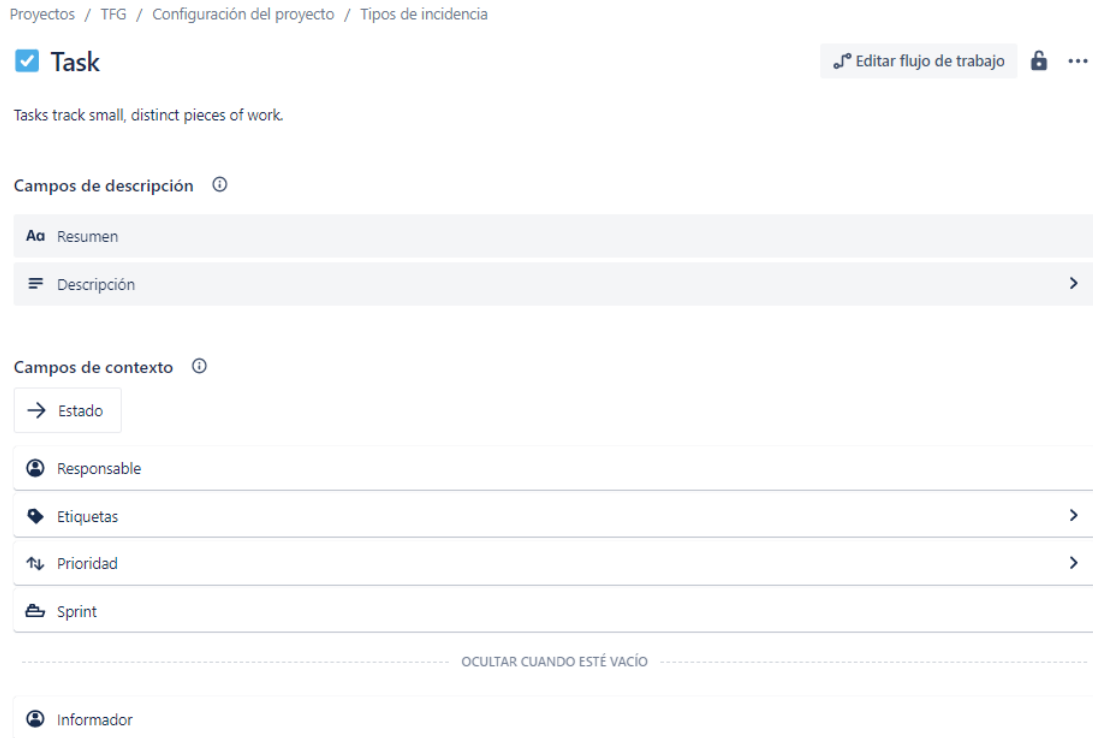


Ilustración 3 - Plantilla Jira para Tareas

Para poder materializar todas las historias de usuario que formarán parte de nuestro *Product Backlog*, tomaremos en cuenta los requerimientos que un supuesto *Product Owner* nos comunicaría durante la primera etapa del proyecto. Generalmente, los principales o más genéricos requerimientos que espera este perfil del producto final suelen documentarse en las etapas iniciales del ciclo de trabajo de la metodología Scrum -etapa conocida como *Initial planning*.

De igual forma, podrán ir apareciendo más requerimientos en cualquier otra etapa del proyecto, característica principal de los proyectos ágiles y su adaptación a los cambios.

Todos estos requerimientos se traducirán digitalmente a Historias de usuario que enriquecerán nuestro *Product Backlog* y de las cuales iremos tomando en cuenta en los tableros *backlog* de cada *sprint*.

## 1.4 Planificación del Trabajo

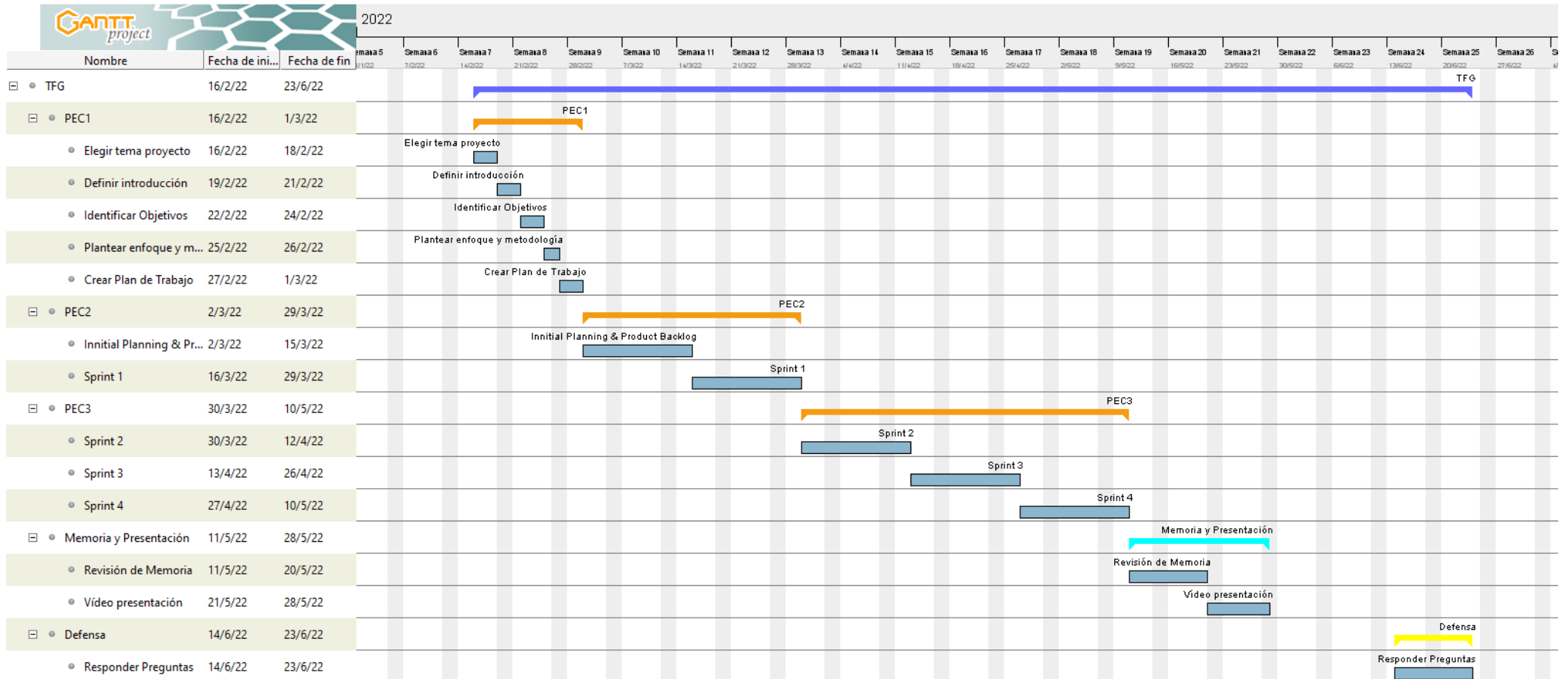


Ilustración 4 - Diagrama Gantt como "Plan de trabajo"



## **1.5 Breve resumen de productos obtenidos**

Prueba de concepto que demuestre una comunicación segura frente a posibles un ciber ataques del tipo “*Man-In-The-Middle*” mediante encriptación de datos entre dos interlocutores.

## **1.6 Breve descripción de los otros capítulos de la memoria**

Descripción por capítulo:

- Capítulo 1
  - Introducción de la memoria, plan de trabajo y enfoque.
- Capítulo 2
  - Initial planning y Product backlog del Proyecto, en la que se definirán los primeros requerimientos transmitidos por el Product Owner y documentados por el Scrum Master.
- Capítulo 3
  - Desarrollo de los Sprints 1, 2, 3 y 4 del proyecto
- Capítulo 7
  - Conclusiones del trabajo y memoria
- Capítulo 8
  - Glosario
- Capítulo 9
  - Bibliografía

## 2. Initial Planning

Como paso previo al primero de los sprints, vamos a recapitular y realizar un primer análisis, conocido como Initial Planning, en el que reflejaremos cómo empezaremos a encauzar este proyecto e identificar los posibles elementos que a priori, van a formar parte de nuestro *Product Backlog*. Este Initial Planning suele realizarse en presencia del *Product Owner* (de aquí en adelante lo identificaremos con las siglas PO) y el *Scrum Master* (que identificaremos de igual manera como SM)

### 2.1 Initial Planning

Tal y como hemos definido en nuestra idea inicial de trabajo “dos interlocutores, A y B, que deben comunicarse de forma segura”, empezaremos a hablar sobre cómo vamos a estructurar los mensajes que intercambiarán entre ellos.

Para trabajar con estructuras de datos encriptadas, existen estándares con los que aseguraremos la protección de los datos. La estructura de datos encriptada que vamos a utilizar en nuestro sistema será del tipo JSON Web Encryption<sup>6</sup> (también conocida por sus siglas como JWE) Implementaremos un microservicio que se encargue de generar y servir estos JWE.

Adicionalmente, necesitaremos utilizar un conjunto de claves para realizar las operaciones de encriptación y desencriptación de datos, así como la firma de estos. Por ejemplo, haremos uso de una clave RSA público-privada para encriptar y desencriptar los datos, garantizando así la protección de estos evitando recibir o enviar información en claro. Por otro lado, también precisaremos de un par de claves simétricas, una privada para el emisor y otra pública para el receptor con el fin de firmar y validar el mensaje, asegurando la autenticidad e integridad de este. Teniendo esto en cuenta, tendremos que generar dichas claves.

Respecto a la realización de transacciones ficticias, contamos con numerosos proveedores de entornos de prueba que se encargan de ofrecer servicios que simulan transacciones online. Entre otros, podemos encontrar la Sandbox de *MasterCard* o la de *PayPal*. Cualquiera de ellas puede adaptarse a las necesidades de éste trabajo. Durante este trabajo, analizaremos ambos proveedores de entornos de prueba y la API que más se adecúe a nuestras necesidades.

Trabajaremos con un lenguaje de programación que nos brinde la oportunidad de crear rápidamente un prototipo funcional de nuestro sistema. Es decir, en el que podamos crear un servidor propio que cuente con servicio de autenticación, control de recursos, servicios atómicos y que nos permita trabajar con facilidad.

En éste caso optaremos el lenguaje de programación orientado a objetos Java. Con este lenguaje podremos hacer uso del framework de *Springboot* y algunos

---

<sup>6</sup> URL: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/json-web-token-jwt/> Vista: 04/03/2022

de sus módulos como *spring-cloud*, *spring-security*, *spring-devtools*, *openfeign*, etc. Con ellos podremos implementar microservicios en muy poco tiempo, servidores como Eureka donde residirán nuestros componentes, un servidor de recursos como *API Gateway* o *Zuul*; y también gracias al framework de *Swagger* podremos documentar nuestro API.

## 2.2 Product Backlog

A modo de sintetizar todas los requisitos mencionados en el apartado [2.1](#), y que serían fruto de numerosas reuniones ficticias entre el *PO* y el *SM*, listamos a continuación los primeros elementos (Historias de Usuario) que van a formar parte de nuestro *Product Backlog* y que posiblemente muchos de ellos se puedan tratar en el primer *sprint*.

- Diseño técnico de arquitectura.
- Diseño del modelo de datos.
- Diagramas UML de los casos de uso.
- Crear repositorio en la plataforma *GIT Hub* para realizar un control de versiones de nuestro código.
- Implementar un microservicio que cuya responsabilidad será la de gestión de todo lo relacionado con los pagos.
- Implementar un microservicio auxiliar cuya responsabilidad será generar *JWE* por cada usuario.
- Implementar un servidor de autorización basado en *OAuth 2.0*
- Implementar un servidor de recursos, que se encargue de gestionar y proteger todos los recursos del servidor y el acceso a estos.
- Implementar un servidor de configuración donde almacenar las claves y valores de configuración más importantes.
- Implementar un servidor para alojar el resto de los microservicios.
- Analizar qué servicios externos se adaptarán mejor para la realización de transacciones ficticias online que queremos llevar a cabo y elegir uno de ellos.
- Averiguar cómo generar nuestras propias claves y o certificados que requerirán los recursos de nuestro sistema para encriptar y desencriptar la información que intercambiarán.
- Buscar una librería java que nos proporcione herramientas y métodos para trabajar con *JWE*.
- Simular un ciber ataque "*Man-In-The-Middle*" contra nuestro sistema.

Teniendo esto en cuenta, nuestro Product Backlog inicial quedará de la siguiente manera:

▼ Backlog (15 incidencias) 0 1 2 [Crear sprint](#)

■ TFG-1 Análisis de plataformas de servicios de pago externas y realizar onboarding	=	TAREAS POR HACER	👤
■ TFG-2 Generación de claves para firma y encriptación de JWE	=	TAREAS POR HACER	👤
■ TFG-3 Buscar librería java : JSON Web Encryption	=	TAREAS POR HACER	👤
■ TFG-4 Creación de repositorios en GIT Hub	=	TAREAS POR HACER	👤
■ TFG-5 Diseño de arquitectura software	=	TAREAS POR HACER	👤
■ TFG-7 Modelado de procesos	=	TAREAS POR HACER	👤
■ TFG-6 Diseño del modelo de datos	=	TAREAS POR HACER	👤
■ TFG-8 Implementar microservicio de pagos	=	TAREAS POR HACER	👤
■ TFG-9 Implementar microservicio generador de JWE	=	TAREAS POR HACER	👤
■ TFG-11 Implementar servidor de microservicios	=	TAREAS POR HACER	👤
■ TFG-12 Implementar servidor de autenticación OAuth 2.0	=	TAREAS POR HACER	👤
■ TFG-13 Implementar servidor de recursos (gateway)	=	TAREAS POR HACER	👤
■ TFG-14 Implementar servidor de configuración	=	TAREAS POR HACER	👤
■ TFG-15 Crear base de datos Postgres SQL con dos esquemas "usuarios + tarjetas" y "pagos"	=	TAREAS POR HACER	👤
■ TFG-16 Simular ciber ataque "Man-In-The-Middle"	=	TAREAS POR HACER	👤

*Ilustración 5 - Product Backlog*

# 3. Sprint 1

## 3.1 Sprint Planning

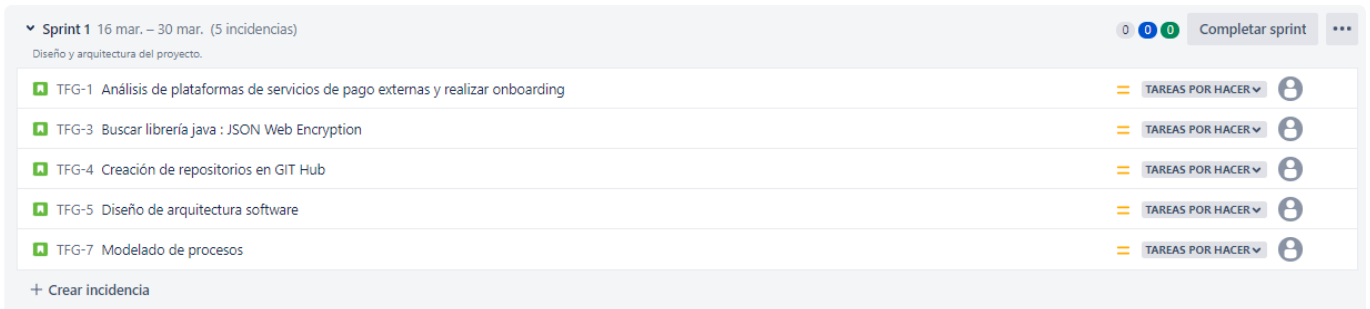


Ilustración 6 - Tablero Sprint 1

Para las siguientes tareas, se recoge la siguiente información:

- HdU : TFG-1 : Análisis de plataformas de servicios de pago externas y realizar onboarding
  - **Descripción:** Buscar y analizar plataformas de pago externas que ofrezcan un entorno de pruebas, así como un proceso de onboarding sencillo.
  - **Criterios de aceptación:** Elegir una plataforma e ir comenzando con el proceso de onboarding.
- HdU : TFG-3 : Buscar librería java : JSON Web Encryption
  - **Descripción:** Buscar una librería java que nos permita trabajar con estructuras de datos JWE fácilmente.
  - **Criterios de aceptación:** Debe ser una librería que nos ofrezca, además, una pequeña guía de cómo usarla para facilitarnos su implementación en nuestro código.
- HdU : TFG-4 : Creación de repositorios GitHub
  - **Descripción:** Utilizar GitHub para la creación de un monorepositorio<sup>7</sup>.
  - **Criterios de aceptación:** Obtener un espacio donde poder subir nuestro código y realizar un correcto control de versiones de este.
- HdU : TFG-5 : Diseño de arquitectura software
  - **Descripción:** Diseño de arquitectura software de nuestro proyecto
  - **Criterios de aceptación:** Obtener un diseño técnico de todo el sistema que tenemos como objetivo.
- HdU : TFG-7 : Modelado de procesos
  - **Descripción:** Crear un diagrama de interacción global de nuestro sistema.
  - **Criterios de aceptación:** Este diagrama debe representar las interacciones tanto en la autenticación, generación del JWE como la recepción de cualquier petición de pagos.

<sup>7</sup> URL: <https://kinsta.com/es/blog/monorepo-vs-multi-repo/> Visitada: 20/03/2022

## 3.2 Sprint Review

HdU : TFG-1 : Análisis de plataformas de pago externas y realizar onboarding

En la elección de un servicio externo para realizar transacciones ficticias, vamos a llevar a cabo una búsqueda de principales plataformas de pago que ofrezcan entornos de prueba a desarrolladores.

Para determinar cuál de ellas será idónea, se ha decidido tomar como referencia los siguientes puntos clave y su respectiva tabla de valoración. La escala de valoración que se usará es una puntuación comprendido entre [0-5], en la que 0 es la peor valoración y 5 la máxima:

Requisito	Valoración [0-5]
Sencillez de integración	
Puesta en marcha	
Facilidades y soporte	
Servicio de pagos	

Tabla 1 - Valoración de servicios externos de pagos

### *Análisis del servicio externo Donate API de MasterCard*

*MasterCard* ofrece una serie de API de diferentes características en su portal de desarrolladores<sup>8</sup>. Entre todas ellas he decidido analizar, por su sencillez, facilidades y uso de comunicación encriptada, los servicios del API *MasterCard Donations*<sup>9</sup>.

Tal y como podemos observar en su portal, ponen a disposición de todo aquel desarrollador que quiera integrar sus servicios, una serie de tutoriales<sup>10</sup> muy intuitivos y que facilitan enormemente esta tarea.

En relación con la comunicación encriptada, nos ofrecen la posibilidad de proporcionarnos claves de encriptación, desencriptación y autenticación.

Como contra, su puesta en marcha no es inmediata ya que debemos enviar un correo al equipo de soporte de la API para que realicen nuestro *onboarding* en el entorno de Sandbox.

Todos estos puntos nos ayudan a tomar la decisión de elegir esta como servicio externo para la realización de transacciones ficticias.

<sup>8</sup> URL: <https://developer.mastercard.com/> Visitada: 20/03/2022

<sup>9</sup> URL: <https://developer.mastercard.com/donations/documentation> Visitada: 20/03/2022

<sup>10</sup> URL: <https://developer.mastercard.com/donations/documentation/tutorials/> Visitada: 20/03/2022

Tras su análisis haremos uso de la tabla 1 para obtener la valoración final de la plataforma de *MasterCard* y su *Donate API*:

Requisito	Valoración [0-5]
Sencillez de integración	4
Puesta en marcha	2
Facilidades y soporte	5
Servicio de pagos	4

#### *Análisis del servicio externo de pagos de PayPal*

*PayPal* también ofrece una serie de API a través de su portal de desarrolladores<sup>11</sup>. En ella vemos que ofrece servicios de pagos, concretamente en la creación y autorización de pagos en su versión gratuita para desarrolladores, sin embargo, esta requiere obligatoriamente el uso de otra API más llamada *Orders API* para que la funcionalidad sea completa. Además, en este caso no tenemos opción a usar medidas adicionales de seguridad en la comunicación con *PayPal* como pueden ser la encriptación de datos.

La puesta en marcha parece inmediata, no es necesario escribir a ningún equipo de soporte para realizar el onboarding. Por otro lado, vemos que el portal no es muy intuitivo, la navegación entre las páginas de documentación sobre la API es pesada además de la información que podemos obtener sobre los contratos de las API, a priori, poco estructurada.

Sobre el soporte, vemos que también nos ofrecen una forma de contacto con algún equipo de soporte genérico<sup>12</sup>.

Tras el análisis de los servicios externos que ofrece *PayPal*, se valora de la siguiente manera a través de la tabla 1:

Requisito	Valoración [0-5]
Sencillez de integración	3
Puesta en marcha	5
Facilidades y soporte	3
Servicio de pagos	2

<sup>11</sup> URL: <https://developer.paypal.com/api/rest/> Visitada: 20/03/2022

<sup>12</sup> URL: <https://developer.paypal.com/docs/support/> Visitada 20/03/2022

## Conclusión y elección de plataforma de pagos

Finalmente, tomando en cuenta los resultados de valoración en las tablas de los apartados 3.2.1 y 3.2.2, se ha decidido usar como servicio externo los servicios ofrecidos por *MasterCard* y su API de donaciones *Donate API*.

HdU : TFG-3 : Buscar librería java : JSON Web Encryption

Tras una búsqueda de librerías que nos permitan trabajar con estructuras de datos JWE en nuestros microservicios, se ha decidido utilizar la librería *Nimbus JOSE + JWT*<sup>13</sup>.

En su web nos indican que, con esta librería, podemos crear, serializar y procesar tanto *JWS*, *JWE* y *JWT*. Aseguran que cubren todos los algoritmos estándar de estas estructuras de datos e implementan, concretamente para los JWE, las especificaciones del estándar *RFC-7516 JSON Web Encryption (JWE)*<sup>14</sup>

Además, ponen a disposición de todos numerosos ejemplos<sup>15</sup> para trabajar con su librería en diferentes escenarios.

Como añadido, esta librería nos ofrece también una solución para integrar una autenticación segura mediante *JWT* y *OAuth 2.0*<sup>16</sup> muy interesante y que se podría incluir en nuestro trabajo, pero para la autenticación y generación de JWT trabajaremos con la propia dependencia del módulo de *Spring*, *spring-security*.

---

<sup>13</sup> URL: <https://connect2id.com/products/nimbus-jose-jwt> Visitada: 21/03/2022

<sup>14</sup> URL: <https://datatracker.ietf.org/doc/html/rfc7516> Visitada: 21/03/2022

<sup>15</sup> URL: <https://connect2id.com/products/nimbus-jose-jwt/examples> Visitada: 21/03/2022

<sup>16</sup> URL: <https://connect2id.com/products/nimbus-jose-jwt/examples/validating-jwt-access-tokens>  
Visitada: 21/03/2022



HdU : TFG-5 : Diseño de arquitectura software

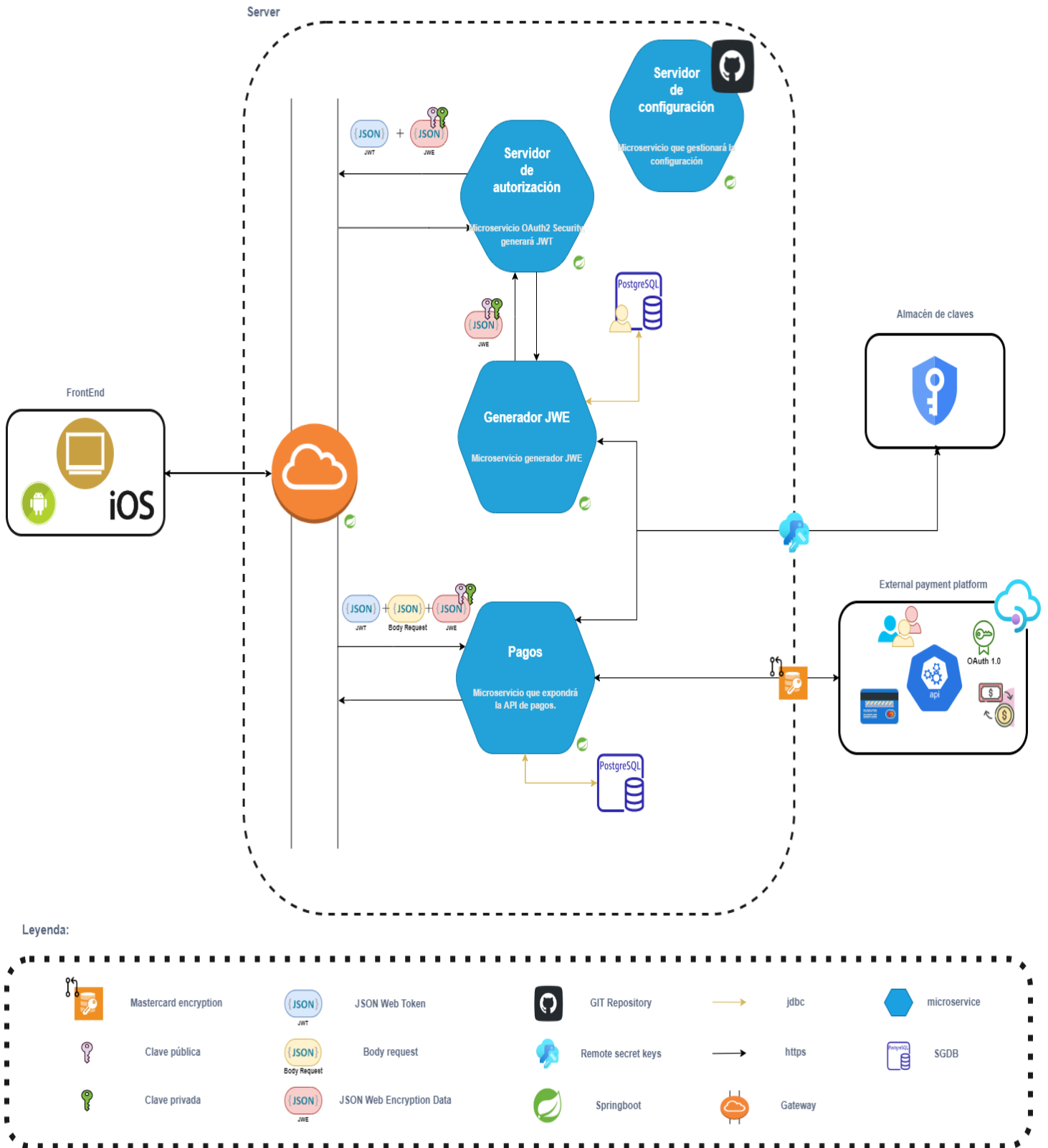


Ilustración 7 - Diseño de arquitectura

Tal y como se ha planteado este proyecto, se muestra en el diseño de arquitectura las tecnologías que vamos a emplear, cómo se estructurarán los diferentes componentes del sistema y las comunicaciones entre ellos.

Las comunicaciones, en un entorno real, deberán cumplir el protocolo *HTTPS*<sup>17</sup> que, mediante el cifrado de los datos con el protocolo *SSL/TLS*<sup>18</sup>, añadiría un grado más de seguridad en el intercambio de información. También sería recomendable el uso de los servicios de *Azure Key Vault* o similares con los que se podrían proteger nuestras claves y certificados en un almacén de claves. En nuestro caso, para simplificar el trabajo y evitar costes económicos durante su desarrollo, prescindiremos de estos dos últimos requisitos y usaremos comunicaciones *HTTP*, así como el almacenamiento de las claves en nuestro entorno local.

Como podemos ver, el intercambio de datos que se producirá entre el *frontend* (interlocutor A) y el microservicio de pagos (interlocutor B) se enviarán los datos no sensibles a través del cuerpo de las peticiones y, por otro lado, los datos sensibles a través de un *header*<sup>19</sup> que llamaremos “*JWE Data*” y cuyo contenido será un *JSON Web Encryption*.

Siempre que añadamos una cabecera customizada, debemos asegurarnos de que respetamos el estándar oficial *RFC-7230 HTTP/1.1 Message Syntax and Routing*<sup>20</sup>

Este *JWE* vendrá dado como información adicional dentro del *JWT* generado en servidor de autorización y que habrá sido creado dentro del microservicio generador de *JWE*. El microservicio de pagos será capaz de descifrar y verificar su contenido al recibirlo.

En las comunicaciones que mantendremos con el servicio externo que hemos elegido de *MasterCard* haremos uso de las claves de encriptación y descifrado que nos han proporcionado durante su proceso de *onboarding* en el entorno de pruebas *Sandbox* y en el cual nos autenticaremos mediante *OAuth 1.0*

De esta forma, durante todas las comunicaciones externas a nuestro sistema nunca viajarán datos sensibles en claro y desprotegidos. Este es el punto fuerte frente a ataques *MITM*.

Por último, para persistir los datos que fueran necesarios almacenar, utilizaremos una base de datos relacional *Postgres SQL*, como serán la de los usuarios y sus tarjetas por un lado en un esquema que llamaremos “*main*”; y toda la información intercambiada con el proveedor externo de pagos (*MasterCard* en éste caso) en otro esquema diferente que llamaremos “*payments*”. El SGBD podría ser cualquier otro de similares características.

---

<sup>17</sup> URL: <https://developers.google.com/search/docs/advanced/security/https?hl=es> Visita: 20/03/2022

<sup>18</sup> URL: <https://revista.seguridad.unam.mx/numero-10/el-cifrado-web-ssl/tls> Visita: 20/03/2022

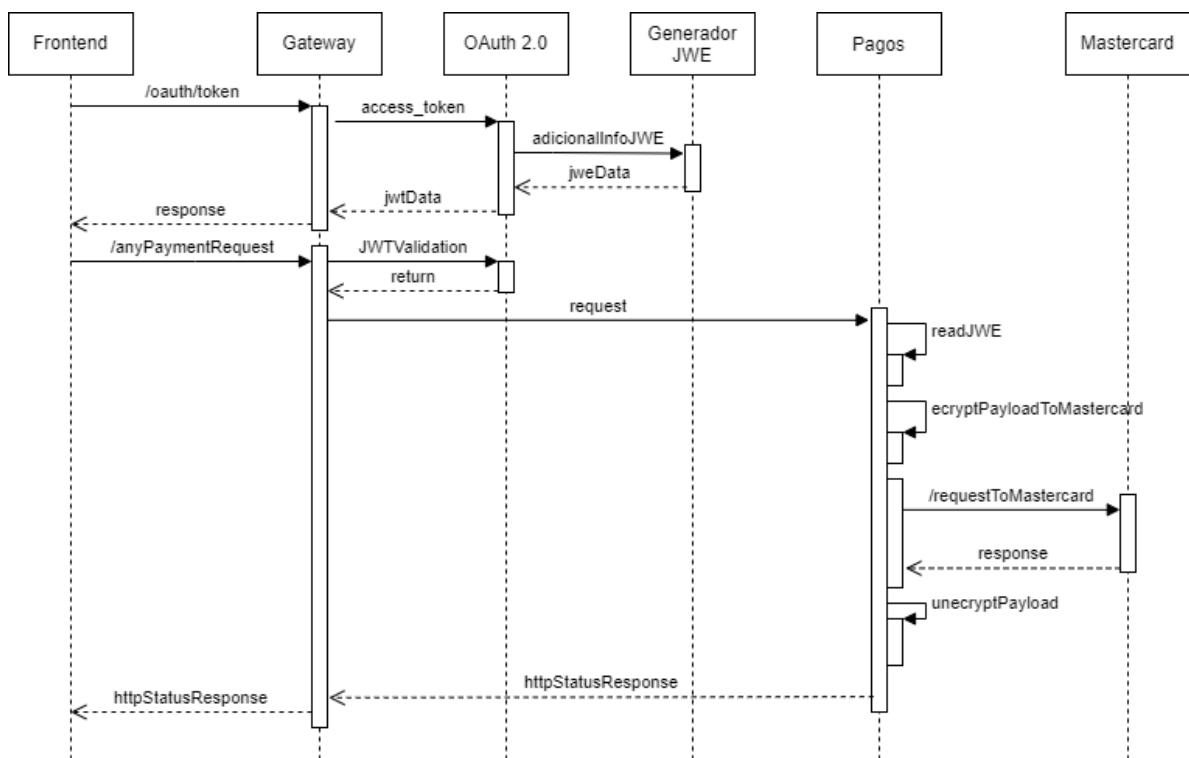
<sup>19</sup> URL: <https://developer.mozilla.org/es/docs/Web/HTTP/Headers> Visitada: 20/03/2022

<sup>20</sup> URL: <https://datatracker.ietf.org/doc/html/rfc7230#section-1.1> Visitada: 20/03/2022

Dado que los dos componentes que vamos a desarrollar son muy simples, y el propio diseño de arquitectura software del [apartado 3.5](#) ya representa con claridad la estructura estática de nuestro sistema, vamos a centrar los esfuerzos en modelar lo verdaderamente interesante de este trabajo, su comportamiento y procesos.

Es decir, queremos representar cómo los dos componentes van a interactuar durante su comunicación y qué procesos van a llevar a cabo para el intercambio de información segura.

*Diagrama global de interacciones*



*Ilustración 8 - Flujo genérico del sistema*

HdU : TFG-4 : Creación de repositorio GitHub

Vamos a realizar un control de versiones de nuestro código con la herramienta *GitHub*. Elegimos esta por proximidad y experiencia previa en su uso.

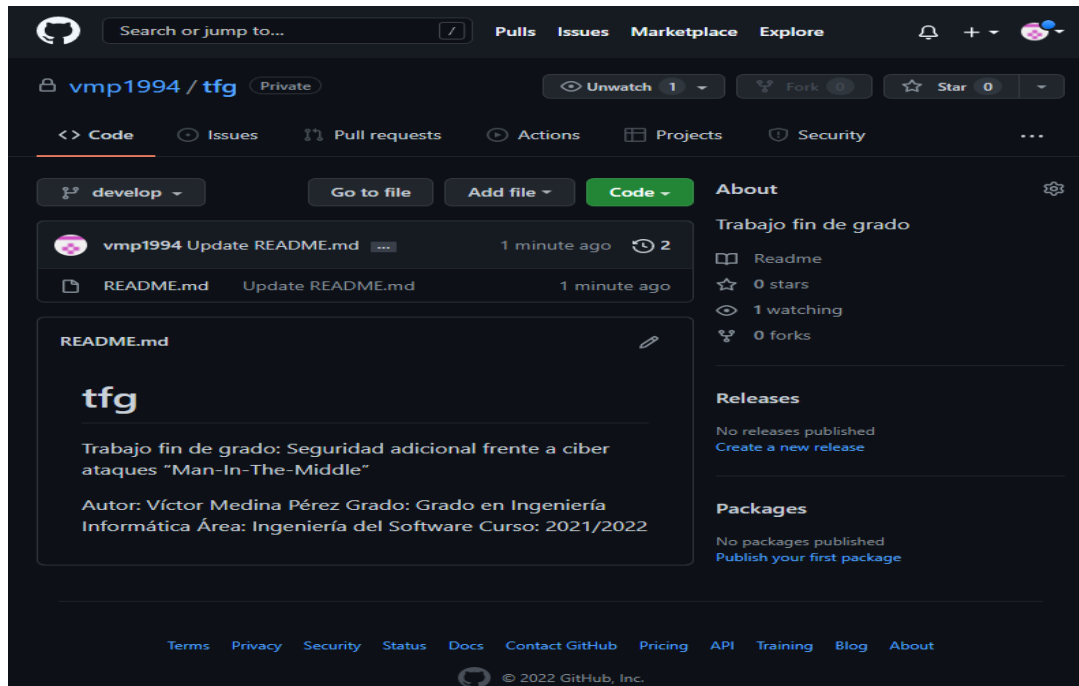


Ilustración 9 - Evidencia creación de repositorio GitHub

La URL de nuestro repositorio es: <https://github.com/vmp1994/tfg.git>

### 3.3 Retrospective

Durante el desarrollo de este primer *sprint* se han realizado varias tareas, pero ninguna que requiera un desarrollo. Es por ello por lo que no se ha llevado a cabo un paso intermedio de "Análisis, diseño e implementación" además de pruebas unitarias.

Por otro lado, tampoco ha sido necesaria la intervención del rol *PO* para la realización de un refinamiento.

Como puntos a destacar, se ha dado un pequeño contratiempo en lo que se refiera al análisis y búsqueda de una plataforma externa de pago ya que, a pesar de haber escrito en repetidas ocasiones al equipo de soporte de *MasterCard*, aún no hemos recibido respuesta. Esto no supone mayor problema para la realización de esta prueba de concepto ya que, aunque forme parte del diseño el incluir un servicio de pago, no es imprescindible para cumplir nuestro objetivo. Se podrá realizar un *mock*<sup>21</sup> de las respuestas que deberíamos recibir por parte de *MasterCard* a través del microservicio de Pagos. Para ello se investigará cómo *mockear* un API<sup>22</sup> fácilmente.

<sup>21</sup> URL: [https://es.wikipedia.org/wiki/Objeto\\_simulado](https://es.wikipedia.org/wiki/Objeto_simulado) Visitada: 27/03/2022

<sup>22</sup> URL: <https://medium.com/@marconolascor/puedes-mockear-las-apis-para-avanzar-839415e208ba> Visitada: 27/03/2022

# 4. Sprint 2

## 4.1 Sprint Planning

Proyectos / TFG

### Sprint 2

Implementar servidores de microservicios, recursos y autorización. Además, se pretende definir en éste ya el modelo de datos y su respectiva BD.

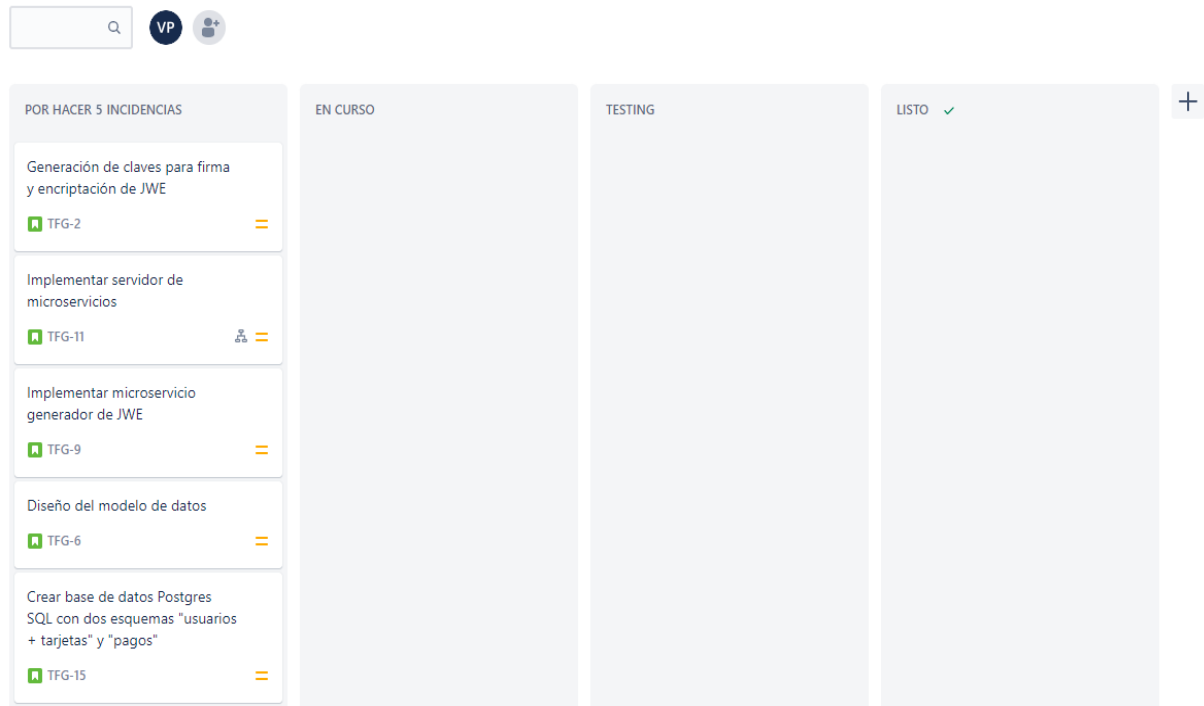


Ilustración 10 - Tablero Sprint 2

Para las siguientes tareas, se recoge la siguiente información:

- HdU : TFG-2 : Generación de claves para firma y encriptación de *JWE*
  - **Descripción:** Se pretende conseguir un certificado simétrico de clave pública/privada para la desencriptación y validación de firma de los datos del *JWE* y también las claves correspondientes para poder encriptar y firmar estos datos.
  - **Criterios de aceptación:** Conseguir un par de claves con un certificado del tipo *PKCS12*; y dos claves, una pública *X509* y otra privada *PKCS8* que pueda leer una aplicación Java.
- HdU : TFG-9 : Implementar microservicio generador de *JWE*
  - **Descripción:** Implementación de un microservicio capaz de generar un *JWE* con todos los datos de las tarjetas de un usuario en concreto, además deberá exponer un *endpoint* para recuperar información de los usuarios en función de su identificación.
  - **Criterios de aceptación:** Servicio de consulta de datos de un usuario y generación de un *JWE* con los datos de las tarjetas del usuario. La generación del *JWE* deberá realizarse mediante la librería de *Nimbus JOSE+JWT* mencionado en el [HdU : TFG-3](#)

- HdU : TFG-11 : Implementar servidor de microservicios
  - **Descripción:** Se pretende implementar un microservicio, a modo de servidor de registro de microservicios que encontramos en nuestro diseño.
  - **Criterios de aceptación:** Un servidor de registro de microservicios con Springboot.
  
- HdU : TFG-6 : Diseño del modelo de datos
  - **Descripción:** Diseño del modelo de datos relacionado con las entidades “usuario”, “rol” y “tarjeta” del esquema “Main” y también de las entidades “Donor”, “Donation”, “One-Time Donation”, “Monthly Donation”, “Guest Donation”, “Charity” y “Card” del esquema “Payments”. El esquema “Main” lo usaremos con el posterior microservicio Generador *JWE*.
  - **Criterios de aceptación:** Diagramas en formato *png*.
  
- HdU : TFG-15 : Crear base de datos SQL Postgres y dos esquemas
  - **Descripción:** Crearemos una base de datos local con dos esquemas diferenciados, el primero de ellos destinado al sistema de autorización y almacenamiento de tarjetas que se llamará “Main”; y el segundo esquema para los datos relacionados con los pagos y la plataforma externa de datos que se llamará “Payments”.
  - **Criterios de aceptación:** Las propias bases de datos con las tablas señaladas en el modelo de datos.

## 4.2 Analysis, design and build

### Etapa de análisis

- Con relación a la HdU : TFG-15
  - Se ha decidido utilizar una base de datos *Postgres SQL* relacional, ya que los datos son de naturaleza financiera.
- Con relación a la HdU : TFG-2 y las claves...
  - Claves de encriptación -> Se va a utilizar un certificado *PKCS12* ya que esto facilita a los recursos de nuestro sistema utilizar un único archivo.
  - Claves de firma -> Se ha decidido utilizar dos archivos diferentes para almacenar la clave pública y privada. Esto es así porque en un entorno real, en una comunicación bilateral, tan sólo el firmante debe tener su clave privada para asegurar la “autoría de los datos” y el receptor de los datos la clave pública para “validar la autoría de los datos”. En éste caso, como los datos son creados por nuestro sistema desde el “Generador *JWE*” y también validados por otros recursos del sistema, podríamos haber utilizado otro certificado *PKCS12*.

### Etapa de diseño

Se ha utilizado el diseño propio de microservicios con tecnología *Springboot*. Es

decir, principalmente utilizando el contexto de aplicación *Springboot* y el contenedor de beans propio.

Como buenas prácticas se ha seguido un método de programación *SOLID* durante la implementación de todas las clases de los microservicios, entre otras cosas, por ejemplo, se han aplicado las ventajas de la inyección de dependencias.

Etapas de construcción

El módulo del proyecto correspondiente al microservicio servidor de aplicaciones “eureka-server” tiene la siguiente estructura:

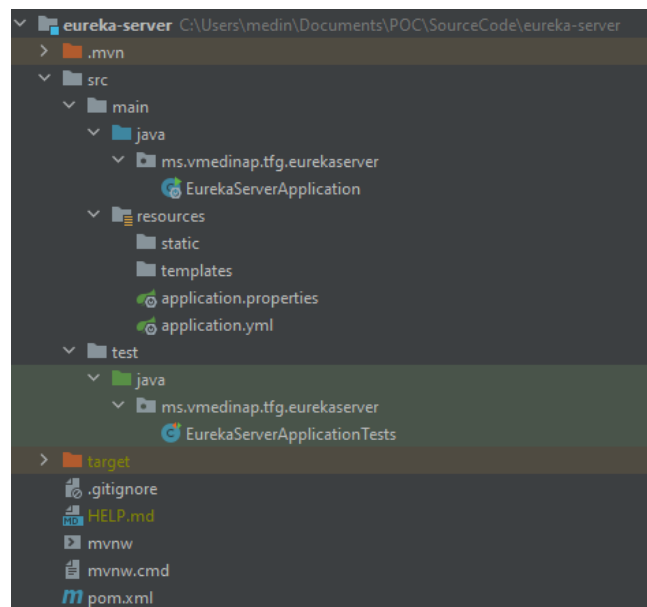


Ilustración 11 - Estructura módulo del microservicio servidor Eureka Server

El módulo del proyecto correspondiente al microservicio Generador *JWE* “jwe-generator” tiene la siguiente estructura:

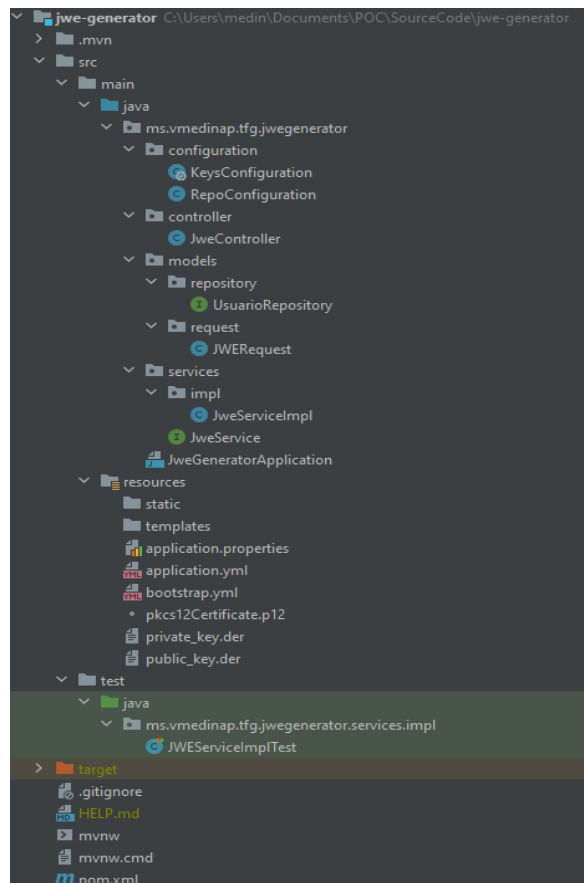


Ilustración 12 - Estructura módulo del microservicio Generador *JWE*

### 4.3 Tests

Se han realizado llamadas a los tres nuevos componentes del sistema a través de un cliente Postman para comprobar el correcto funcionamiento de:

- La configuración del nuevo servidor de recursos *Spring Cloud Gateway* y acceso al resto de recursos del sistema.
- Correcta comunicación y recogida de valores subidos al servidor de configuración en el archivo *properties*.
- Se ha implementado una clase test llamada *JWEServiceImplTest* para poder probar los servicios relacionados con la manipulación de *JWE*.



## 4.4 Sprint Review

HdU : TFG-11 : Implementar servidor de microservicios

### Implementar servidor de microservicios

Adjuntar Añadir una incidencia secundaria Vincular incidencia

#### Descripción

HdU : TFG-11 : Implementar servidor de microservicios

- **Descripción:** Se pretende implementar un microservicio, a modo de servidor de registro de microservicios que encontramos en nuestro diseño.
- **Criterios de aceptación:** Un servidor de registro de microservicios con Springboot.

#### Incidencias secundarias

Ordenar por

100 % hecho

TFG-18	Generar JAR inicial con Spring Initializr	FINALIZADA
TFG-19	Configurar application.yml inicial del servidor Eureka	FINALIZADA

Ilustración 13 – Tarea Jira HdU : TFG-11

Tal y como nos ofrece *Springboot*, para crear un microservicio a modo de servidor, podemos aprovechar uno de sus múltiples módulos; en éste caso el módulo de *spring-cloud*.

Con éste podremos descargarnos la dependencia cuyo artefacto es *spring-cloud-starter-netflix-eureka-server*. Este nos permitirá implementar un servidor de registro de microservicios conocido como *Spring Cloud Eureka Server*<sup>23</sup>.

Este tipo de servidor está enfocado para trabajar con arquitecturas orientadas a microservicios, por lo que lo convierte en el idóneo para esta prueba de concepto.

Nos permite tener un registro y control de todos los recursos con los que van a componer el sistema de forma rápida y sencilla, además de un balanceador de carga que, en un sistema real, ayudaría a ofrecer una mayor disponibilidad de nuestro servicio. Es decir, podríamos tener más instancias de nuestros microservicios para poder distribuir un pico de carga de peticiones recibidas entre todas ellas. Este balanceador de carga se conoce como *load balancer*, que es el propio de *Spring Cloud*.

<sup>23</sup> URL: <https://blog.bi-geek.com/arquitecturas-spring-cloud-netflix-eureka/> Visitada: 05/04/2022

## HdU : TFG-6 : Diseño del modelo de datos

Proyectos / TFG / Añadir epic / TFG-6

### Diseño del modelo de datos

Adjuntar Añadir una incidencia secundaria Vincular incidencia

#### Descripción

HdU : TFG-6 : Diseño del modelo de datos

- Descripción:** Diseño del modelo de datos relacionado con las entidades "usuario", "rol" y "tarjeta" del esquema "Main" y también de las entidades "Donor", "Donation", "One-Time Donation", "Monthly Donation", "Guest Donation", "Charity" y "Card" del esquema "Payments". El esquema "Main" lo usaremos con el posterior microservicio Generador JWE.
- Criterios de aceptación:** Diagramas en formato png.

#### Archivos adjuntos ((4))



#### Incidencias secundarias

Ordenar por

100 % hecho

FFG-33	Esquema Main	FINALIZADA
FFG-34	Esquema Payments	FINALIZADA

Ilustración 14 – Tarea Jira HdU : TFG-6

### Esquema Main

Para este sprint, necesitaremos utilizar al menos tres entidades: usuarios, roles y tarjetas.

Las dos primeras se utilizarán para llevar a cabo el servicio de autenticación. Se establecerá una relación *ManyToMany* entre los usuarios y los roles, muchos usuarios podrán tener muchos roles.

Por otro lado, también estableceremos una relación *OneToMany* donde un usuario podrá tener ninguna o muchas tarjetas.

El modelo de datos quedará entonces de la siguiente manera, representado primero por un diagrama de clases y posteriormente relacional:

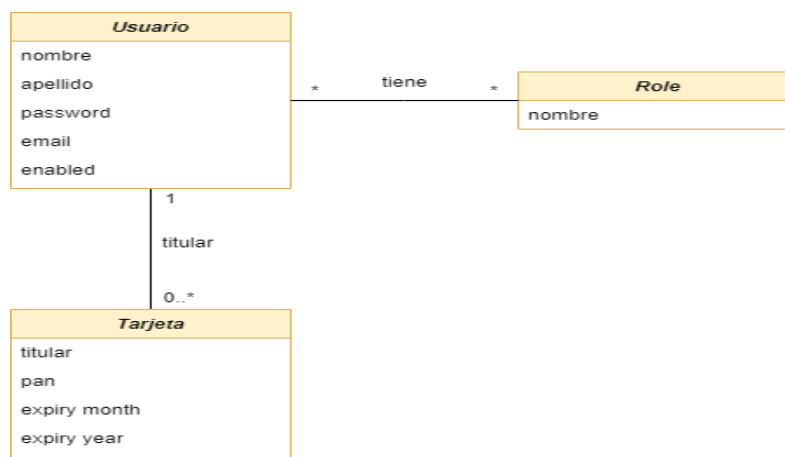


Ilustración 15 - Diagrama de clases del esquema Main

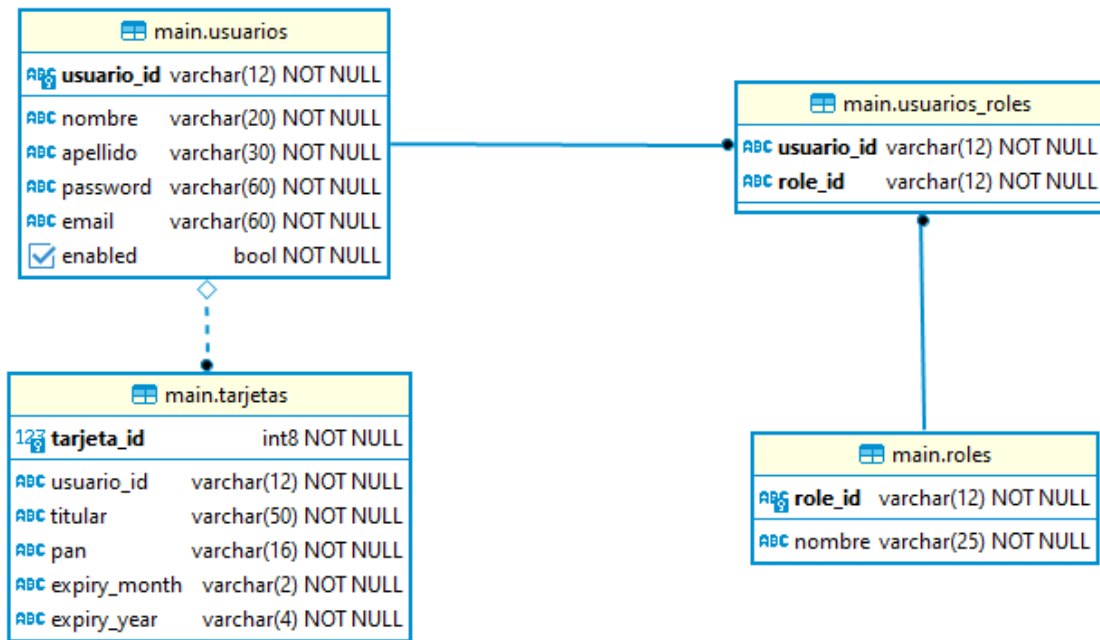


Ilustración 16 - Modelo datos relacional esquema Main

### Esquema Payments

Por otro lado, para almacenar todos los datos relacionados con los pagos realizados en una plataforma externa debemos revisar la documentación del *API de MasterCard Donate API*.

Empezando por el enrolamiento de clientes a la plataforma de *MasterCard*, podemos identificar tres nuevas entidades: *Donor* (donante), *Card* (tarjeta) y *Donation* (donación).

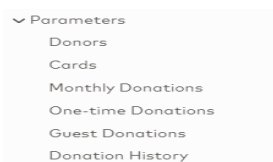


Ilustración 17 - Entidades de MasterCard Donate API

Como debemos hacer una relación entre este esquema, y el principal, haremos que la entidad *donor* tenga información sobre la identificación de la entidad usuario del esquema “*Main*”.

Dicho esto, representaremos a continuación los diagramas del modelo de clases y datos.

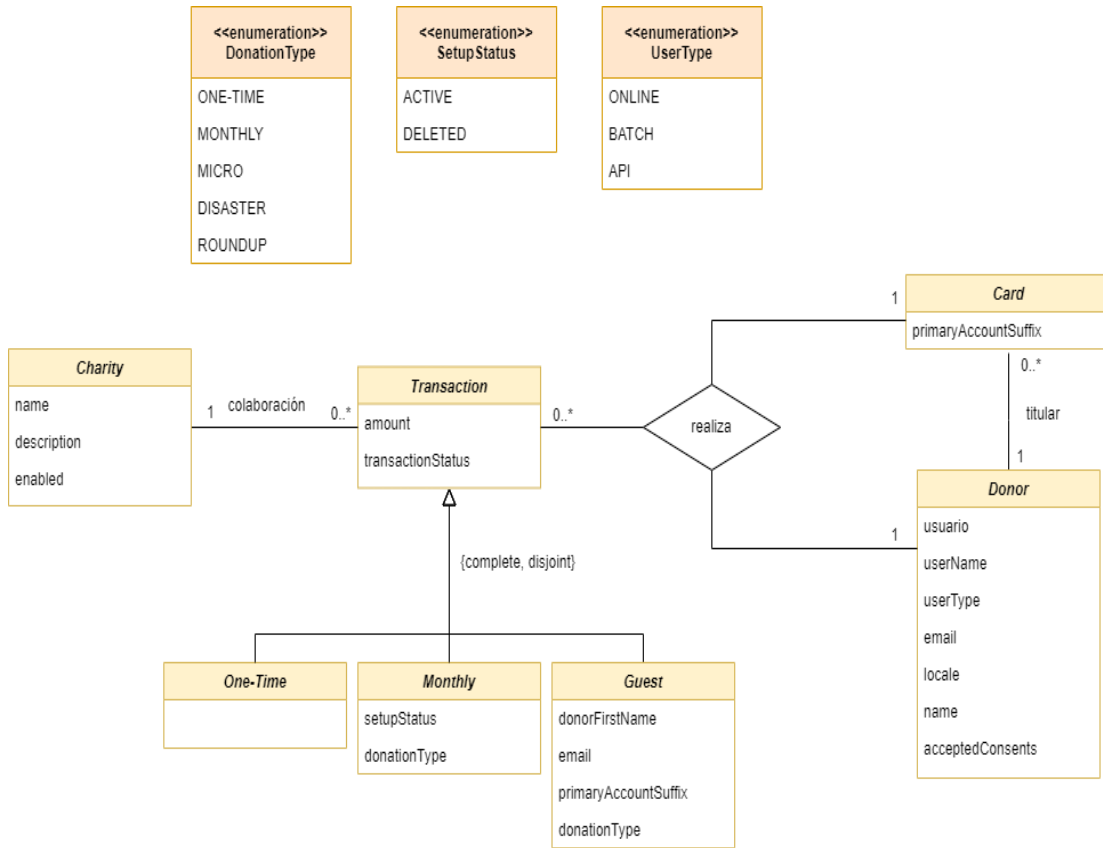


Ilustración 18 - Diagrama modelo de clases del esquema Payments

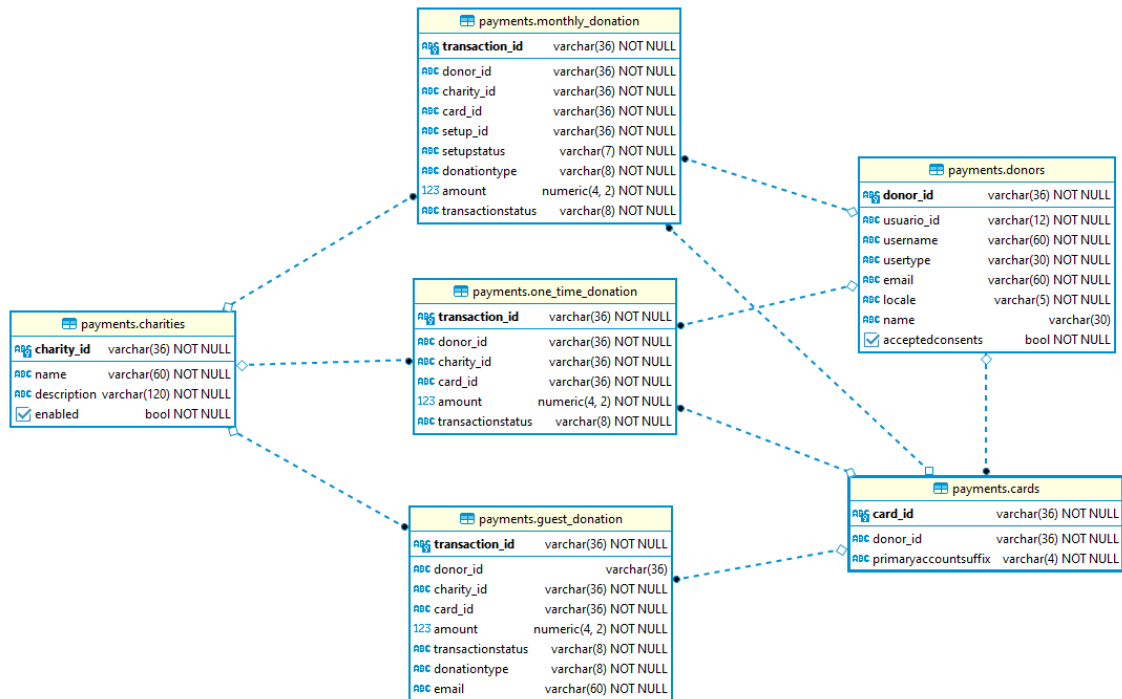


Ilustración 19 - Modelo datos relacional esquema Payments

## HdU : TFG-15 : Crear base de datos SQL Postgres y dos esquemas

Proyectos / TFG / Añadir epic / TFG-15

### Crear base de datos Postgres SQL con dos esquemas "usuarios + tarjetas" y "pagos"

Adjuntar Añadir una incidencia secundaria Vincular incidencia

#### Descripción

HdU : TFG-15 : Crear base de datos SQL Postgres y dos esquemas

- **Descripción:** Crearemos una base de datos local con dos esquemas diferenciados, el primero de ellos destinado al sistema de autorización y almacenamiento de tarjetas que se llamará "Main"; y el segundo esquema para los datos relacionados con los pagos y la plataforma externa de datos que se llamará "Payments".
- **Criterios de aceptación:** Las propias bases de datos con las tablas señaladas en el modelo de datos.

#### Incidencias secundarias

Ordenar por

100 % hecho

TFG-31	DDL - Creación del esquema Main y sus tablas	FINALIZADA
TFG-32	DDL - Creación del esquema Payments y sus tablas	FINALIZADA

Ilustración 20 – Tarea Jira HdU : TFG-15

A continuación, mostraremos las sentencias SQL de creación de los esquemas y respectivas tablas:

#### DDL - Creación del esquema Main y sus tablas

#### DDL Main

```
CREATE SCHEMA main AUTHORIZATION postgres;
CREATE TABLE main.roles (
    role_id varchar(12) NOT NULL,
    nombre varchar(25) NOT NULL,
    CONSTRAINT roles_pkey PRIMARY KEY (role_id)
);
CREATE TABLE main.usuarios (
    usuario_id varchar(12) NOT NULL,
    nombre varchar(20) NOT NULL,
    apellido varchar(30) NOT NULL,
    "password" varchar(60) NOT NULL,
    email varchar(60) NOT NULL,
    enabled bool NOT NULL,
    CONSTRAINT usuarios_email_key UNIQUE (email),
    CONSTRAINT usuarios_pkey PRIMARY KEY (usuario_id)
);
CREATE TABLE main.tarjetas (
    tarjeta_id int8 NOT NULL,
    usuario_id varchar(12) NOT NULL,
    titular varchar(50) NOT NULL,
    pan varchar(16) NOT NULL,
    expiry_month varchar(2) NOT NULL,
    expiry_year varchar(4) NOT NULL,
    CONSTRAINT tarjetas_pkey PRIMARY KEY (tarjeta_id),
    CONSTRAINT tarjetas_usuario_id_fkey FOREIGN KEY (usuario_id) REFERENCES main.usuarios(usuario_id)
);
CREATE TABLE main.usuarios_rols (
    usuario_id varchar(12) NOT NULL,
    role_id varchar(12) NOT NULL,
    CONSTRAINT usuarios_rols_usuario_id_role_id_key UNIQUE (usuario_id, role_id),
    CONSTRAINT usuarios_rols_role_id_fkey FOREIGN KEY (role_id) REFERENCES main.roles(role_id),
    CONSTRAINT usuarios_rols_usuario_id_fkey FOREIGN KEY (usuario_id) REFERENCES main.usuarios(usuario_id)
);
```

## DDL Payments

```
CREATE SCHEMA payments AUTHORIZATION postgres;
CREATE TABLE payments.charities (
    charity_id varchar(36) NOT NULL,
    "name" varchar(60) NOT NULL,
    description varchar(120) NOT NULL,
    enabled bool NOT NULL,
    CONSTRAINT charities_pkey PRIMARY KEY (charity_id)
);
CREATE TABLE payments.donors (
    donor_id varchar(36) NOT NULL,
    usuario_id varchar(12) NOT NULL,
    username varchar(60) NOT NULL,
    usertype varchar(30) NOT NULL,
    email varchar(60) NOT NULL,
    locale varchar(5) NOT NULL,
    "name" varchar(30) NULL,
    acceptedconsents bool NOT NULL,
    CONSTRAINT donors_email_key UNIQUE (email),
    CONSTRAINT donors_pkey PRIMARY KEY (donor_id),
    CONSTRAINT donors_username_key UNIQUE (username),
    CONSTRAINT donors_usuario_id_key UNIQUE (usuario_id)
);
CREATE TABLE payments.cards (
    card_id varchar(36) NOT NULL,
    donor_id varchar(36) NOT NULL,
    primaryaccountsuffix numeric(16) NOT NULL,
    CONSTRAINT cards_pkey PRIMARY KEY (card_id),
    CONSTRAINT cards_donor_id_fkey FOREIGN KEY (donor_id) REFERENCES payments.donors(donor_id)
);
CREATE TABLE payments.guest_donation (
    transaction_id varchar(36) NOT NULL,
    donor_id varchar(36) NULL,
    charity_id varchar(36) NOT NULL,
    card_id varchar(36) NOT NULL,
    amount numeric(4, 2) NOT NULL,
    transactionstatus varchar(8) NOT NULL,
    donationtype varchar(8) NOT NULL,
    email varchar(60) NOT NULL,
    CONSTRAINT guest_donation_pkey PRIMARY KEY (transaction_id),
    CONSTRAINT guest_donation_card_id_fkey FOREIGN KEY (card_id) REFERENCES payments.cards(card_id),
    CONSTRAINT guest_donation_charity_id_fkey FOREIGN KEY (charity_id) REFERENCES payments.charities(charity_id)
);
CREATE TABLE payments.monthly_donation (
    transaction_id varchar(36) NOT NULL,
    donor_id varchar(36) NOT NULL,
    charity_id varchar(36) NOT NULL,
    card_id varchar(36) NOT NULL,
    setup_id varchar(36) NOT NULL,
    setupstatus varchar(7) NOT NULL,
    donationtype varchar(8) NOT NULL,
    amount numeric(4, 2) NOT NULL,
    transactionstatus varchar(8) NOT NULL,
    CONSTRAINT monthly_donation_pkey PRIMARY KEY (transaction_id),
    CONSTRAINT monthly_donation_setup_id_key UNIQUE (setup_id),
    CONSTRAINT monthly_donation_card_id_fkey FOREIGN KEY (card_id) REFERENCES payments.cards(card_id),
    CONSTRAINT monthly_donation_charity_id_fkey FOREIGN KEY (charity_id) REFERENCES
payments.charities(charity_id),
    CONSTRAINT monthly_donation_donor_id_fkey FOREIGN KEY (donor_id) REFERENCES payments.donors(donor_id)
);
CREATE TABLE payments.one_time_donation (
    transaction_id varchar(36) NOT NULL,
    donor_id varchar(36) NOT NULL,
    charity_id varchar(36) NOT NULL,
```

```
card_id varchar(36) NOT NULL,  
amount numeric(4, 2) NOT NULL,  
transactionstatus varchar(8) NOT NULL,  
CONSTRAINT one_time_donation_pkey PRIMARY KEY (transaction_id),  
CONSTRAINT one_time_donation_card_id_fkey FOREIGN KEY (card_id) REFERENCES payments.cards(card_id),  
CONSTRAINT one_time_donation_charity_id_fkey FOREIGN KEY (charity_id) REFERENCES  
payments.charities(charity_id),  
CONSTRAINT one_time_donation_donor_id_fkey FOREIGN KEY (donor_id) REFERENCES payments.donors(donor_id)  
);
```

## HdU : TFG-2 : Generación de claves para firma y encriptación de JWE

Proyectos / TFG / Añadir epic / TFG-2

### Generación de claves para firma y encriptación de JWE

Adjuntar Añadir una incidencia secundaria Vincular incidencia

#### Descripción

HdU : TFG-2 : Generación de claves para firma y encriptación de JWE

- **Descripción:** Se pretende conseguir un certificado simétrico de clave pública/privada para la desencriptación y validación de firma de los datos del JWE y también las claves correspondientes para poder encriptar y firmar estos datos.
- **Criterios de aceptación:** Conseguir un par de claves con un certificado del tipo PKCS12; y dos claves, una pública X509 y otra privada PKCS8 que pueda leer una aplicación Java.

#### Incidencias secundarias

Ordenar por

100 % hecho

TFG-20	Creación de par de claves RSA pública y privada (certificado PKCS12) para el cifrado de datos	FINALIZADA
TFG-30	Creación de clave pública x509 y privada PKCS8 (formato *.der) para la firma de datos	FINALIZADA

Ilustración 21 – Tarea Jira HdU : TFG-2

Con el objetivo de crear una comunicación segura entre cliente y servidor, crearemos una serie de claves que logren proteger nuestros datos mediante la firma de estos y la encriptación de estos.

Respecto a la distribución que haremos de estas claves, proporcionaremos al microservicio Generador de JWE las siguientes claves:

- Una clave privada *PKCS8* para firmar los datos del *JWE*, la llamaremos “*private\_key.der*”
- La clave pública del certificado *PKCS12*, “*pkcs12Certificate.p12*”, para el encriptado de los datos del *JWE*

Por otro lado, el microservicio “Pagos”, tendrá acceso a las siguientes claves:

- La clave pública *X509* que llamaremos “*public\_key.der*” y que habrá sido creada a partir de la privada “*private\_key.der*”. Con ella validará la firma de los datos del *JWE*
- La clave privada del certificado *PKCS12*, “*pkcs12Certificate.p12*”, para el desencriptado de los datos del *JWE*



### Creación de clave pública x509 y privada PKCS8 (formato \*.der) para la firma de datos

Crearemos este par de claves a través de *OpenSSL*<sup>24</sup>. Con esta herramienta podremos crear fácilmente y en dos pasos el par de claves que necesitaremos<sup>25</sup>.

En primer lugar, crearemos la clave privada “*private\_key.der*” con el primer comando, y con el segundo le daremos un formato *PKCS8*:

```
openssl genrsa -out private_key.pem 2048

openssl pkcs8 -topk8 -inform PEM -outform DER -in
private_key.pem -out private_key.der -nocrypt
```

Una vez tengamos creada nuestra clave privada, crearemos a partir de ésta la clave pública “*public\_key.der*”. Para ello ejecutaremos el siguiente comando:

```
openssl rsa -in private_key.pem -pubout -outform DER -out
public_key.der
```

### Creación de par de claves RSA pública y privada (certificado PKCS12) para el cifrado de datos

En este caso, para la creación del certificado *RSA* con formato *PKCS12*, utilizaremos la herramienta Java “*keytool*”<sup>26</sup> incluida en el *JRE* con el siguiente comando<sup>27</sup>:

```
keytool -genkeypair -alias alias -storetype PKCS12 -keyalg RSA
-keystore pkcs12Certificate.p12
```

Durante el proceso de creación, nos solicitará que configuremos un alias y una contraseña para poder acceder a sus claves.

**Alias:** “alias”  
**Contraseña:** “pkcs12pwd”

---

<sup>24</sup> URL: <https://www.openssl.org/> Visitada 12/04/2022

<sup>25</sup> URL: <https://programmerclick.com/article/5237548473/> Visitada: 12/04/2022

<sup>26</sup> URL: <https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html> Visitada: 15/04/2022

<sup>27</sup> URL: <https://www.sslmarket.es/ssl/help-trabajar-con-los-certificados-en-java-keystore> Visitada: 15/04/2022

## HdU : TFG-9 : Implementar microservicio generador de JWE

Proyectos / TFG / Añadir epic / TFG-9

### Implementar microservicio generador de JWE

Adjuntar Añadir una incidencia secundaria Vincular incidencia

#### Descripción

HdU : TFG-9 : Implementar microservicio generador de JWE

- **Descripción:** Implementación de un microservicio capaz de generar un JWE con todos los datos de las tarjetas de un usuario en concreto, además deberá exponer un *endpoint* para recuperar información de los usuarios en función de su identificación.
- **Criterios de aceptación:** Servicio de consulta de datos de un usuario y generación de un JWE con los datos de las tarjetas del usuario. La generación del JWE deberá realizarse mediante la librería de *Nimbus JOSE+JWT* mencionado en la HdU : TFG-3

#### Incidencias secundarias

Ordenar por

100 % hecho

TFG-22	Creación de clases entidad: Usuario, Rol y Tarjeta	FINALIZADA
TFG-24	Implementar repositorio REST para el CRUD	FINALIZADA
TFG-25	Decidir Claims del JWE que generaremos	FINALIZADA
TFG-26	Implementar servicio generación JWE con librería Nimbus JOSE + JWT	FINALIZADA

Ilustración 22 – Tarea Jira HdU : TFG-9

Como microservicio cuya responsabilidades serán tanto la de proporcionar un servicio CRUD para la clase entidad Usuario como la de generar un JWE que formará parte de la información adicional que se añadirá al token de autenticación de cada usuario, debemos tener en cuenta que este supondrá el mayor esfuerzo técnico.

#### Creación de clases entidad: Usuario, Rol y Tarjeta

El primer punto que se ha tomado en cuenta es el de la creación de las clases entidad definidas en la [HdU : TFG-6](#). Con la ayuda de las anotaciones de persistencia `@Entity`, `@Table`, `@Id`, `@ManyToMany` y `@OneToMany`, lograremos implementar el entramado de relaciones que existirá entre las clases entidad y la base de datos definidas.

#### Implementar repositorio REST para el CRUD

Como segundo punto, utilizando el módulo de Spring Data JPA<sup>28</sup>, podríamos crear la clase repositorio fácilmente, con métodos que gracias a las ventajas de *Query methods*<sup>29</sup> podremos generar nuestras consultas con un código muy intuitivo. Pero por otro lado podríamos hacerlo aún más rápidamente utilizando el módulo Spring Data REST<sup>30</sup> añadiendo esta como dependencia a nuestro POM.

<sup>28</sup> URL: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/> Visita:10/04/2022

<sup>29</sup> URL: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods> Visita:10/04/2022

<sup>30</sup> URL: <https://spring.io/projects/spring-data-rest> Visita: 10/04/2022

```

<!-- Módulo Spring Data Rest para implementar más fácilmente un servicio CRUD -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>

```

Ilustración 23 - Dependencia Maven Spring Data REST

Una vez importada esta dependencia, simplemente con la siguiente anotación en nuestra clase repositorio, se generará automáticamente un servicio *CRUD* para la tabla usuarios:

```
@RepositoryRestResource(path="usuarios")
```

Además, crearemos una clase configuración que implemente *RepositoryRestConfigurer*<sup>31</sup> para que nuestro CRUD, aparte de la información de la entidad, también nos muestre el identificador de esa entidad.

De esta forma, cuando llamemos al endpoint del microservicio <http://localhost:8081/usuarios/USER-1> nos devolverá la siguiente información:

```

{
  "id": "USER-1",
  "nombre": "Victor",
  "apellido": "Medina",
  "password": "12345",
  "email": "vmedinap@uoc.edu",
  "enabled": true,
  "tarjetas": [
    {
      "id": 0,
      "titular": "Victor Medina Pérez",
      "pan": "1111222233334444",
      "expiry_month": 1,
      "expiry_year": 2040,
      "_links": {
        "usuario": {
          "href": "http://localhost:8081/usuarios/USER-1"
        }
      }
    },
    {
      "id": 1,
      "titular": "Victor Medina Pérez",
      "pan": "2222333344445555",
      "expiry_month": 2,
      "expiry_year": 2040,
      "_links": {
        "usuario": {
          "href": "http://localhost:8081/usuarios/USER-1"
        }
      }
    },
    {
      "id": 2,
      "titular": "Victor Medina Pérez",
      "pan": "3333444455556666",
      "expiry_month": 3,
      "expiry_year": 2040,
      "_links": {
        "usuario": {
          "href": "http://localhost:8081/usuarios/USER-1"
        }
      }
    }
  ],
  "roles": [
    {
      "id": "ADMIN",
      "nombre": "ROLE ADMIN"
    }
  ],
  "_links": {
    "self": {
      "href": "http://localhost:8081/usuarios/USER-1"
    },
    "usuario": {
      "href": "http://localhost:8081/usuarios/USER-1"
    }
  }
}

```

<sup>31</sup> URL: <https://docs.spring.io/spring-data/rest/docs/current/reference/html/#getting-started.configuration> Visita:

## Decidir Claims del JWE que generaremos

En un tercer punto, se decide qué *Claims*<sup>32</sup> van a formar parte de nuestros *JWE* y que serán usados para llevar la información de las tarjetas de los usuarios, ya que ésta será la información sensible que se pretende proteger. Como queremos toda la información posible en nuestro *JWE*, añadiremos tanto la información del usuario, así como la de las tarjetas que tenga asociadas. De esta forma, podremos además hacer una comprobación extra que consista en la validación del usuario informado tanto en el *Access token JWT* de autorización, así como el informado en el *JWE*.

## Implementar servicio generación JWE con librería Nimbus JOSE + JWT

En cuarto lugar, una vez hemos completado los puntos anteriores, implementaremos el servicio encargado de generar el *JWE* que se proporcionará como información adicional en el token de autorización. Tal y como se ha decidido, haremos uso de la librería de *Nimbus JOSE+JWT* ; concretamente daremos especial atención al ejemplo que nos exponen en su web oficial y es el de encriptado de *JWT* mediante encriptación RSA<sup>33</sup>.

Este será el aspecto de un *JWE* sin desencriptar:

```
eyJraWQiOiJqd2VZDZkXJ0aWZpY2F0ZSIsImN0eSI6IkpXRSIsImVudYiI6IkEyNTZHQ00iLCJhbGciOiJSU0EtT0FFUC0yNTYifQ.QSrqioK5BIHXcAJ5kaFJbPUZJd7F1vbZZoyOj4Eo-qZe2tIRIjc4QhSFXFLGva_QLCURvCRkOYw4cg-XWQbfPBn3E1ssqw_4DdaCFOJtq0aGYMXyfbfNs0spZPrsFydQ4gfVvxltLce54G7JbSWDp8ledDtOF514Rvm-sMODqJGYerkDO1U5BufUnUWjnuUudaP3O-xtFbE2gALnqiW6rh_58-Acud_8JJjun1Gr_FILvNSzlf2clEqo-i_IMHKWW4numnCYbtbCqBTJ2mHEKYMOFmg2INvFH5OS-1OEawG6tiolbtL0qJHj_5kaLs18qoWM2BktnRKJM0eaGO3Jw.mlltfikeSbw_7gK3.TOWYukGS1KY7Ca7u-5cj6JzqQE0Hfgp1YlptsmOr0W0eVANQVxiuQdYJ1CKehORRC76_5eilTgQORJxEcRuryPiqCH6p9DPrbQ9mR14HoA6VWIKoe7c5E-VISCOt3bpWykbPuNdL0USKesqLSgDI0F-L71kYfzf3BwQdqs8JR2fIRm0wSo59z3GeNhwfi7EPwapofrYwxviapatQ_SYWZGLf2DVo781sI2TRV2VpRtD7z_IPGlucJrWpDZMrqxWDqMS-khf-F--D5Qgv5uP0gOrazGTIWwgtgjj-6ZQ7tZJ4HlGNI0gzqPdcez2vvQGgnltFaOK9ZyShvpaYMQbgoT9RRVJclEuHVM30-29hCnzNk3ZI8G3alzfwGXUrAT-tRFTxoX_bCkQpfpouifH_5x1ontHFD3gnjHS48UPVEHN_8drZZ9si_PIKgmgrqt4KHk7diQOcPkT4vlnu1x6tUaXUYCjU9LrXNI_aetm3mEkOQ7ejLN8eSjPsw6E60k_eEX704VQCoBQjM6UgJWarzU2SZqHz1Z4jBR0wzGwgNKy4SVkE6UItjsz03KViXnEaSYFLkX5k0JEZrW-a8K-wIRgbz4zxMNx4pVVo_YVWowg6NnQLjSABBA5ez5XdDM8ektNPaeyxm8uTKBm42OprXjAdtHrvfbAofqIS9Dc2ZA-s0euBFZAv7kelt88biKGezyV7KFbFFH9aTRmyMY1TG3crk5bW5VNEkK_vjYiDo33sdByJeOC1gPBlma2MylISAmjlir88L_QEcnt8SAK-e6TyBGVkhIEbg_ae6fQ3_uPviiZvDBtGDruRurHJwrhU4vDR5UZSsOx6HYo-7HjWgTPdQbl84GMOMLeihPrGrRqwm5rjM4JUR0FDKMHvE5UqRAKJR5u6e9IC0P6f3ABBuFtqoz7jfkVZKr4ihWCBz7Yv8lf8XRpK5J1aHr289T6HXnsIFdjo97uQENbgyiS-fyC9Tika2kzGo0NyuK5OWzumQrma9R8ba14F1ac4pdg6SRFFIR_GgcNQH8_62d2_lvdetK4X_5TLjC2TRA3hqpS1VGbsuCM7oVQikYu5Z-atWdr8kQUscP2ujLZluRHTak_QXdad0w1ugmKOQdx2tC2smygFW5_SLhQUK1zDN5v0eumR-H44e_nSnr9KNZATvY_XcMWDLrz9nU_gAj1LUf1S6VOpdx9afRjvx99p1jzFQ7-gR0cnHyM0uoxJh4SgSDChiqsDr2rZ_0x8jBG1gmVw2SrQKQqFjzqSTZH0HkI5tM1TYU8kJKLRMgMVHm4_0736pUCKUnxtnSEH3qazfZpDnCUK4hbJ-42h_S1nGVx4vk8z4fD5SQvrKpSc-X7Ibkk0xXqmVS-VTij_D0CLU-jbTD5BlNB0elbdmx2HnRcaNrpJu2rIHWT2YvoydWL-h1MviwgkkJE1-tVaZLBMkSxYl_M_sSdSZETdV3_aSJvtzt4a0mT8PbnOz_hNTD4qPwHjfsKW1fHH3732EbQRY1sCthpd-mA6k5Shjpv2Ah6aDpLwAK58Z5L4i9nzcVMXLl65p7hovtrXn1dFpaZdnecXWNE4aP3ppg.b4ludNxo091smd0xpizIQ
```

<sup>32</sup> URL: <https://connect2id.com/products/server/docs/integration/claims-source> Visita: 10/04/2022

<sup>33</sup> URL: <https://connect2id.com/products/nimbus-jose-jwt/examples/jwt-with-rsa-encryption> Visita: 10/04/2022

Y este es el aspecto del contenido de un *JWE* una vez descriptado, es decir, la información sensible del usuario:

```
{
  "usuario": {
    "password": "$2a$10$59gEyVnZsWdfXFPqJQxBAONRGEZ3sJGJc7jDzgaTsmc879X/aDP0S",
    "tarjetas": [
      {
        "expiry_month": 1,
        "id": 1,
        "pan": 1111222233334444,
        "titular": "V\u00edctor Medina P\u00e9rez",
        "expiry_year": 2040
      },
      {
        "expiry_month": 2,
        "id": 2,
        "pan": 2222333344445555,
        "titular": "V\u00edctor Medina P\u00e9rez",
        "expiry_year": 2040
      },
      {
        "expiry_month": 3,
        "id": 2,
        "pan": 3333444455556666,
        "titular": "V\u00edctor Medina P\u00e9rez",
        "expiry_year": 2040
      }
    ],
    "apellido": "Medina",
    "roles": [
      {
        "id": "ADMIN",
        "nombre": "ROLE_ADMIN"
      },
      {
        "id": "USER",
        "nombre": "ROLE_USER"
      }
    ],
    "id": "USER-1",
    "nombre": "V\u00edctor",
    "email": "vmedinap@uoc.edu",
    "enabled": true
  }
}
```

## 4.5 Retrospective

Al trabajar con una versi\u00f3n de java superior a la 8, hemos necesitado incluir la dependencia de *org.glassfish.jaxb* con su artefacto *jaxb-runtime* en todos nuestros microservicios ya que esta dej\u00f3 de incluirse en las respectivas JDK.

# 5. Sprint 3

## 5.1 Sprint Planning

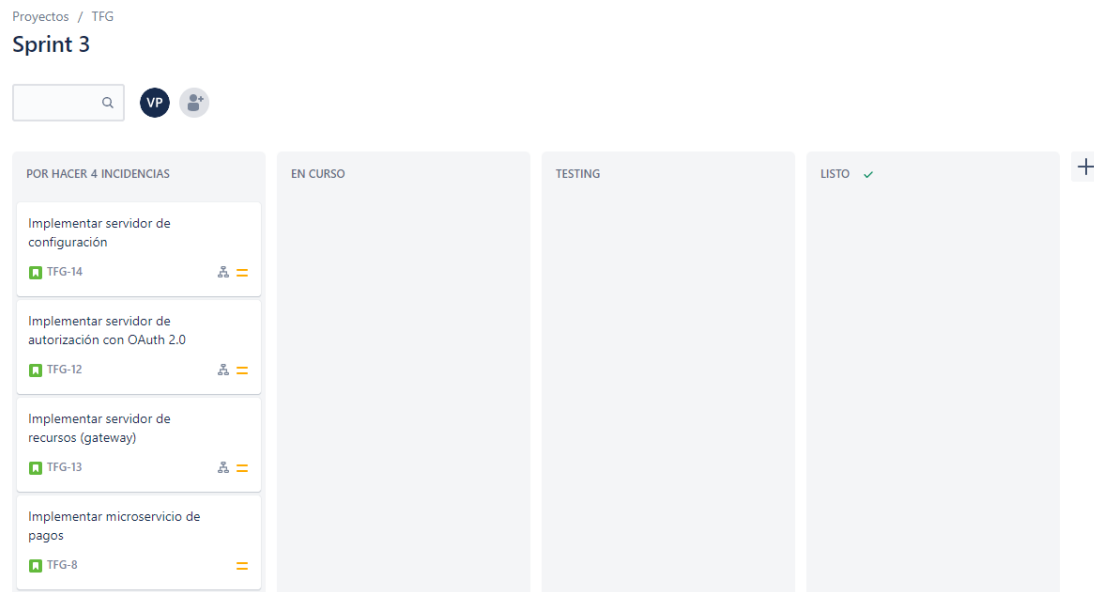


Ilustración 24 - Tablero Sprint 3

Para las siguientes tareas, se recoge la siguiente información:

- HdU : TFG-12 : Implementar servidor de autorización con OAuth 2.0
  - **Descripción:** Este microservicio se encargará de autorizar a todos aquellos usuarios del sistema. Es decir, para aquellos que formen parte de la BD, podrán obtener un token de acceso con el que se identificarán ante el resto de los recursos del servidor.
  - **Criterios de aceptación:** Generará JWT con toda la información necesaria del usuario en cuestión. Es decir, identificación del usuario, así como el listado de tarjetas que tenga asociado (esta será información adicional que se incluirá en futuros sprints como un JWE dentro del JWT)
- HdU : TFG-13 : Implementar servidor de recursos (Gateway)
  - **Descripción:** Tenemos que lograr implementar un microservicio que sirva como puerta de entrada a todos los recursos que formen parte del servidor. Es decir, éste será nuestro *Gateway* para acceder a todos los recursos.
  - **Criterios de aceptación:** El resultado final de esta HdU será la de implementar un microservicio a modo de servidor de recursos que controle y proteja el acceso al resto de componentes del sistema.
- HdU : TFG-14 : Implementar servidor de configuración
  - **Descripción:** Este microservicio se encargará de reunir todas aquellas configuraciones de las que dependan el resto de los microservicios del sistema.

- **Criterios de aceptación:** Este microservicio deberá estar disponible en el repositorio GitHub.
- HdU : TFG-8 : Implementar microservicio de pagos
  - **Descripción:** Implementar microservicio de pagos encargado de recibir peticiones del *frontend* tanto para crear clientes en la plataforma de pago externa, así como realizar transacciones ficticias con esta.
  - **Criterios de aceptación:** Este microservicio debe ser capaz de poder descryptar y validar el *JWE* recibido. En caso de no poder contar con una plataforma externa como MasterCard, sustituirla por un *mockeado* de respuestas de un API tal y como se mencionó en la [Retrospective del Sprint 1](#).

## 5.2 Analysis, design and build

### Etapa de análisis

En este sprint, se ha decidido crear un módulo adicional al proyecto que se llama “db-commons” y su finalidad será la de evitar duplicidad de código en lo que se refiere a las clases entidad “Usuario”, “Tarjeta” y “Role”.

Tanto los microservicios “Generador JWE” como el servidor de autorización “oauth-server” tendrán en su pom la dependencia de este nuevo módulo:

```
<dependency>
  <groupId>ms.vmedinap.tfg</groupId>
  <artifactId>db-commons</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <scope>compile</scope>
</dependency>
```

Ilustración 25 - Dependencia Maven db-commons

### Etapa de diseño

Se ha utilizado el diseño propio de microservicios con tecnología Springboot. Es decir, principalmente utilizando el contexto de aplicación Springboot y el contenedor de beans propio.

Como buenas prácticas se ha seguido un método de programación SOLID durante la implementación de todas las clases de los microservicios, entre otras cosas, por ejemplo, se han aplicado las ventajas de la inyección de dependencias.

### Etapa de construcción

El módulo del proyecto correspondiente al microservicio servidor de configuración “config-server” tiene la siguiente estructura:

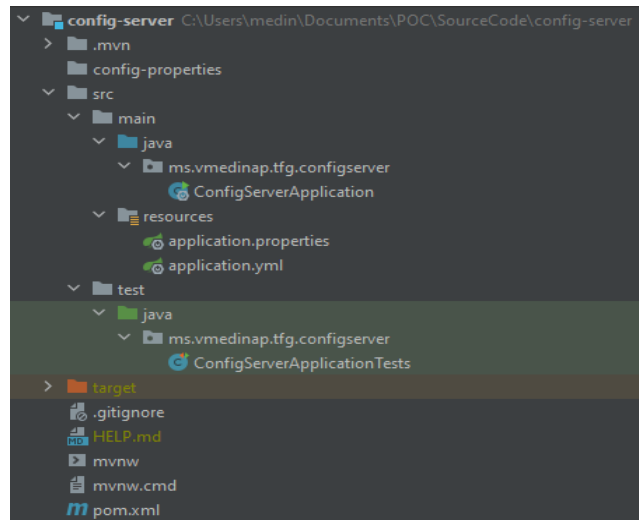


Ilustración 26 - Estructura módulo config-server

El módulo del proyecto correspondiente al microservicio servidor de recursos “gateway-server” tiene la siguiente estructura:

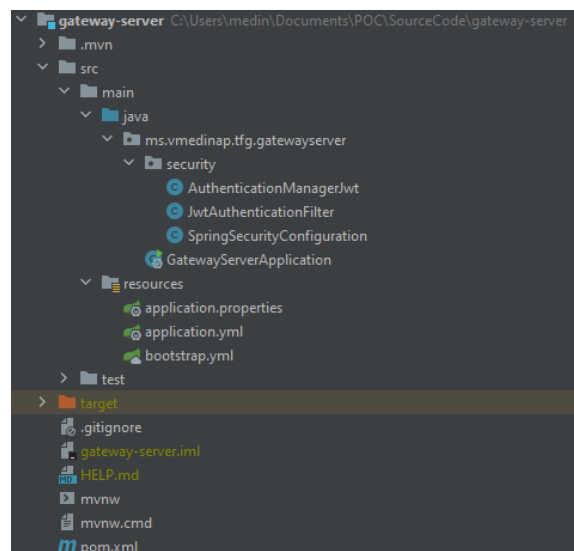


Ilustración 27 - Estructura módulo gateway-server

El módulo del proyecto correspondiente al microservicio servidor de recursos “oauth-server” tiene la siguiente estructura:



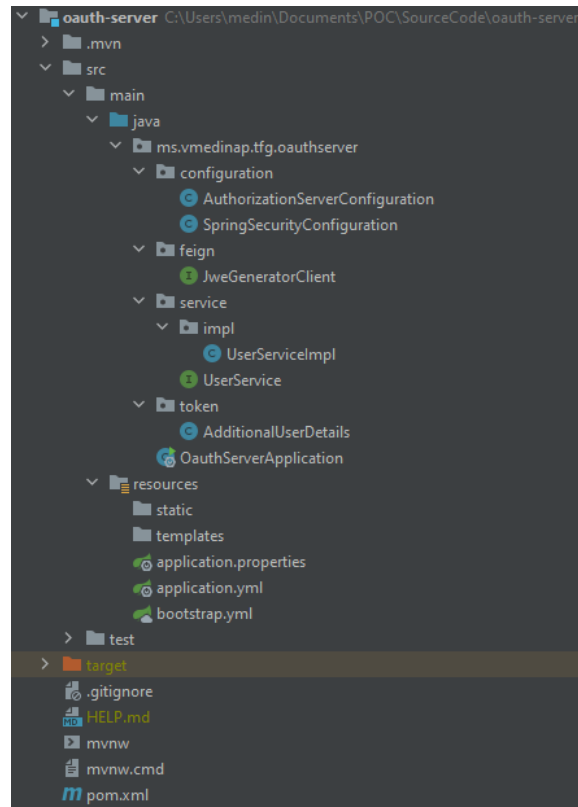


Ilustración 28 - Estructura módulo oauth-server

El módulo del proyecto correspondiente al JAR que contendrá las entidades comunes “db-commons” tiene la siguiente estructura:

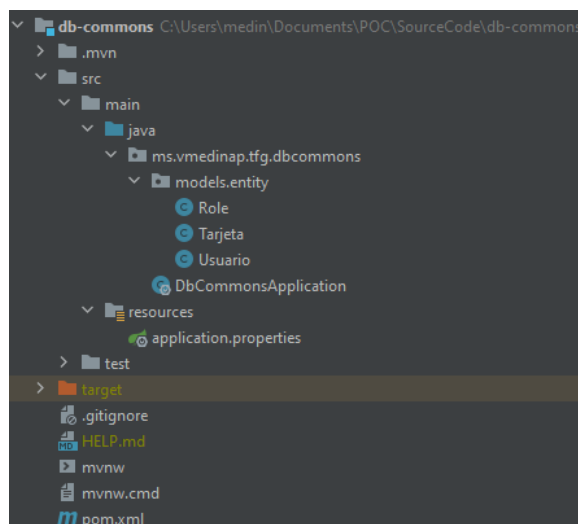


Ilustración 29 - Estructura módulo db-commons

### 5.3 Tests

Se han realizado llamadas a los tres nuevos componentes del sistema a través de un cliente Postman para comprobar el correcto funcionamiento de:

- La configuración del nuevo servidor de recursos Spring Cloud Gateway y acceso al resto de recursos del sistema.

- La configuración del propio servidor de configuración y su comunicación con el nuevo repositorio para almacenar los archivos *properties* de cada microservicio.
- La respuesta del servidor de autorización frente a cada petición de token, así como la validez de estos *access\_token* en el resto de los microservicios del sistema.
- Se ha creado la clase test *AuthenticationManagerJwtTest* en el microservicio del servidor de recursos, para poder testear la implementación que logra validar el *access\_token* usando un algoritmo *Hmac256* y la librería *Nimbus JOSE+JWT*

## 5.4 Sprint Review

HdU: TFG-14 : Implementar servidor de configuración

Proyectos / TFG / Añadir epic / TFG-14

### Implementar servidor de configuración

Adjuntar Añadir una incidencia secundaria Vincular incidencia ...

Descripción

HdU : TFG-14 : Implementar servidor de configuración

- **Descripción:** Este microservicio se encargará de reunir todas aquellas configuraciones de las que dependan el resto de los microservicios del sistema.
- **Criterios de aceptación:** Este microservicio deberá estar disponible en el repositorio GitHub.

Incidencias secundarias Ordenar por ... +

100 % hecho

TFG-40	Creación de un repositorio GitHub auxiliar dedicado a almacenar las propiedades	FINALIZADA
TFG-41	Configurar resto de recursos del sistema como clientes del servidor de configuración	FINALIZADA

Ilustración 30 – Tarea Jira HdU : TFG-14

Vamos a crear un servidor de configuración sencillo que nos va a permitir centralizar la configuración de todos nuestros microservicios, además de por entornos.

*Creación de un repositorio GitHub auxiliar dedicado a almacenar las propiedades*

Para ello vamos a utilizar *Spring Cloud Configuration Server*<sup>34</sup> mediante un repositorio de GitHub. Podríamos [utilizar el que ya creamos al principio](#) de este proyecto o uno nuevo.

En éste caso, como en un futuro se podría querer dar un trato diferente ahí donde localicemos nuestras propiedades más sensibles, vamos a crear un nuevo repositorio.

<sup>34</sup> URL: [https://cloud.spring.io/spring-cloud-config/multi/multi\\_\\_spring\\_cloud\\_config\\_server.html](https://cloud.spring.io/spring-cloud-config/multi/multi__spring_cloud_config_server.html)  
Visitada: 02/04/2022



Ilustración 31 - Repositorio GitHub para propiedades

<https://github.com/vmp1994/tfg-properties>

Una vez creado, introduciremos las propiedades que nos interesen en éste nuevo repositorio para cada uno de los recursos.

Hasta el momento podríamos incluir las propiedades del microservicio “Generador JWE”, como podrían ser los nombres de las claves necesarias para encriptar/desencriptar datos, así como firmarlos y verificar la firma:

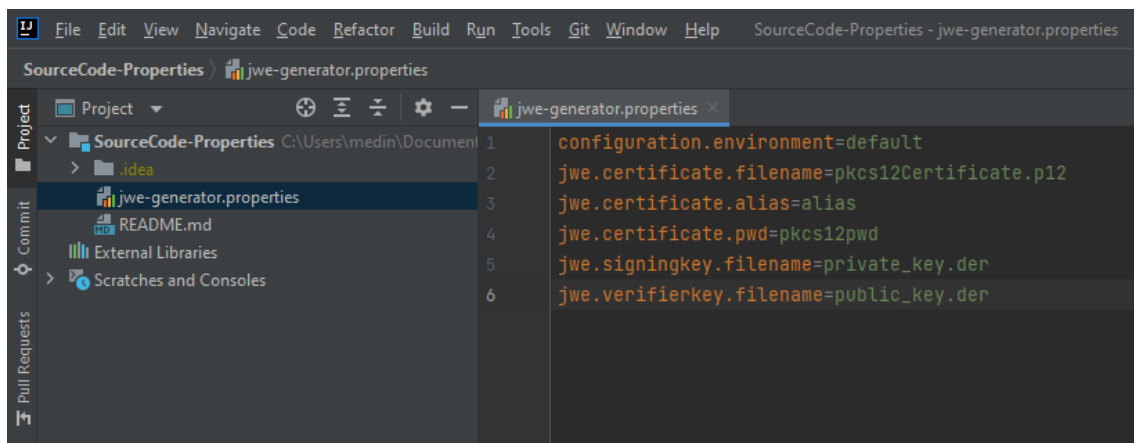


Ilustración 32 - Contenido jwe-generator.properties

*Configurar resto de recursos del sistema como clientes del servidor de configuración*

Posteriormente, modificaremos todos los microservicios que accederán a estas propiedades, como clientes del *Spring Cloud Config Server*, es decir, *Spring Cloud Config Clients*.

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

Ilustración 33 - Dependencia Maven Cloud

Además, esto implicará una pequeña reconfiguración de cada uno de sus `application.yml`<sup>35</sup> así como de la creación del fichero de configuración adicional `bootstrap.yml`<sup>36</sup>.

<sup>35</sup> URL: <https://www.baeldung.com/spring-cloud-configuration> Visitada: 03/04/2022

<sup>36</sup> URL: [https://cloud.spring.io/spring-cloud-config/multi/multi\\_spring\\_cloud\\_config\\_client.html#config-first-bootstrap](https://cloud.spring.io/spring-cloud-config/multi/multi_spring_cloud_config_client.html#config-first-bootstrap) Visita: 03/04/2022

## HdU : TFG-13 : Implementar servidor de recursos (Gateway)

Proyectos / TFG / Añadir epic / TFG-13

### Implementar servidor de recursos (gateway)

Adjuntar Añadir una incidencia secundaria Vincular incidencia

#### Descripción

HdU : TFG-13 : Implementar servidor de recursos (Gateway)

- **Descripción:** Tenemos que lograr implementar un microservicio que sirva como puerta de entrada a todos los recursos que formen parte del servidor. Es decir, éste será nuestro Gateway para acceder a todos los recursos.
- **Criterios de aceptación:** El resultado final de esta HdU será la de implementar un microservicio a modo de servidor de recursos que controle y proteja el acceso al resto de componentes del sistema.

#### Incidencias secundarias

Ordenar por

100 % hecho

TFG-20	Generar JAR inicial con Spring Initializr	FINALIZADA
TFG-21	Configuración de los recursos del servidor en formato YML	FINALIZADA

Ilustración 34 – Tarea Jira HdU : TFG-13

Como servidor de recursos, nuevamente *Springboot* con el módulo de *spring-cloud*, nos ofrece varias opciones para implementar un microservicio que funcione como tal.

Tenemos la opción de implementar un *API Gateway* como *Zuul API Gateway* o *Spring Cloud Gateway*<sup>37</sup>. Para elegir cuál, vamos a realizar un pequeño análisis respecto a cuál de los dos es más recomendable.

Empezando por *Zuul API Gateway*, vemos que tiene entre otras características:

- Enrutamiento dinámico de los microservicios registrados en *Eureka Server*.
- Permite centralizar el acceso a cualquier recurso del sistema gracias a poder configurar un prefijo de todas las rutas.
- Se integra de forma automática con *load balancer*.
- Todas las peticiones que pasan por *Zuul* pasan por defecto por el balanceador de cargas.
- Maneja tolerancia a fallos, latencia y time-out.
- Permite configurar filtros personalizados, para seguridad o autorización.
- Sólo es compatible con las versiones 2.2, 2.3 o 2.4 de *Springboot*.

Por otro lado, *Spring Cloud Gateway*, tiene las siguientes características:

- Enrutamiento dinámico de los microservicios registrados en *Eureka Server*.

<sup>37</sup> URL: <https://spring.io/projects/spring-cloud-gateway> Visitada: 05/04/2022

- Filtros especializados para la autorización, seguridad, monitorización, etc. Pero también nos permite crear filtros personalizados al igual que con *Zuul*.
- Este está implementado por el propio equipo de *Spring Cloud*, por eso es el que se recomienda.
- También nos permite tener una ruta base como acceso común a cualquier otro recurso del sistema.
- Integra también el balanceo de carga tanto con *load balancer*.
- La facilidad de centralizar la seguridad de todas las rutas a nuestros microservicios.
- Compatible con cualquier versión de *Springboot*.

Ambos nos ofrecen prácticamente lo mismo, con pequeñas diferencias.

Ya que la comunidad de *Spring Cloud* recomienda hacer uso de su *API Gateway*, decidimos utilizar *Spring Cloud Gateway*.

Este microservicio se registrará como un recurso más del servidor Eureka.

#### *Configuración de los recursos del servidor en formato YML*

Para configurar cada uno de los recursos de los que vamos a incluir en el servidor, implementamos la siguiente configuración<sup>38</sup> para reflejar todos aquellos microservicios actualmente implementados:

```
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      routes:
        - id: jwe-generator
          uri: lb://jwe-generator
          predicates:
            - Path=/api/usuarios/**
          filters:
            - StripPrefix=2
        - id: oauth-server
          uri: lb://oauth-server
          predicates:
            - Path=/api/oauth/**
          filters:
            - StripPrefix=2
```

Conforme vayan aumentando los recursos del servidor de Eureka, se podrán ir incluyendo en las rutas que configuraremos dentro del *application.yml* del servidor de recursos.

---

<sup>38</sup> URL: <https://cloud.spring.io/spring-cloud-gateway/reference/html/#gateway-starter> Visita: 21/04/2022

## Implementar servidor de autorización con OAuth 2.0

Adjuntar Añadir una incidencia secundaria Vincular incidencia

### Descripción

Añadir una descripción...

### Incidencias secundarias

Ordenar por

100 % hecho

FFG-35	Crear cliente REST para comunicarse con el Generador JWE	-	FINALIZADA
FFG-36	Implementar servicio para recuperar Usuarios y JWE en el token	-	FINALIZADA
FFG-37	Configurar contexto de seguridad de Spring Security	-	FINALIZADA
FFG-38	Establecer nuevo Claim para el JWE e incluirlo como información adicional en el t...	-	FINALIZADA
FFG-39	Incluir configuración OAuth2 en el Gateway	-	FINALIZADA

Ilustración 35 – Tarea Jira HdU : TFG-12

En éste caso podemos trabajar con el módulo de *spring-security*, para traernos la dependencia de *spring-security-oauth2* que nos obligaría a trabajar, entonces con *spring-security-oauth* v2.3.8. A partir de la versión 2.4 en adelante, se elimina el componente que nos genera el token por lo que deberíamos de trabajar con librerías externas (como la mencionada en la [HdU : TFG-3](#))

Como lo ideal es la de trabajar con el mayor número de herramientas oficiales, usaremos la versión v2.3.8 de *spring-security-oauth*.

Por otro lado, también trabajaremos con *spring-security-jwt*.

### Crear cliente REST para comunicarse con el Generador JWE

Para comunicar nuestro servidor de autorización con el microservicio Generador de *JWE*, tendremos que crear un cliente *REST*. Esto, podemos hacerlo de muchas maneras, pero una de las más sencillas es a través de la librería de *Spring Cloud*, *Spring Cloud Feign*<sup>39</sup>.

Al igual que hicimos con *Spring Data REST* de forma declarativa, aquí podemos hacer lo mismo para crear nuestro cliente *REST* a través de las anotaciones *@EnableFeignClients* en la clase principal y *@FeignClient* en la interfaz que creamos como cliente.

<sup>39</sup> URL: <https://www.adictosaltrabajo.com/2017/09/26/spring-cloud-feign-declarative-rest-client/>  
Visita: 25/04/2022

Así, a través de *Eureka Server* con la anotación `@FeignClient`, podrá encontrar el recurso del microservicio Generador *JWE* automáticamente y llamar a los servicios que éste ofrece:

- Buscar un usuario en BD en función del campo email y devolverlo como respuesta.
- Generar *JWE* con los datos de las tarjetas relacionadas con el usuario pasado por parámetro y devolverlo como respuesta.

#### *Implementar servicio para recuperar Usuarios y JWE en el token*

Implementamos un servicio que se encargará de recuperar el detalle del usuario que se va a autenticar y que debe extender de la interfaz *UserDetailsService* de *Spring Security*<sup>40</sup>.

Este servicio usará el cliente *Feign* anteriormente implementado para buscar por el campo "email" a los usuarios persistentes en la base de datos, a través del *endpoint REST Service* del recurso/microservicio Generador *JWE*.

#### *Configurar contexto de seguridad de Spring Security*

Configuramos el contexto de seguridad de *Spring Security*<sup>41</sup> para que el proceso de autorización, que queremos que se lleve a cabo, se realice a través del servicio que hemos implementado para recuperar Usuarios y *JWE* con el cliente *Feign* en el paso anterior. Para ello, crearemos dos clases de configuración.

La primera de estas clases extenderá de *WebSecurityConfigurerAdapter*. Con esta, podremos indicarle al *AuthenticationManager* de *Spring Security* el servicio a través del cual se recuperarán los detalles de los usuarios. Este servicio es el que hemos implementado anteriormente y que extiende de *UserDetailsService*.

Además, *Spring Security* obliga a proteger la información relacionada con las contraseñas por lo que en esta configuración deberemos tener esto en cuenta. Una de las clases que nos proponen usar para proteger esa información es *BCryptPasswordEncoder* del módulo de *Spring Security*.

En la segunda de estas clases configuración, vamos a configurar todo aquello que tendrá que ver con el *JWT*, como el proceso de autenticación, la validación y generación del *JWT*; y finalmente los clientes de este servidor de autorización, es decir, el resto de los recursos del sistema. Esto lo haremos extendiendo de la clase *AuthorizationServerConfigurerAdapter*.

---

<sup>40</sup> URL: <https://www.adictosaltrabajo.com/2020/05/21/introduccion-a-spring-security/#userdetails>  
Visita: 25/04/2022

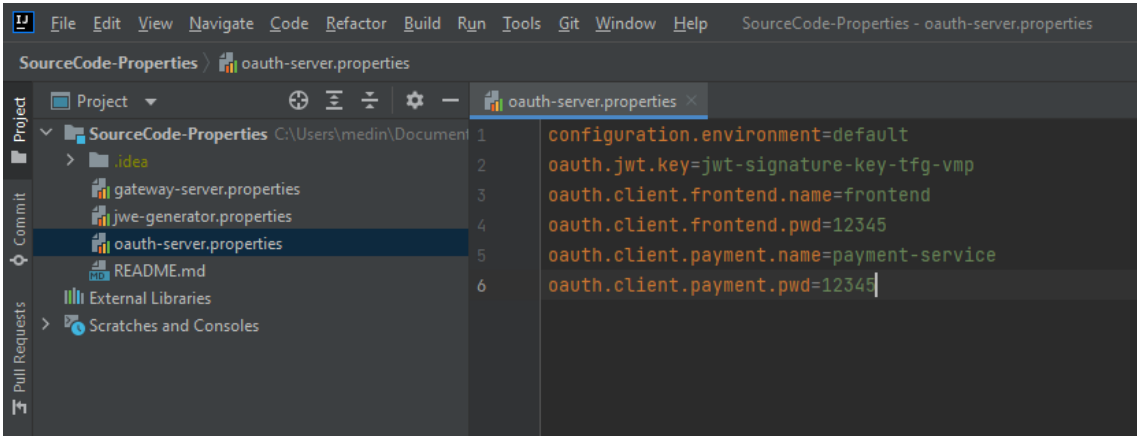
<sup>41</sup> URL: <https://www.adictosaltrabajo.com/2020/05/21/introduccion-a-spring-security/#autenticacion> Visita: 25/04/2022

En esta configuración, además, vamos a establecer la clave con la que se validarán los *JWT*. Esta debe ser una clave secreta. Esta clave secreta ha de cumplir con el estándar *RFC 7518 – HMAC with SHA-2 Functions*<sup>42</sup> la cual cita:

‘Una clave del mismo tamaño que la salida hash (por ejemplo, 256 bits para "HS256") o mayor DEBE usarse con este algoritmo. Este requisito se basa en el Efecto de seguridad del HMAC Key de NIST SP 800-117 [NIST.800-107], que establece que la fuerza de seguridad efectiva es el mínimo de la fuerza de seguridad de la clave y dos veces el tamaño del valor hash interno).’

Teniendo esto en cuenta, la clave secreta se alojará en el fichero “*oauth-server.properties*”, alojado en nuestro servidor de configuración Git implementado en la [HdU : TFG-14](#).

Esta propiedad se corresponde al campo “*oauth.jwt.key=jwt-signature-key-tfg-vmp*” que podéis ver a continuación:



The screenshot shows an IDE window titled 'SourceCode-Properties - oauth-server.properties'. The file explorer on the left shows the project structure with 'oauth-server.properties' selected. The main editor area displays the following content:

```
1 configuration.environment=default
2 oauth.jwt.key=jwt-signature-key-tfg-vmp
3 oauth.client.frontend.name=frontend
4 oauth.client.frontend.pwd=12345
5 oauth.client.payment.name=payment-service
6 oauth.client.payment.pwd=12345
```

Ilustración 36 - Contenido *oauth-server.properties*

Una vez terminemos todos estos pasos, podemos realizar ya una llamada exitosa a nuestro servidor de autorización para conseguir un token de acceso:

```
1 curl --location --request POST 'http://localhost:8090/api/oauth/oauth/token' \
2 --header 'Authorization: Basic cGF5bWVudC1zZXJ2aWN1OjEyMzQ1' \
3 --header 'Content-Type: application/x-www-form-urlencoded' \
4 --data-urlencode 'username=vmedinap@uoc.edu' \
5 --data-urlencode 'password=12345' \
6 --data-urlencode 'grant_type=password'
```

*Establecer nuevo Claim para el JWE e incluirlo como información adicional en el token de acceso*

Estableceremos “*jweData*” como nuevo *Claim* dentro del token de acceso y que contendrá el JWE que contendrá la información de las tarjetas del usuario del token de acceso.

<sup>42</sup> URL: <https://datatracker.ietf.org/doc/html/rfc7518> Visitada: 28/04/2022







```

    "pan": "*****6666",
    "expiry_month": "*",
    "expiry_year": "*"
  }
},
"jweData":
"eyJraWQiOiJqd2VDZXJ0aWZpY2F0ZSIsImN0eSI6IkpXRSIsImVuYyI6IklEYNTZHQ00iLCJhbGciOiJSU0EtT0FFUC0yNTYifQ.VTBRV0oaxZ2EFu5qRfecrjHNxVdi6FYsjLQ_OQK6hk5lujtxu497y6YsT2u9XeZgEqJpPr9PGiabU5O2Te-ZhAz2-Y2NN9OI3RPMtOd8m0EwUzKU9FgFamPT3SHt9Dh_LTMDSvK0r2KBE4qP2BegPVBAff2xil8WkYU4QwyBmK6GkoMOceCP8BVkftIEZM6C18kyOIMYe2Lr-f7bTnGUmamCd6Yd4tgssydC3gczB7i4qUxbJpuStzfJSFmAv07bYhL9-ICnwi2awdc4m9hrGWhwmn5CXYpQBo3KU-Zltj_97bjRAAK-NyxJyHBh0NTP-ekqVxyqI4Umb12rSjgA.lgvlLkBXSSZZR31BY.pB5HsFO-N3C_0K65AnFQYIq54PTP513o8YncpNMhIGDitQzgv-KZrF30TzYGEalKfxPJM68D6DvwxqPKIV4DT4xFvda5LNew5mSSfYHTv7ehgfobAnizATIiFLONYQ9QxA91X97xkM-VUIAWGyZqChOLxwssiSjmtqoMBRWfUdjJaBjrfJR68IRhBwa9WDr_eumcz49uvas8272qb29SMERvsQdSoloqUnV-sNM7B3RLcidXyaBbWT6Xz-DADXQ1X_k4cXOWXHMB1R0tkuQPXs_0wyzb7ww6WIQpNMvyhb8cVGWCh2efXf2pkD5-1-u8t3zW54q9QenOW2KVrpkjZj2sNUNiFOFeOvvlbd8WtsXotDPUPGw-U1GF0Pu87NyZaa0WQMAKT8-bidcol3guXbKEDZcaeceKfw8vxMZ0p6rg2acceavI3SGx1OcfZeLa0t81bZ6RzYfGHSubiHRXqAICBUMq751kKwXrZ-wGRcmwzNlie86knjQILHsnasFYXdrn9cXKqvgOJ-flbKpc8c4FssjOh-v05q6u_GQVukPO1Llw83GVqf8CP43eIVBLQVah1m0NnXGIGGFRATKM6olkynJbaqcBKM6GO5CjvYDEUyVd7vV-F4na9rg-g617JEtPuaiZOptjsuE-Gct7-yZ8JNdKuYkE3vVJBC7umGM1WBCV3vPYk3PTnp0-GM-owAM0OY5fiFYeHr4t_p6TQQHe-zMnVdc_RYxM8n2adGmRc5EoXSv7YvndDru34r138n35zyNW_Aj5dOYJm2L4JpOJvx_9pl0eT-8O-jeci6VyhUvG2m35r_Jm5AhYZKfolNtLhDBX-3dMn8XPTC-jTUxLp0Y2GSzLE15ilz9Kuox4UQYTGgec2fcfL_BX58cCT-ve9QYfMwOPKwGvbNfYnmp1RssT0EDdN3k6h84embCelewZ7pocPB7b5njW_n-hYnqGHlBQ54a-tJpUWHb-c76o3aol7iunEklSyEZA2Ivn_5Rc-FPHN_m3TcleSF34X6MZwrSsD9fbde7JkuPUBOMe84Fv5r4K88miy8O94159wNHGS8Mcx2341UulZ9B5GIXO8PezNYJ7DhmUHsytllsLyNN2brLusBZvpZ9cVHtJLJmya6bs_DYsscAUU61buYEwNuvYoFu3eVjpBIBt-MAAVirLUy8vmqI_T_ErFjlq7OM4Z1d7mv2lpY59G1MnG166xv9ttHVz9sPW-QCSMqJBwgpjRcc7fZ85SZFM4oMcnolGn7EI4oVJJE7GkoZaT50SAsJP-f2kcSotx0t4oaUJOGmhySeFsSmoeUR0neky_skAmKcb7xFO5k1TZxeuVpdZJXdzalzh4ZZ_MzJWmP7hccpqYTRTiR1w77BRIHCTmDziYjqZoeEWWPib8JIZM8GppSPYHsloZT8XvZJQ-y39PG8EJgNmyk_wGT3QOX_8quAsozfH2adeUps8SUGu2nNyGkC3sJICHsniMca7PwXxx3K2T_8JktBJV_a2RNFJSJbFrPxZSW87eX5VLDWEy3n0zp0GzEnqBpg.CygAmax98FDJzaQaf4Z1Qw",
"jti": "f2782b27-b681-47d2-a2ad-15cf690c41ce"
}

```

A través de la página [JWT.IO](https://jwt.io)<sup>44</sup> podremos comprobar si nuestro `access_token` cumple con el estándar *RFC 7519*<sup>45</sup> así como el contenido de este:

```

{
  "tarjetas": [
    {
      "id": 0,
      "titular": "Victor Medina Perez",
      "pan": "*****4444",
      "expiry_month": "*",
      "expiry_year": "*"
    },
    {
      "id": 1,
      "titular": "Victor Medina Perez",
      "pan": "*****5555",
      "expiry_month": "*",
      "expiry_year": "*"
    },
    {
      "id": 2,
      "titular": "Victor Medina Perez",
      "pan": "*****6666",
      "expiry_month": "*",
      "expiry_year": "*"
    }
  ],
  "user_name": "vmedinap@uoc.edu",
  "jweData":
"eyJraWQiOiJqd2VDZXJ0aWZpY2F0ZSIsImN0eSI6IkpXRSIsImVuYyI6IklEYNTZHQ00iLCJhbGciOiJSU0EtT0FFUC0yNTYifQ.VTBRV0oaxZ2EFu5qRfecrjHNxVdi6FYsjLQ_OQK6hk5lujtxu497y6YsT2u9XeZgEqJpPr9PGiabU5O2Te-ZhAz2-Y2NN9OI3RPMtOd8m0EwUzKU9FgFamPT3SHt9Dh_LTMDSvK0r2KBE4qP2BegPVBAff2xil8WkYU4QwyBmK6GkoMOceCP8BVkftIEZM6C18kyOIMYe2Lr-f7bTnGUmamCd6Yd4tgssydC3gczB7i4qUxbJpuStzfJSFmAv07bYhL9-ICnwi2awdc4m9hrGWhwmn5CXYpQBo3KU-Zltj_97bjRAAK-NyxJyHBh0NTP-ekqVxyqI4Umb12rSjgA.lgvlLkBXSSZZR31BY.pB5HsFO-N3C_0K65AnFQYIq54PTP513o8YncpNMhIGDitQzgv-KZrF30TzYGEalKfxPJM68D6DvwxqPKIV4DT4xFvda5LNew5mSSfYHTv7ehgfobAnizATIiFLONYQ9QxA91X97xkM-VUIAWGyZqChOLxwssiSjmtqoMBRWfUdjJaBjrfJR68IRhBwa9WDr_eumcz49uvas8272qb29SMERvsQdSoloqUnV-sNM7B3RLcidXyaBbWT6Xz-DADXQ1X_k4cXOWXHMB1R0tkuQPXs_0wyzb7ww6WIQpNMvyhb8cVGWCh2efXf2pkD5-1-u8t3zW54q9QenOW2KVrpkjZj2sNUNiFOFeOvvlbd8WtsXotDPUPGw-U1GF0Pu87NyZaa0WQMAKT8-bidcol3guXbKEDZcaeceKfw8vxMZ0p6rg2acceavI3SGx1OcfZeLa0t81bZ6RzYfGHSubiHRXqAICBUMq751kKwXrZ-wGRcmwzNlie86knjQILHsnasFYXdrn9cXKqvgOJ-flbKpc8c4FssjOh-v05q6u_GQVukPO1Llw83GVqf8CP43eIVBLQVah1m0NnXGIGGFRATKM6olkynJbaqcBKM6GO5CjvYDEUyVd7vV-F4na9rg-g617JEtPuaiZOptjsuE-Gct7-yZ8JNdKuYkE3vVJBC7umGM1WBCV3vPYk3PTnp0-GM-owAM0OY5fiFYeHr4t_p6TQQHe-zMnVdc_RYxM8n2adGmRc5EoXSv7YvndDru34r138n35zyNW_Aj5dOYJm2L4JpOJvx_9pl0eT-8O-jeci6VyhUvG2m35r_Jm5AhYZKfolNtLhDBX-3dMn8XPTC-jTUxLp0Y2GSzLE15ilz9Kuox4UQYTGgec2fcfL_BX58cCT-ve9QYfMwOPKwGvbNfYnmp1RssT0EDdN3k6h84embCelewZ7pocPB7b5njW_n-hYnqGHlBQ54a-tJpUWHb-c76o3aol7iunEklSyEZA2Ivn_5Rc-FPHN_m3TcleSF34X6MZwrSsD9fbde7JkuPUBOMe84Fv5r4K88miy8O94159wNHGS8Mcx2341UulZ9B5GIXO8PezNYJ7DhmUHsytllsLyNN2brLusBZvpZ9cVHtJLJmya6bs_DYsscAUU61buYEwNuvYoFu3eVjpBIBt-MAAVirLUy8vmqI_T_ErFjlq7OM4Z1d7mv2lpY59G1MnG166xv9ttHVz9sPW-QCSMqJBwgpjRcc7fZ85SZFM4oMcnolGn7EI4oVJJE7GkoZaT50SAsJP-f2kcSotx0t4oaUJOGmhySeFsSmoeUR0neky_skAmKcb7xFO5k1TZxeuVpdZJXdzalzh4ZZ_MzJWmP7hccpqYTRTiR1w77BRIHCTmDziYjqZoeEWWPib8JIZM8GppSPYHsloZT8XvZJQ-y39PG8EJgNmyk_wGT3QOX_8quAsozfH2adeUps8SUGu2nNyGkC3sJICHsniMca7PwXxx3K2T_8JktBJV_a2RNFJSJbFrPxZSW87eX5VLDWEy3n0zp0GzEnqBpg.CygAmax98FDJzaQaf4Z1Qw",

```

<sup>44</sup> URL: <https://jwt.io> Visita: 28/04/2022

<sup>45</sup> URL: <https://datatracker.ietf.org/doc/html/rfc7519> Visita: 28/04/2022

```

yNTYifQ.VTBRVoaxZ2EFu5qRfecrjHNxVdi6FYsJLQ_OQK6hk5lujtxu497y6YsT2u9XeZgEQgJpPr9PGiabU5O2Te-
ZhAz2-
Y2NN9OI3RPMtOd8m0EwUzKU9FgFAMPT3SHt9Dh_LTMDSvK0r2KBE4qP2BegPVBAff2xil8WkYU4QwyBrmK6Gko
MOceCP8BVKtfIEZM6CI8kyOIMYe2Lr-f7bTnGUMamCd6Yd4tgssydC3gczB7i4qUxbJpuStzfPJSFmAv07bYhL9-
lCnwi2awdc4m9hrGWhwmn5CXYpQBo3KU-Zltj_97bj4RAAK-NyxJyHBh0NTP-
ekqVxyql4Umb12rSjgA.lgvlLkBXSSZZR31BY.pB5HsFO-N3C_0K65AnFQYlq54PTPs13o8YncpNMhIGDitQzgzv-
KZrF30TzYGEalKfxPJM68D6DvwxqPKIV4DT4xFvda5LNew5mSSfYHTv7ehgfobAnlZATiiFLONYQ9QxA91X97xkM-
VUIAWGyZqChOLxwssiSjmtqoMBRWFUdjJaBJrJR68IRhBwa9WDr_eumcz49uvas8272qb29SMErvsQdSoloqUnV-
sNM7B3RLcidXyaBbWT6Xz-
DADXQ1X_k4cXOwXHMB1R0tkuQPXs_0wyzb7ww6WIQpNMvyhb8cVWVCh2efXf2pkD5-1-
u8t3zW54q9QenOW2KVRpkjZj2sNUUnIFOFeOvvlbd8WtsXotDPUPGw-U1GF0Pu87NyZaa0WQMAkT8-
bidcol3guXbKEDZcaeceKfW8vxMZ0p6rg2acseavl3SGx1OcfZeLa0t81bZ6RzYfGHSublHRXqAICBUMq751kJkwxRz-
wGRCmwzNlie86knjQILHsnasFYXdrn9cXKqvgJ-fIbKpc8c4FssjOh-
v05q6u_GQUvKPO1Llw83GVqf8CP43eIVBLQVah1m0NnXGIGGFRATKM6oIkynJbaqcBKM6GO5CVjyDEUyVd7vV
F4na9rg-g617JEtPuaiZOptsuE-Gct7-yZ8JNdKuYkE3vVJbc7umGM1WBCV3vPYYk3PTnp0-GM-
owAM0OY5fiFyehR4t_p6TQQHe-
zMnVdc_RYxM8n2adGmRc5EoXsvv7YvnDDru34r138n35zyNW_Aj5dOYJm2L4JpOJvx_9pl0eT-8O-
jeci6VyhUvG2m35r_Jm5AhYZKfoINiLhDBX-3dMn8XPTC-
jTUxLp0Y2GSzLE15ilz9KuoX4UQYTGgec2fcfL_BX58cCT-
ve9QYfMwOPKwGvbNfYnmp1RssT0EDdN3k6h84embCelewZ7pocPB7b5njW_n-hYnqGHlBQ54a-tJpUWHb-
c76o3aol7iunEkISyEZA2lvn_5Rc-
FPHN_m3TcleSF34X6MZwrSsD9fbde7JkuPUBOME84Fv5r4K88miy8O94159wNHGS8Mcx2341UulZ9B5GIXO8Pez
NYJ7DhmUHsyttsjlyNN2brLusBZvpZ9cVHtJLJmya6bs_DYsscAUU61buYEwNuvYoFu3eVjpbIBt-
MAAVirLUy8vmqIT_ErFjlq7OM4Z1d7mv2lpY59G1MnG166xv9ttHVz9sPW-
QCSMqJBwgjprRcc7fZz85SZFM4oMcnolGn7El4oVJjE7GkoZaT50SAsJP-
f2kcSotx0t4oaUJOGmhySeFsSmoeUR0neky_skAmKcb7xFON5k1TZxeuVpdZJXdzalzh4ZZ_MzJWmP7hpcqYTRTi
R1w77BRIHCTmDZiYjqizoeEWWPib8JIZM8GppSPYHsloZT8XvZJQ-
y39PG8EJgNmyk_wGT3QOX_8quAsozfH2adeUps8SUGu2nNyGkC3sJICHsniMca7PwXxx3K2T_8JktBJV_a2RNFJ
SjBfrPxZSW87eX5VLDWEy3n0zp0GzEnqbPg.CygAmax98FDJzaQaf4Z1Qw",
  "scope": [
    "read",
    "write"
  ],
  "exp": 1652003342,
  "authorities": [
    "ROLE_ADMIN"
  ],
  "jti": "f2782b27-b681-47d2-a2ad-15cf690c41ce",
  "client_id": "payment-service"
}

```

Como se puede observar, la información que viaja en el JWT nunca viajará en claro, sino que estará ofuscada. Esto es así para que cualquier interceptor de paquetes que logre ver el contenido de las comunicaciones, entre un cliente y nuestro sistema, no pueda usar estos datos.

Por otro lado, el campo “jweData” sí que contendrá toda la información sensible del usuario ([ver ejemplo de contenido de “jweData”](#)), pero esta estará totalmente protegida gracias a la firma y encriptación de estos datos con claves que sólo conocerán el cliente y nuestro sistema.

### *Incluir configuración OAuth2 en el Gateway*

Para configurar nuestro servidor de recursos de Spring Cloud Gateway para proteger los accesos a estos, vamos a tener que incluir varias dependencias adicionales a nuestra implementación inicial.

Estas dependencias serán *Spring Boot Security*, *Spring Cloud Config* y *Spring Cloud Bootstrap* (estas dos últimas para poder obtener del servidor de configuración la clave de validación del token)

```

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bootstrap</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>

```

Ilustración 37 - Dependencias Maven Cloud/Security/Config

Además de las dependencias que se encargarán de trabajar con los *JWT* que manejará el servidor de recursos Gateway para validarlos. En éste caso, utilizaremos la misma librería que analizamos anteriormente, [Nimbus JOSE+JWT](#), y que también aplicamos en el microservicio “Generador JWE”.

```

<!-- Librería Nimbus JOSE+JWT para trabajar con JWT -->
<dependency>
  <groupId>com.nimbusds</groupId>
  <artifactId>nimbus-jose-jwt</artifactId>
  <version>${nimbus-jose-jwt.version}</version>
</dependency>

```

Ilustración 38 - Dependencia Maven Nimbus-JOSE JWT

Para ello, nos hemos apoyado en su página oficial, concretamente en el ejemplo de *JWT with HMAC*<sup>46</sup>.

Una vez hecho esto, procedemos a implementar las clases de configuración necesarias de *Spring Security* en *Webflux* dentro de nuestro servidor de recursos Gateway, para que podamos proteger nuestras rutas y validar los token de acceso.

Lo primero que tendremos que hacer es implementar una clase configuración que anotaremos con la siguiente etiqueta *@EnableWebFluxSecurity* (recordar que *Spring Cloud Gateway* utiliza tecnología reactiva) Desde esta clase configuración, podremos indicar qué rutas estarán protegidas por un token de acceso y la de nuestro servidor de autorización que estará abierto a todo tipo de peticiones.

En segundo lugar, nos ocuparemos de implementar el filtro de autenticación del JWT de acceso. Este filtro requiere de la implementación de dos clases de configuración más: una que implemente la interfaz

<sup>46</sup> URL: <https://connect2id.com/products/nimbus-jose-jwt/examples/jwt-with-hmac> Visita: 30/04/2022

*ReactiveAuthenticationManager*<sup>47</sup> donde establecemos la validación de la firma del token que establecimos en el servidor de autorización; y otra para implementar el propio filtro con *WebFilter* donde haremos uso de la validación de la firma mencionada en al anterior clase de implementación (ambas utilizan programación reactiva)

## 5.5 Retrospective

El no tener experiencia en programación reactiva ha supuesto un esfuerzo adicional a la que ya va implícita en este Sprint.

Por otro lado, debido a la gran carga de trabajo que ha supuesto la implementación del microservicio Generador de JWE, así como todo el análisis que ha requerido en paralelo la implementación del servidor de autorización OAuth2; no ha dado tiempo a terminar la “HdU : TFG-8 : Implementar microservicio de pagos”. Dada esta situación, se ha acordado abordar esta historia de usuario en el siguiente *sprint*, el Sprint 4:

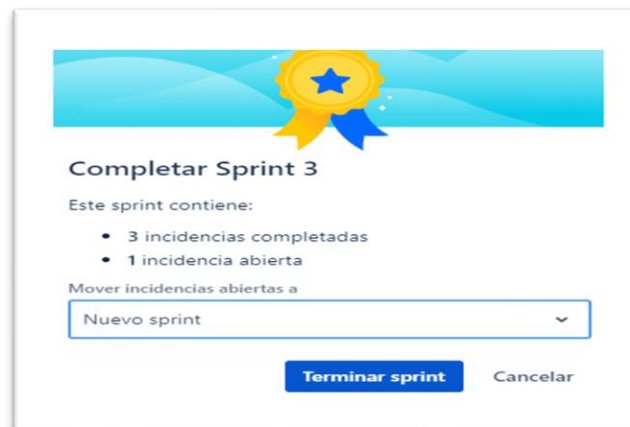


Ilustración 39 - Aviso Jira : informe sprint incompleto

Uno de los aspectos más difíciles de este sprint ha sido el aprender un poco sobre programación reactiva para poder implementar nuestro servidor de autorización.

---

<sup>47</sup> URL: <https://www.baeldung.com/spring-security-authenticationmanagerresolver#authentication-manager-resolver-reactive> Visita:30/04/2022

# 6. Sprint 4

## 6.1 Sprint Planning

Proyectos / TFG

Sprint 4

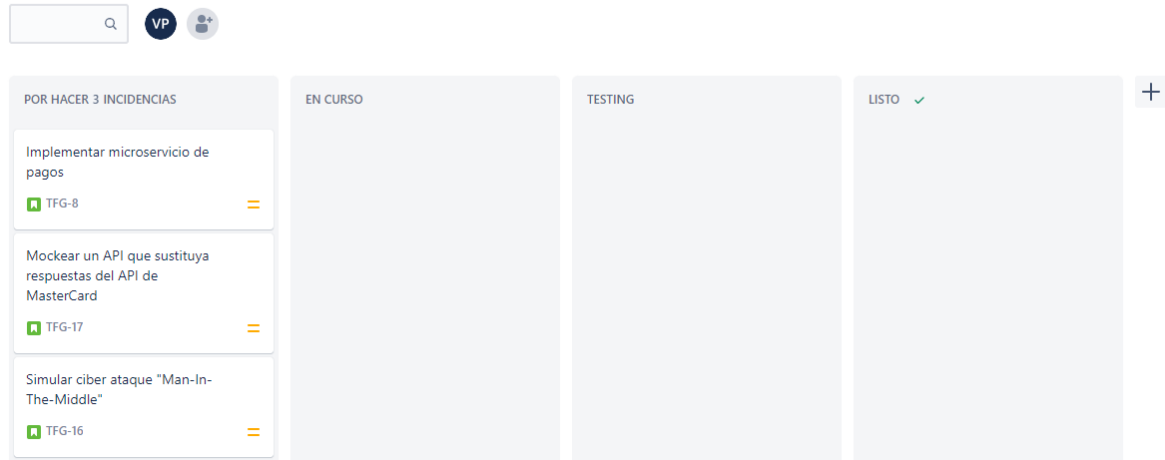


Ilustración 40 - Tablero Sprint 4

Para las siguientes tareas, se recoge la siguiente información:

- HdU : TFG-8 : Implementar microservicio de pagos
  - **Descripción:** Implementar microservicio de pagos encargado de recibir peticiones del *frontend* tanto para crear clientes en la plataforma de pago externa, así como realizar transacciones ficticias con esta.
  - **Criterios de aceptación:** Este microservicio debe ser capaz de poder descifrar y validar el *JWE* recibido. En caso de no poder contar con una plataforma externa como MasterCard, sustituirla por un *mockeado* de respuestas de un API tal y como se mencionó en la [Retrospective del Sprint 1](#).
- HdU : TFG-16 : Simular ciber ataque “Man-In-The-Middle”
  - **Descripción:** Deberemos simular un ataque de ciberseguridad del tipo MITM. Con esto se pretende demostrar que, aun recibiendo ataques de ese tipo, la información sensible estará protegida mediante una estructura *JWE*.
  - **Criterios de aceptación:** Se tiene que interceptar la comunicación entre un hipotético cliente *frontend* y nuestro sistema. Durante esta, debemos intentar leer los datos sensibles que viajan a través de una llamada al servidor de autenticación o cualquier otro servicio del API de Pagos. Será un éxito si vemos que los datos interceptados son inservibles para el interceptor.
- HdU : TFG-17 : Mockear un API que sustituya las respuestas del API de MasterCard
  - **Descripción:** Crear un API que simule ser una plataforma de pago.

- **Criterios de aceptación:** Esta API mockeada debe devolver las respuestas que esperábamos del API *Donate API* de MasterCard.

## 6.2 Analysis, design and build

### Etapa de análisis

Durante el análisis de este sprint, se ha decidido sustituir el API mockeado por una clase `@Service` que nos genere respuestas dinámicamente para cada servicio de creación de Donor, Card y OneTime.

Para ello, se ha hecho un estudio previo a cómo deberían ser estas respuestas teniendo en cuenta la documentación del portal de desarrolladores de MasterCard, concretamente el API de *Donate API*<sup>48</sup> que es el que en un principio se pensó integrar a nuestro sistema.

### Etapa de diseño

Se ha utilizado el diseño propio de microservicios con tecnología Springboot. Es decir, principalmente utilizando el contexto de aplicación Springboot y el contenedor de beans propio.

Como buenas prácticas se ha seguido un método de programación SOLID durante la implementación de todas las clases de los microservicios, entre otras cosas, por ejemplo, se han aplicado las ventajas de la inyección de dependencias.

### Etapa de construcción

El módulo del proyecto correspondiente al microservicio de pagos “payment-service” tiene la siguiente estructura:

---

<sup>48</sup> URL: <https://developer.mastercard.com/donations/documentation/api-reference/> Visita 11/05/2022





Ilustración 41 - Estructura del módulo payment-service

### 6.3 Tests

Se han realizado llamadas al API de pagos a través de un cliente Postman para comprobar el correcto funcionamiento de:

- La configuración del nuevo microservicio de pagos en el servidor de recursos Gateway
- Correcta comunicación y recogida de valores subidos al servidor de configuración en el archivo *properties*.

- Funcionalidad de los servicios de creación de nuevos Donor, Cards y transacciones OneTime.

## 6.4 Sprint Review

HdU : TFG-17 : Mockear un API que sustituya respuestas del API de MasterCard

Proyectos / TFG / Añadir epic / TFG-17

### Mockear un API que sustituya respuestas del API de MasterCard

Adjuntar Añadir una incidencia secundaria Vincular incidencia ...

#### Descripción

HdU : TFG-17 : Mockear un API que sustituya las respuestas del API de MasterCard

- **Descripción:** Crear un API que simule ser una plataforma de pago.
- **Criterios de aceptación:** Esta API mockeada debe devolver las respuestas que esperábamos del API *Donate API* de MasterCard.

*Ilustración 42 – Tarea Jira HdU : TFG-17*

Después de analizar esta tarea a lo largo del desarrollo del Sprint 4, se decidió abordar esta HdU de otra manera. Se ha sustituido la creación de un API por la implementación de un servicio que devuelva respuestas dinámicas dentro del propio microservicio de pagos.

Ver por tanto la tarea abordada dentro de la [HdU : TFG-8 -> Implementar API mockeada como una clase servicio @Service](#)

HdU : TFG-8 : Implementar microservicio de pagos

Proyectos / TFG / Añadir epic / TFG-8

### Implementar microservicio de pagos

Adjuntar Añadir una incidencia secundaria Vincular incidencia ...

#### Descripción

HdU : TFG-8 : Implementar microservicio de pagos

- **Descripción:** Implementar microservicio de pagos encargado de recibir peticiones del *frontend* tanto para crear clientes en la plataforma de pago externa, así como realizar transacciones ficticias con esta.
- **Criterios de aceptación:** Este microservicio debe ser capaz de poder descryptar y validar el *JWE* recibido. En caso de no poder contar con una plataforma externa como MasterCard, sustituirla por un *mockeado* de respuestas de un API tal y como se mencionó en la *Retrospective del Sprint 1*

#### Incidencias secundarias

Ordenar por ... +

100 % hecho

TFG-42	Crear clases entidad: Donor, Card, Charity, Transaction y One-Time	FINALIZADA
TFG-43	Implementar repositorios para las entidades del esquema Payments	FINALIZADA
TFG-44	Implementar servicio de descryptación del JWE y configuración correspondiente	FINALIZADA
TFG-45	Implementar API mockeada como una clase servicio @Service	FINALIZADA
TFG-46	Implementar servicio para trabajar con Donors	FINALIZADA
TFG-47	Implementar servicio para trabajar con Cards	FINALIZADA
TFG-48	Implementar servicio para realizar transacciones del tipo One-Time	FINALIZADA
TFG-49	Documentar API de pagos	FINALIZADA

*Ilustración 43 – Tarea Jira HdU : TFG-8*

Aun no siendo una de las historias de usuario más prioritarias, dado que su ausencia no influiría en la prueba de concepto que pretendemos plasmar en este trabajo, se ha acometido.

#### *Crear clases entidad: Donor, Card, Charity, Transaction y One-Time*

Para ello, se han creado las clases entidad que necesitamos para ceñirnos al modelo de datos previamente definido en el apartado [4.4 HdU : TFG-6](#) del Sprint 2.

Una vez más, hemos recurrido al módulo Spring Data JPA para utilizar sus anotaciones como las de definición de tabla, entidad, relaciones entre entidades, etc.

Estas no se han añadido las clases entidad al módulo *db-commons* (tal y como se hizo en con las clases Usuario, Tarjeta y Role anteriormente) ya que estas clases sólo tienen utilidad para este microservicio.

Para la entidad “One-Time” y “Transaction” hemos recurrido a la etiqueta `@MappedSuperclass` de *Hibernate*<sup>49</sup> para poder representar la herencia de atributos y relaciones de la base de datos, en el esquema Payments, para todas aquellas entidades que hereden de “Transaction”.

#### *Implementar repositorios para las entidades del esquema Payments*

En éste caso, como queremos tener clases que nos faciliten la persistencia en base de datos (y no un servicio *CRUD* como un recurso *REST*), nos basta con utilizar el módulo *Spring Data JPA* y no el *Sprint Data REST*.

Tal y como se mencionó durante la implementación del microservicio “Generador JWE”, utilizaremos las ventajas de *Query Methods* y la interfaz *CrudRepository* para implementar nuestros repositorios (uno para cada clase entidad).

#### *Implementar servicio de descriptación del JWE y configuración correspondiente*

Implementaremos un servicio el cual sólo pueda descriptar y validar la firma de la información contenida en el JWE que recibirá por cabecera (a diferencia del microservicio “Generador JWE” al que dotamos de la capacidad tanto para encriptar como descriptar información, firmar y validar firma del JWE)

Para ello, podremos reutilizar código implementado en el microservicio “Generador JWE”.

Por otro lado, tendremos que subir al servidor de configuración, los parámetros que necesitaremos para poder manipular las llaves de descriptación, así como de validación de firma. Es decir, el nombre del certificado para la descriptación, así como el nombre de la clave pública:

---

<sup>49</sup> URL: <https://www.baeldung.com/hibernate-inheritance> Visita: 07/04/2022

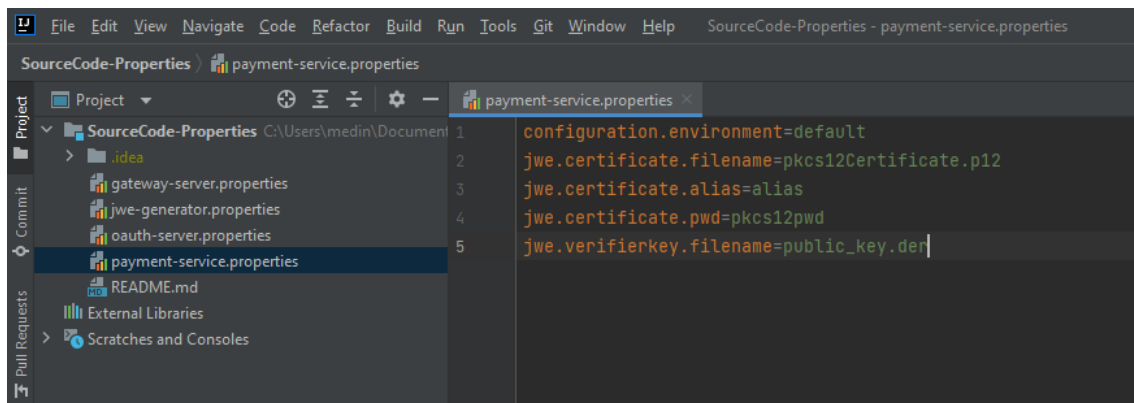


Ilustración 44 - Contenido payment-service properties

### Implementar API mockeada como una clase servicio @Service

La API mockeada la hemos sustituido finalmente por una clase @Service que nos devolverá respuestas dinámicas a los endpoint ficticios de creación de Donor, Card y transacciones del tipo One-Time.

### Implementar servicio para trabajar con Donors

Con el servicio que simulará la API de un externo, junto a la información recibida por la propia de la petición, implementaremos un nuevo servicio capaz de persistir una entidad “Donor” en la BD.

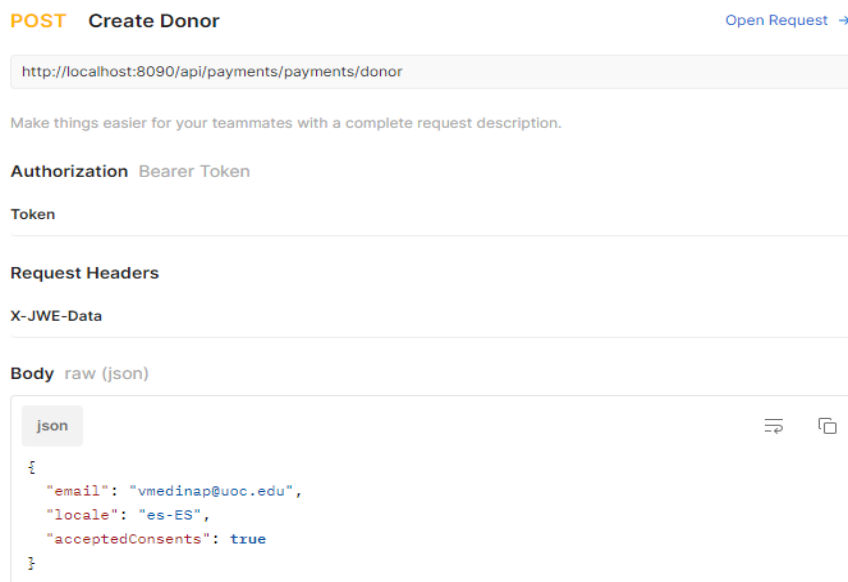


Ilustración 45 - Detalle servicio “Create Donor”

### Implementar servicio para trabajar con Cards

Con el servicio que simulará la API de un externo, junto a la información recibida por la propia de la petición, implementaremos un nuevo servicio capaz de persistir una entidad “Card” en la BD.

Para evitar el intercambio de datos sensibles sin proteger, en el cuerpo de llamada para la creación de donor se envía los cuatro últimos dígitos de la tarjeta que se quiere dar de alta en la plataforma; y estos últimos cuatro dígitos se compararán con las tarjetas que recibimos encriptadas en la cabecera X-JWE-Data.

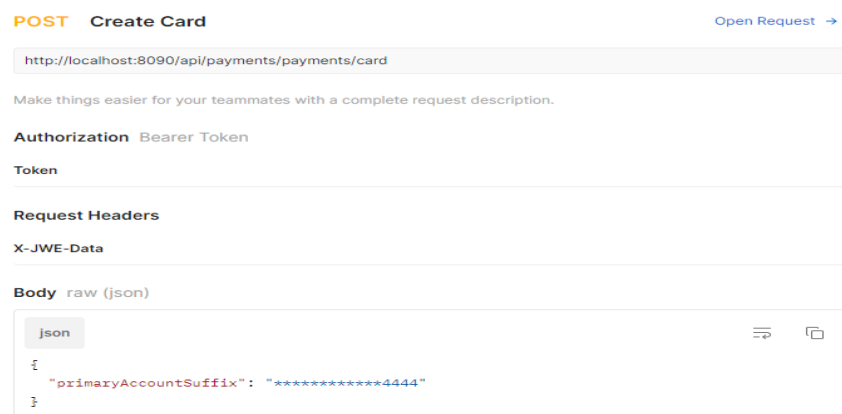


Ilustración 46 - Detalle servicio "Create Card"

### Implementar servicio para realizar transacciones del tipo One-Time

Con el servicio que simulará la API de un externo, junto a la información recibida por la propia de la petición, implementaremos un nuevo servicio capaz de persistir una entidad "OneTime" en la BD.

Previamente, y tal y como nos indica el portal del desarrollador de MasterCard, con la API "Donate API", deberemos insertar una *charity* ficticia en la base de datos para poder realizar la transacciones.

Así pues, para realizar una transacción sin exponer datos sensibles desprotegidos durante la llamada, se validará y seleccionará el card que coincida con los cuatro últimos dígitos del PAN que viaja en el cuerpo de la llamada con las tarjetas que viajan encriptadas en la cabecera X-JWE-Data.

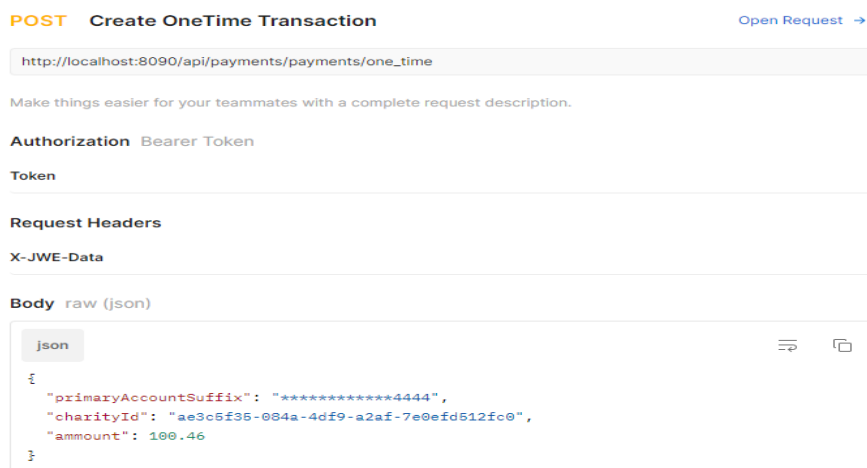


Ilustración 47 - Detalle servicio "Create OneTime Transaction"

## Documentar API de pagos

Finalmente, documentaremos tanto el servicio de pagos con la librería de documentación de APIs, *Swagger OpenAPI*<sup>50</sup>.

Para ello, hemos añadido una dependencia más a nuestro POM:

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.4</version>
</dependency>
```

Ilustración 48 - Dependencia Maven - OpenApi

De esta forma, podremos ver los servicios del API publicados y documentados de la siguiente manera (accediendo a la interfaz de usuario de Swagger para este microservicio):

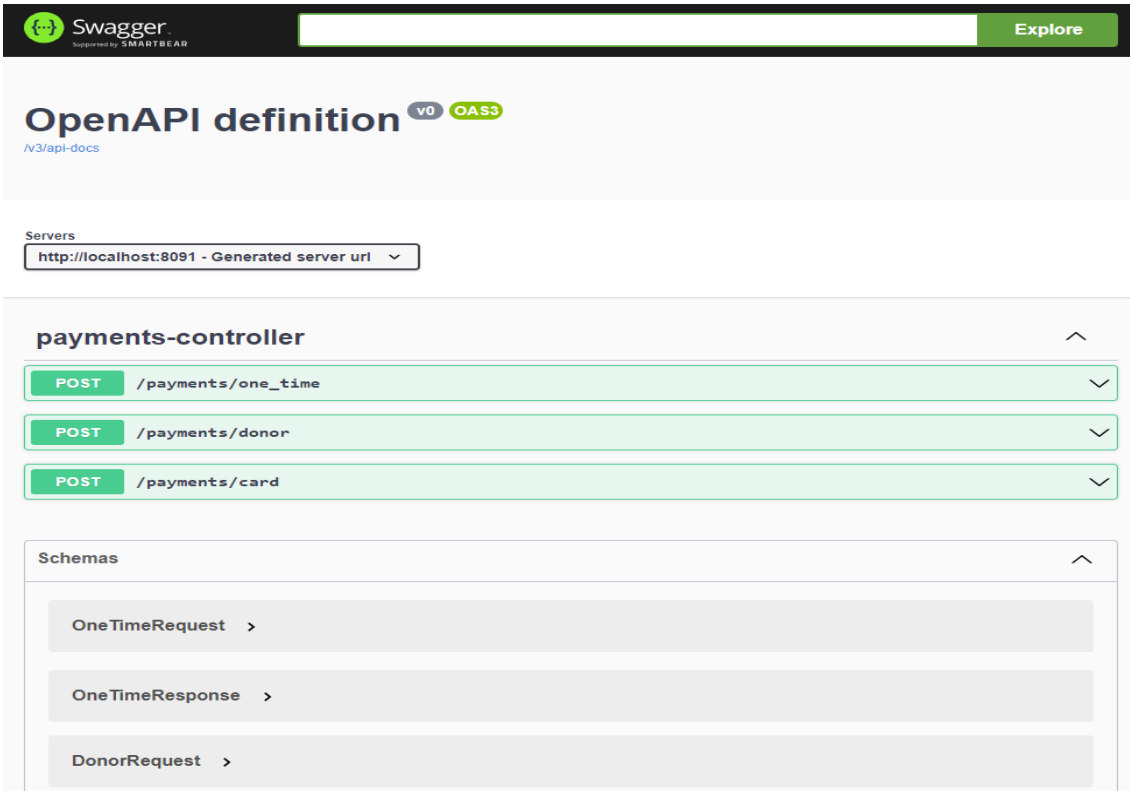


Ilustración 49 - OpenApi definition de microservicio Pagos

<sup>50</sup> URL: <https://www.baeldung.com/spring-rest-openapi-documentation> Visita: 07/05/2022

## HdU : TFG-16 : Simular ciber ataque "Man-In-The-Middle"

Proyectos / TFG / Añadir epic / TFG-16

### Simular ciber ataque "Man-In-The-Middle"

Adjuntar Añadir una incidencia secundaria Vincular incidencia ...

#### Descripción

HdU : TFG-16 : Simular ciber ataque "Man-In-The-Middle"

- **Descripción:** Deberemos simular un ataque de ciberseguridad del tipo MITM. Con esto se pretende demostrar que, aún recibiendo ataques de ese tipo, la información sensible estará protegida mediante una estructura *JWE*.
- **Criterios de aceptación:** Se tiene que interceptar la comunicación entre un hipotético cliente frontend y nuestro sistema; durante esta comunicación debemos intentar leer los datos sensibles que viajan a través de una llamada al servidor de autenticación o cualquier servicio del API de Pagos. Será un éxito si vemos que los datos interceptados son inservibles para el interceptor.

#### Incidencias secundarias

Ordenar por ... +

100 % hecho

TFG-50	Intercepción de llamada y respuesta del servicio de autorización	FINALIZADA
TFG-51	Intercepción de llamada y respuesta del servicio de creación de Donor	FINALIZADA
TFG-52	Intercepción de llamada y respuesta del servicio de creación de Card	FINALIZADA
TFG-53	Intercepción de llamada y respuesta del servicio de transacción One-Time	FINALIZADA

Ilustración 50 – Tarea Jira HdU : TFG-16

En la realización de esta última tarea se han instalado dos programas:

- Postman para simular las peticiones lanzadas desde un hipotético cliente frontend.
- Wireshark para interceptar el tráfico de paquetes en nuestro equipo.

Para el programa de Postman, hemos creado una colección de llamadas con los servicios que cualquier cliente externo utilizaría una vez se integrase con nuestro API de Pagos.

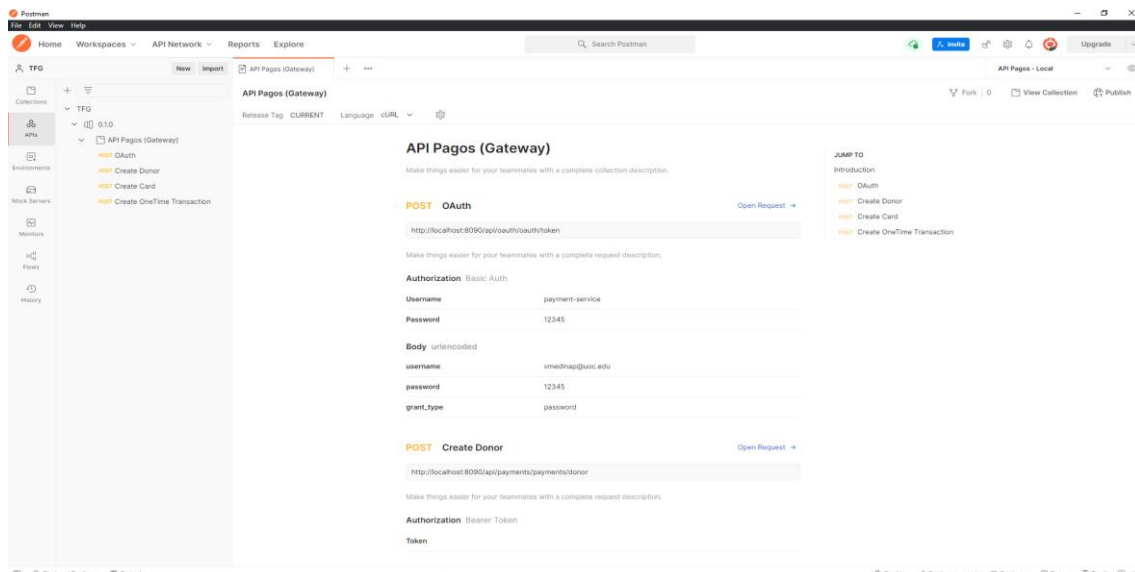


Ilustración 51 - Presentación Postman del API Pagos (A través de Gateway)

Por otro lado, con *Wireshark* hemos configurado un filtro de llamadas capaz de interceptar todas las llamadas HTTP que se hayan realizado en nuestra propia red local.

El filtro empleado para capturar este tráfico con *Wireshark* es el siguiente:  
**(ipv6.src == ::1) && http**

### Intercepción de llamada y respuesta del servicio de autorización

#### Request:

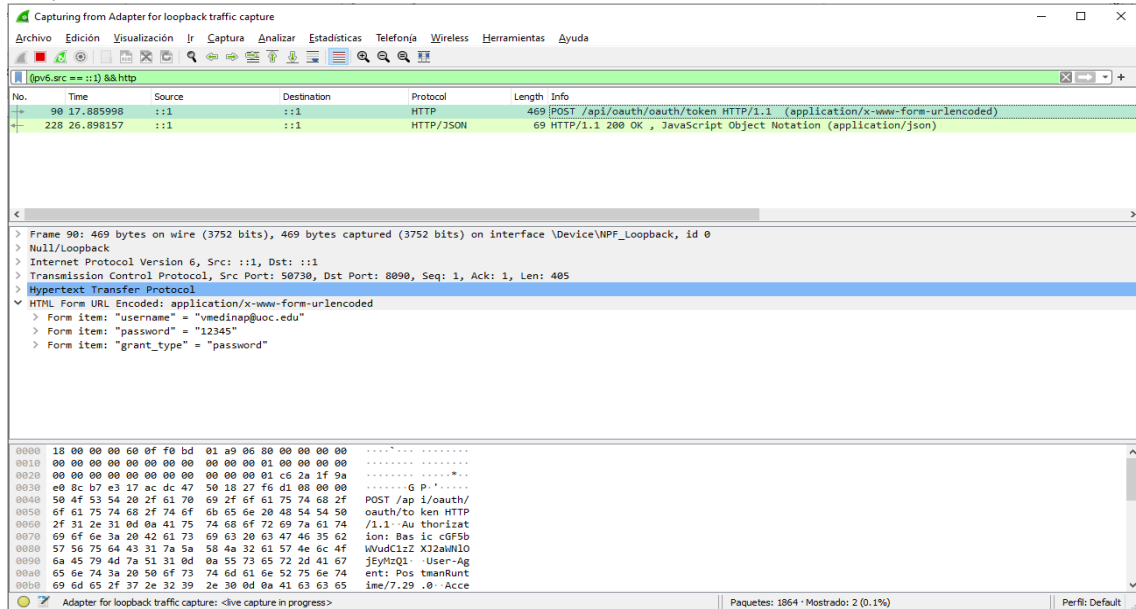


Ilustración 52 - Wireshark intercepción oauth token request

#### Response:

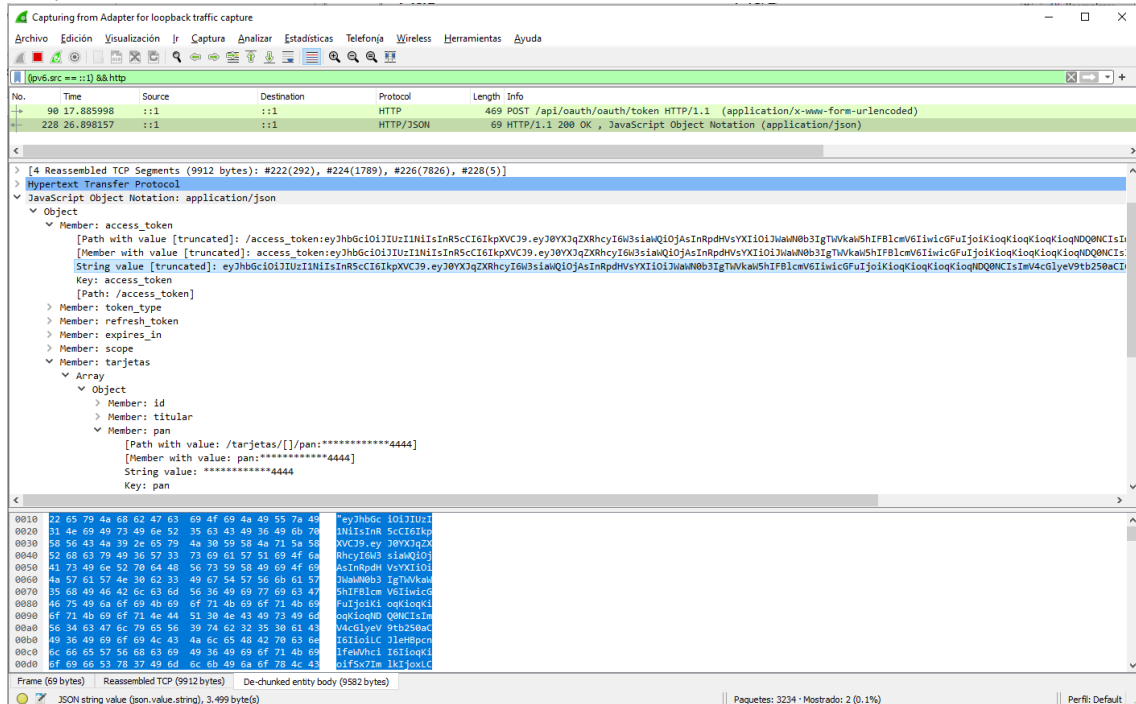


Ilustración 53 - Wireshark intercepción oauth token response



## Intercepción de llamada y respuesta del servicio de creación de Donor

### Request:

The image shows a Wireshark capture of an HTTP POST request. The packet list pane shows a POST request to /api/payments/payments/donor. The packet details pane shows the request body as a JavaScript Object Notation (JSON) object with the following structure:

```

{
  "email": "vmedinap@uoc.edu"
}

```

The packet bytes pane shows the raw data of the request, including the HTTP headers and the JSON body.

Ilustración 54 - Wireshark intercepción create Donor request

### Response:

The image shows a Wireshark capture of an HTTP POST response. The packet list pane shows a POST response to /api/payments/payments/donor. The packet details pane shows the response body as a JavaScript Object Notation (JSON) object with the following structure:

```

{
  "donorId": "832ce9ff-fa5f-4e9f-8151-c73fa2008983",
  "userName": "vmedinap@uoc.edu"
}

```

The packet bytes pane shows the raw data of the response, including the HTTP headers and the JSON body.

Ilustración 55 Wireshark intercepción create Donor response

## Intercepción de llamada y respuesta del servicio de creación de Card

### Request:

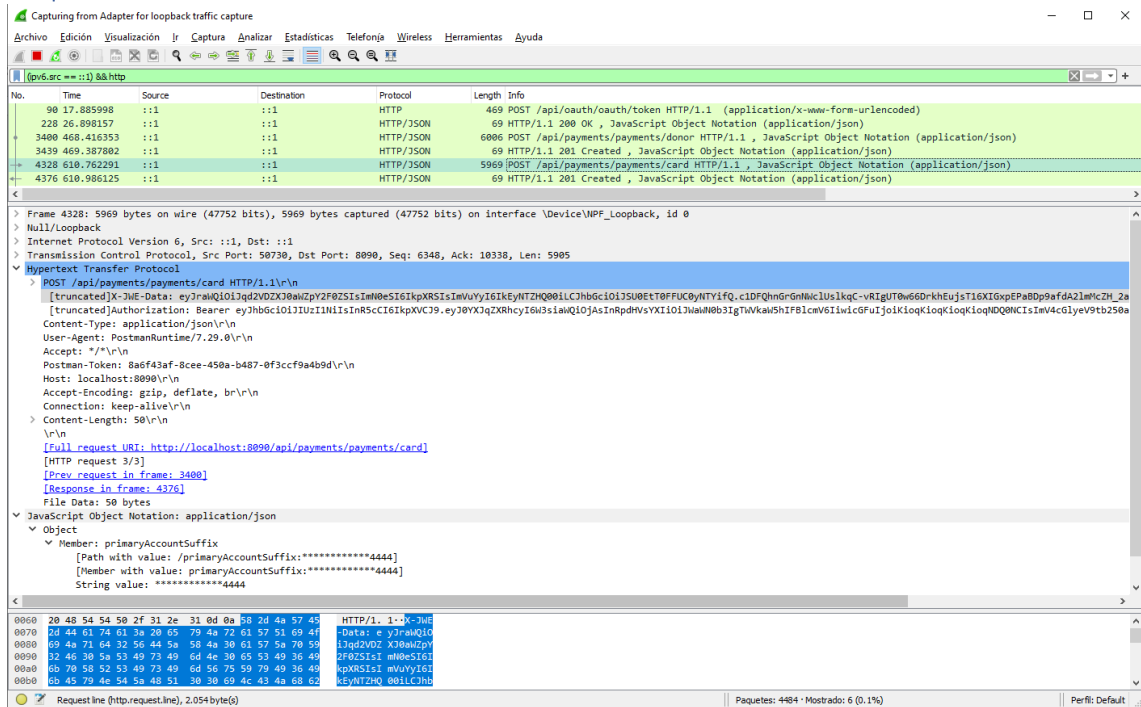


Ilustración 56 - Wireshark intercepción create Card request

### Response:

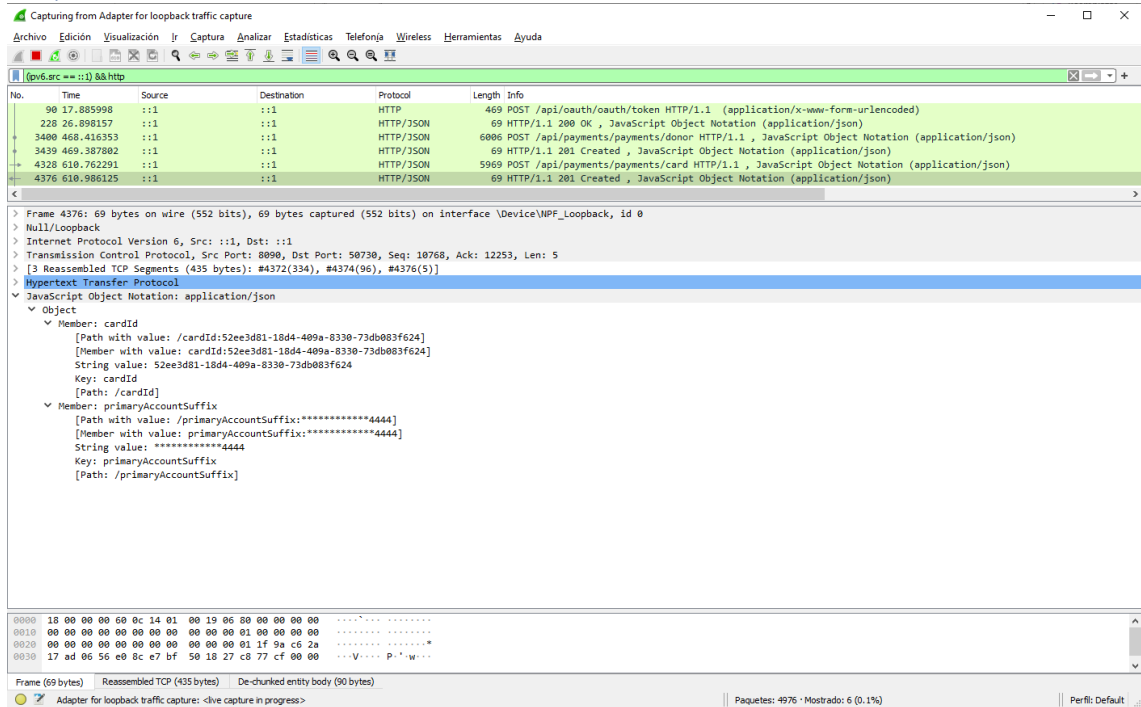


Ilustración 57 - Wireshark intercepción create Card response

## Intercepción de llamada y respuesta del servicio de transacción One-Time

### Request:

Request line (http.request.line), 2,054 byte(s)

Paquetes: 5520 - Mostrado: 8 (0.1%)

Ilustración 58 - Wireshark intercepción create One-Time request

### Response:

Frame 5304: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface \Device\NPF\_{...} id 0

Null/Loopback

Internet Protocol Version 6, Src: ::1, Dst: ::1

Transmission Control Protocol, Src Port: 8090, Dst Port: 50730, Seq: 11236, Ack: 18242, Len: 5

[3 Reassembled TCP Segments (468 bytes): #5300(329), #5302(134), #5304(5)]

JavaScript Object Notation: application/json

Object

- Member: transactionId
  - [Path with value: /transactionId:693c4ef1-d449-4d79-a456-fb6ba3f528]
  - [Member with value: transactionId:693c4ef1-d449-4d79-a456-fb6ba3f528]
  - String value: 693c4ef1-d449-4d79-a456-fb6ba3f528
  - Key: transactionId
  - [Path: /transactionId]
- Member: transactionStatus
  - [Path with value: /transactionStatus:SUCCESS]
  - [Member with value: transactionStatus:SUCCESS]
  - String value: SUCCESS
  - Key: transactionStatus
  - [Path: /transactionStatus]
- Member: transactionMessage
  - [Path with value: /transactionMessage:Transaction APPROVED]
  - [Member with value: transactionMessage:Transaction APPROVED]
  - String value: Transaction APPROVED
  - Key: transactionMessage
  - [Path: /transactionMessage]

Frame (69 bytes) | Reassembled TCP (468 bytes) | De-chunked entity body (128 bytes)

Adapter for loopback traffic capture: <live capture in progress>

Paquetes: 5790 - Mostrado: 8 (0.1%)

Ilustración 59 - Wireshark intercepción create One-Time response

Tal y como se puede apreciar, ninguno de los datos interceptados podría ser utilizado fuera del sistema ya que carecerían de valor en cualquier contexto que no fuera este.

Por ejemplo, el PAN de la tarjeta no se muestra completamente ni durante la creación de una tarjeta en el sistema como durante una transacción, por lo que es un dato sensible enmascarado y por lo tanto protegido. Toda esta información sensible viajará encapsulada en un token encriptado, es decir en el *JWE* de la cabecera *X-JWE-Data*, y que tan sólo podrá ser descifrado y validado por el propio sistema una vez lo reciba.

Con esto podemos asegurar que, independientemente de que esta información viaje por la red entre un cliente cualquiera y nuestro servidor, estos datos estarán totalmente protegidos.

## **6.5 Retrospective**

Para la realización del microservicio de pagos, ha surgido la necesidad de atajar un problema de tiempo en relación con la tarea de mockear un API que sustituya las respuestas de la plataforma *MasterCard*. El *work around* empleado ha sido el de implementar un servicio que genere respuestas con valores parcialmente aleatorios, es decir, en la generación de identificadores para “*Donor*”, “*Cards*” y “*One Time*” (cumpliendo con las características que podemos consultar del portal del desarrollador de *MasterCard*, concretamente del API *Donate API*)

Por otro lado, durante la intercepción de todas las llamadas recibidas a nuestro sistema, podemos observar que ningún dato sensible se muestra en claro y, por tanto, supongan un riesgo de exposición frente a cualquier ciberdelincuente que los lea.

Lo que sí se ha podido comprobar es que la autenticación, en una comunicación HTTP dentro de una red local como es la utilizada en este trabajo, se puede interceptar las credenciales en claro de un usuario. En el apartado de [intercepción de la petición de autorización](#), se observamos que, en el cuerpo de la llamada, se puede leer la contraseña del usuario. Aún así, debemos contar que esta llamada, en un entorno real, se realizaría a través de comunicaciones HTTPS y que sólo sería válida la petición si es acompañada por las credenciales de algún [cliente autorizado para usar nuestro servidor](#).

## **7. Conclusiones**

A lo largo de este TFG, he aprendido realmente la importancia de las etapas iniciales de un proyecto: el definir la naturaleza del proyecto, identificar los objetivos que se quieren cumplir, estudiar las tecnologías que se adaptarán mejor para cumplir los requisitos que se desean, el análisis y diseño técnico del producto que queremos obtener... todos estos aspectos son decisivos para el éxito.

Y no con éxito querría referirme al hecho de terminar con un producto final aceptable, o con unos objetivos cumplidos, sino a que a eso y mucho más, como un desarrollo “sencillo y saludable” de todo lo que ha supuesto la realización de este proyecto. Todas las tomas de decisiones que he llevado a cabo me han resultado fáciles de ejecutar, o cómo el uso correcto de todo un conjunto de herramientas destinadas a la gestión de proyectos han sido claves para poder llevar un ritmo y rumbo correctos en cada *sprint*.

El uso de la metodología Scrum me ha permitido poder construir poco a poco y sin pausa todo el sistema, cumpliendo las fechas a pesar de los contratiempos vividos. Esto ha sido gracias a esa capacidad de adaptación al cambio que, en parte, caracteriza esta forma de trabajar. Así he sido capaz de afrontar todos los imprevistos que se han presentado debido, por ejemplo, a errores de estimación también muy presentes en este ámbito.

Después de estos cuatro *sprints* puedo afirmar que he cumplido con el objetivo inicialmente planteado en la introducción. Está claro que en este campo que nos ocupa en la Ingeniería del Software hay mil maneras de lograr un objetivo y, al igual que en la cocina cada uno se prepara una tortilla a su manera, pero, al fin y al cabo, una tortilla es una tortilla; en este TFG he demostrado una forma de proteger la información que puedan intercambiarse dos interlocutores en una red frente a ciber ataques del tipo *MITM*.

Como líneas de futuro que no he podido tomar en cuenta en este proyecto, destacaría la implementación de todos los tests unitarios para garantizar una cobertura amplia del proyecto o el análisis estático de nuestro código con herramientas como *SonarQube*, *Kiuwan* o *Dependency Check* para mejorar la calidad y seguridad del código.

## 8. Glosario

- SM – *Scrum Master*
- PO – *Product Owner*
- SGBD – Sistema gestor base de datos
- MITM – Acrónimo de un ciber ataque del tipo “*Man in the Middle*”
- JWT – Acrónimo de *JSON Web Token*
- JWS – Acrónimo de *JSON Web Signature*
- JWE – Acrónimo de *JSON Web Encryption*
- *Mockeado* – Del término *mock*, hace referencia a un objeto el cual se ha falsificado su comportamiento de manera que simula el del objeto original sin serlo realmente.
- *Work around* – Solución temporal a un problema que de alguna u otra manera la solución más correcta no se puede llevar a cabo por falta de recursos en el momento en el que se identifica.

## 9. Bibliografía

Instituto Nacional de Ciberseguridad. (4 de Marzo de 2022). *incibe*. Obtenido de <https://www.incibe.es/protege-tu-empresa/blog/el-ataque-del-man-middle-empresa-riesgos-y-formas-evitarlo>

## 10. Anexos

Colecciones y entornos de Postman

Colección API Pagos

```
{
  "info": {
    "_postman_id": "77610fd6-ad0e-4978-bcb8-7f8840767cfa",
    "name": "API Pagos (Gateway)",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
  "item": [
    {
      "name": "OAuth",
      "event": [
        {
          "listen": "test",
          "script": {
            "id": "52c202d7-aef9-45d4-b68a-6c2e49c9b698",
            "exec": [
              "var jsonData = JSON.parse(responseBody);\r\n",
              "pm.environment.set(\"access_token\", jsonData.access_token);\r\n",
              "pm.environment.set(\"jwe_data\", jsonData.jweData);\r\n",
              ""
            ],
            "type": "text/javascript"
          }
        }
      ],
      "id": "e25e9983-1ffe-453a-88f4-d977da1f3040",
      "protocolProfileBehavior": {
        "disableBodyPruning": true,
        "disableUrlEncoding": true,
        "followOriginalHttpMethod": false,
        "strictSSL": true
      },
      "request": {
        "auth": {
          "type": "basic",
          "basic": [
            {
              "key": "username",
              "value": "{{client_gateway}}",
              "type": "string"
            },
            {
              "key": "password",
              "value": "{{client_pwd}}",
              "type": "string"
            }
          ]
        }
      },
      "method": "POST",
      "header": [
      ],
      "body": {
        "mode": "urlencoded",
        "urlencoded": [

```

```

        {
            "key": "username",
            "value": "{{user_email}}",
            "type": "text"
        },
        {
            "key": "password",
            "value": "{{user_pwd}}",
            "type": "text"
        },
        {
            "key": "grant_type",
            "value": "password",
            "type": "text"
        }
    ]
},
"url": {
    "raw": "{{host}}/api/oauth/oauth/token",
    "host": [
        "{{host}}"
    ],
    "path": [
        "api",
        "oauth",
        "oauth",
        "token"
    ]
}
},
"response": [
]
},
{
    "name": "Create Donor",
    "event": [
        {
            "listen": "prerequisite",
            "script": {
                "id": "216ad02f-bc1c-474d-8beb-599a4dac39f4",
                "exec": [
                    ""
                ],
                "type": "text/javascript"
            }
        },
        {
            "listen": "test",
            "script": {
                "id": "22b83716-d6b2-4a00-8a58-068dc0393f13",
                "exec": [
                    "var jsonData = JSON.parse(responseBody);\r\n",
                    "pm.environment.set(\"donor_id\", jsonData.donorId);"
                ],
                "type": "text/javascript"
            }
        }
    ],
    "id": "a26c44bf-0359-4263-b4e6-9fbd53d05c8c",
    "protocolProfileBehavior": {
        "disableBodyPruning": true
    },
    "request": {
        "auth": {
            "type": "bearer",
            "bearer": [
                {
                    "key": "token",
                    "value": "{{access_token}}",
                    "type": "string"
                }
            ]
        }
    }
},
},

```

```

"method": "POST",
"header": [
  {
    "key": "X-JWE-Data",
    "value": "{{jwe_data}}",
    "type": "text"
  }
],
"body": {
  "mode": "raw",
  "raw": "{\r\n  \"email\": \"vmedinap@uoc.edu\", \r\n  \"locale\": \"es-ES\", \r\n
  \"acceptedConsents\": true\r\n}",
  "options": {
    "raw": {
      "language": "json"
    }
  }
},
"url": {
  "raw": "{{host}}/api/payments/payments/donor",
  "host": [
    "{{host}}"
  ],
  "path": [
    "api",
    "payments",
    "payments",
    "donor"
  ]
}
},
"response": [
]
},
{
  "name": "Create Card",
  "id": "4cf522fa-5cc9-42d6-96b4-205446213fbe",
  "protocolProfileBehavior": {
    "disableBodyPruning": true
  },
  "request": {
    "auth": {
      "type": "bearer",
      "bearer": [
        {
          "key": "token",
          "value": "{{access_token}}",
          "type": "string"
        }
      ]
    },
    "method": "POST",
    "header": [
      {
        "key": "X-JWE-Data",
        "value": "{{jwe_data}}",
        "type": "text"
      }
    ],
    "body": {
      "mode": "raw",
      "raw": "{\r\n  \"primaryAccountSuffix\": \"*****4444\"\r\n}",
      "options": {
        "raw": {
          "language": "json"
        }
      }
    },
    "url": {
      "raw": "{{host}}/api/payments/payments/card",
      "host": [
        "{{host}}"
      ],

```



```

        "path": [
            "api",
            "payments",
            "payments",
            "card"
        ]
    },
    "response": [
    ]
},
{
    "name": "Create OneTime Transaction",
    "id": "778caae4-cc6c-4065-bdff-18a9319b3325",
    "protocolProfileBehavior": {
        "disableBodyPruning": true
    },
    "request": {
        "auth": {
            "type": "bearer",
            "bearer": [
                {
                    "key": "token",
                    "value": "{{access_token}}",
                    "type": "string"
                }
            ]
        },
        "method": "POST",
        "header": [
            {
                "key": "X-JWE-Data",
                "value": "{{jwe_data}}",
                "type": "text"
            }
        ],
        "body": {
            "mode": "raw",
            "raw": "{\r\n  \"primaryAccountSuffix\": \"*****4444\", \r\n  \"charityId\":  

\"ae3c5f35-084a-4df9-a2af-7e0efd512fc0\", \r\n  \"amount\" : 100.46\r\n}",
            "options": {
                "raw": {
                    "language": "json"
                }
            }
        },
        "url": {
            "raw": "{{host}}/api/payments/payments/one_time",
            "host": [
                "{{host}}"
            ],
            "path": [
                "api",
                "payments",
                "payments",
                "one_time"
            ]
        }
    },
    "response": [
    ]
}
]
}
}

```

## Entorno API Pagos

```
{
  "id": "6e293db1-3550-49b8-a1c0-33d7171e2839",
  "name": "API Pagos - Local",
  "values": [
    {
      "key": "host",
      "value": "http://localhost:8090",
      "enabled": true,
      "type": "default"
    },
    {
      "key": "access_token",
      "value": "",
      "enabled": true,
      "type": "default"
    },
    {
      "key": "client_gateway",
      "value": "payment-service",
      "enabled": true,
      "type": "default"
    },
    {
      "key": "client_pwd",
      "value": "12345",
      "enabled": true,
      "type": "default"
    },
    {
      "key": "user_email",
      "value": "vmedinap@uoc.edu",
      "enabled": true,
      "type": "default"
    },
    {
      "key": "user_pwd",
      "value": "12345",
      "enabled": true,
      "type": "default"
    },
    {
      "key": "jwe_data",
      "value": "",
      "enabled": true,
      "type": "default"
    }
  ]
}
```

## Scripts SQL

### Creación de esquema Main

```
--- DROP TABLES
drop table usuarios_roles;
drop table tarjetas;
drop table roles;
drop table usuarios;

-- CREATE TABLES
CREATE TABLE usuarios (
  usuario_id varchar(12) not null primary key,
  nombre varchar(20) not null,
  apellido varchar(30) not null,
  password varchar(60) not null,
  email varchar(60) not null,
  enabled boolean not null,
  UNIQUE(email)
);

CREATE TABLE roles (
  role_id varchar(12) not null primary key,
  nombre varchar(25) not null
);

CREATE TABLE tarjetas (
  tarjeta_id int8 not null primary key,
  usuario_id varchar(12) not null,
  titular varchar(50) not null,
  pan varchar(16) not null,
  expiry_month varchar(2) not null,
  expiry_year varchar(4) not null,
  FOREIGN KEY (usuario_id) REFERENCES usuarios(usuario_id)
);

CREATE TABLE usuarios_roles (
  usuario_id varchar(12) not null,
  role_id varchar(12) not null,
  FOREIGN KEY (usuario_id) REFERENCES usuarios(usuario_id),
  FOREIGN KEY (role_id) REFERENCES roles(role_id),
  UNIQUE (usuario_id, role_id)
);
```

### Creación de esquema Payments

```
--- DROP TABLES
drop table one_time_donation;
drop table monthly_donation;
drop table guest_donation;
drop table charities;
drop table cards;
drop table donors;

-- CREATE TABLES
CREATE TABLE donors (
  donor_id varchar(36) not null primary key,
  usuario_id varchar(12) not null,
  userName varchar(60) not null,
  userType varchar(30) not null,
  email varchar(60) not null,
  locale varchar(5) not null,
  name varchar(30),
  acceptedConsents boolean not null,
  UNIQUE(usuario_id),
  UNIQUE(userName),
  UNIQUE(email)
);

CREATE TABLE cards (
  card_id varchar(36) not null primary key,
  donor_id varchar(36) not null,
  primaryAccountSuffix varchar(4) not null,
```

```

FOREIGN KEY (donor_id) REFERENCES donors(donor_id)
);

CREATE TABLE charities (
  charity_id varchar(36) not null primary key,
  name varchar(60) not null,
  description varchar(120) not null,
  enabled boolean not null
);

CREATE TABLE one_time_donation (
  transaction_id varchar(36) not null primary key,
  donor_id varchar(36) not null,
  charity_id varchar(36) not null,
  card_id varchar(36) not null,
  amount numeric(4,2) not null,
  transactionStatus varchar(8) not null,
  FOREIGN KEY (donor_id) REFERENCES donors(donor_id),
  FOREIGN KEY (charity_id) REFERENCES charities(charity_id),
  FOREIGN KEY (card_id) REFERENCES cards(card_id)
);

CREATE TABLE monthly_donation (
  transaction_id varchar(36) not null primary key,
  donor_id varchar(36) not null,
  charity_id varchar(36) not null,
  card_id varchar(36) not null,
  setup_id varchar(36) not null,
  setupStatus varchar(7) not null,
  donationType varchar(8) not null,
  amount numeric(4,2) not null,
  transactionStatus varchar(8) not null,
  FOREIGN KEY (donor_id) REFERENCES donors(donor_id),
  FOREIGN KEY (charity_id) REFERENCES charities(charity_id),
  FOREIGN KEY (card_id) REFERENCES cards(card_id),
  UNIQUE(setup_id)
);

CREATE TABLE guest_donation (
  transaction_id varchar(36) not null primary key,
  donor_id varchar(36),
  charity_id varchar(36) not null,
  card_id varchar(36) not null,
  amount numeric(4,2) not null,
  transactionStatus varchar(8) not null,
  donationType varchar(8) not null,
  email varchar(60) not null,
  FOREIGN KEY (charity_id) REFERENCES charities(charity_id),
  FOREIGN KEY (card_id) REFERENCES cards(card_id)
);

```

## Inserción de datos iniciales en BD

```

INSERT INTO main.roles
(role_id, nombre)
VALUES('USER', 'ROLE_USER');

INSERT INTO main.roles
(role_id, nombre)
VALUES('ADMIN', 'ROLE_ADMIN');

INSERT INTO main.usuarios
(usuario_id, nombre, apellido, "password", email, enabled)
VALUES('USER-1', 'Víctor', 'Medina',
'$2a$10$59gEyVnZsWdfXFPqJQxBAONRGEZ3sJGJc7jDzgaTsmc879X/aDP0S', 'vmedinap@uoc.edu', true);

INSERT INTO main.usuarios
(usuario_id, nombre, apellido, "password", email, enabled)
VALUES('USER-2', 'Wilson', 'Medina',
'$2a$10$Ty11wkuRlReE6emWkm97Zef0g0vEdB9zyMao6udPAm3JMYRTIRfWS', 'wilson@email.com', true);

INSERT INTO main.usuarios

```

```

(usuario_id, nombre, apellido, "password", email, enabled)
VALUES('USER-3', 'Río', 'Medina',
'$2a$10$5TZHga5gdq7rYTzkq8WTre/4N/2NjxiRBUabFkmsFUF2N3.Cgir5i', 'rio@email.com', true);

INSERT INTO main.tarjetas (tarjeta_id,usuario_id,titular,pan,expiry_month,expiry_year)
VALUES(0,'USER-1','Victor Medina Perez','1111222233334444','1','2040');

INSERT INTO main.tarjetas (tarjeta_id,usuario_id,titular,pan,expiry_month,expiry_year)
VALUES(1,'USER-1','Victor Medina Perez','2222333344445555','2','2040');

INSERT INTO main.tarjetas (tarjeta_id,usuario_id,titular,pan,expiry_month,expiry_year)
VALUES(2,'USER-1','Victor Medina Perez','3333444455556666','3','2040');

INSERT INTO main.tarjetas (tarjeta_id,usuario_id,titular,pan,expiry_month,expiry_year)
VALUES(3,'USER-2','Wilson Medina Perez','4444555566667777','4','2040');

INSERT INTO main.tarjetas (tarjeta_id,usuario_id,titular,pan,expiry_month,expiry_year)
VALUES(4,'USER-3','Río Medina Perez','5555666677778888','5','2040');

INSERT INTO main.usuarios_roles
(usuario_id, role_id)
VALUES('USER-1', 'ADMIN');

INSERT INTO main.usuarios_roles
(usuario_id, role_id)
VALUES('USER-2', 'USER');

INSERT INTO main.usuarios_roles
(usuario_id, role_id)
VALUES('USER-3', 'USER');

```