SPS

PRÁCTICA MULESOFT TRAINEE

Documentación del proceso seguido para su resolución

JUNIO DE 2025

Preparado por

ÁNGEL DE LA CRUZ

PROPORCIONADO POR

LILIANA CRUZ

CONTENIDOS

1 INTRODUCCIÓN

1.1 Contexto

El presente documento es uno de los entregables de una prueba técnica realizada en SPS, que tiene como objetivo demostrar las habilidades de autoaprendizaje y solución de problemas a través de la exploración de una tecnología llamada "MuleSoft".

El objetivo de este documento es presentar el proceso seguido para la resolución de la práctica. Principalmente, plasmar en forma de guía el desarrollo de esta, pensando que un lector la pueda seguir paso a paso para replicarla y obtener los mismos resultados.

1.2 Resumen

Los pasos seguidos para la resolución de la práctica fueron los siguientes:

1. **Registro de métricas**: medir permite mejorar. Pensando en esto, seguí *PSP* (*Personal Software Process*), donde la métrica principal que registré fue el tiempo en minutos invertido en la solución. Como resumen:
    a. Familiarización con a herramienta y configuración del ambiente: 194 minutos (poco más de 3 horas).
    b. Implementación de solución base: 481 minutos (poco más de 8 horas).
    c. Exploración extendida sobre MuleSoft y mejora de la solución: 639 minutos (casi 11 horas).

    Vale la pena destacar que los tiempos de implementación incluyen la documentación paralela del proceso.

2. **Familiarización con la herramienta**: como paso previo al desarrollo de la práctica, consulté los recursos de aprendizaje recomendados y algunos otros más para comprender MuleSoft y las diversas herramientas que lo conforman, como *Anypoint Studio*, *Anypoint Platform* y *CloudHub*.
3. **Desarrollo base**: una vez comprendido el entorno de desarrollo y el objetivo de cada una de las herramientas, implementé la solución base solicitada. Es decir, la configuración del proyecto con los archivos de propiedades y propiedades seguras, el archivo global, la creación de la especificación de la API y el despliegue de esta en *CloudHub*.
4. **Mejoras de la solución**: habiendo solucionado la práctica, exploré conceptos extra para mejorar la solución. Estos fueron: la creación de una API en *API*

Manager y la conexión del proyecto con esta a través de *Autodiscovery,* con el fin de implementar la política de *client-id Enforcement*, la integración de paginación, monitoreo básico, validación de parámetros y manejo centralizado de errores en el *endpoint*.

5. **Análisis final del proceso**: como parte final de la solución, realicé un breve análisis del proceso seguido, de la percepción sobre la herramienta, las dificultades presentadas y de lo aprendido al utilizarla. Esto, se ve reflejado en la conclusión de este documento.

1.3 Evidencias

Como resultado del desarrollo de la práctica y acompañando al presente documento, se encuentran algunas otras evidencias:

1. Repositorio de código de GitHub,
2. Especificación de la API en Exchange y
3. API desplegada en CloudHub: https://spsmarketingapi-ar49hp.5sc6y6-4.usa-e2.cloudhub.io

   Para el uso de esta API, se necesita de autenticación básica de HTTP. A continuación se muestran las credenciales para probarla.

   Username: 31886cf1cbeb486d9b4fe3229fe42133

   Password: C2D642035Ba84025bc074a497668FFc1

2 DESARROLLO

Esta sección es la más larga del documento, pues está pensada como una guía que permita que otro desarrollador principiante la siga paso a paso para llegar a la misma solución o una muy similar.

2.1 Prerrequisitos

Para poder seguir el desarrollo de la práctica, es necesario:

1. Tener un REST Client instalado y configurado, preferentemente Postman, pues este es el utilizado a lo largo de toda la práctica.
2. Tener una cuenta en CloudHub.
3. Instalar y configurar Anypoint Studio.
4. Instalar y configurar Java localmente.
5. Tener descargado el archivo .jar para encriptar cadenas de texto, proporcionado por Mulesoft.

2.2 Implementación base

2.2.1 Configuración del proyecto

El primer paso, previo a la implementación de la API, es crear el proyecto Mule. Para ello, abriremos *Anypoint Studio* y en la pantalla principal, seleccionaremos la opción *Create a Mule Proyect*. En caso de no tener esta opción como se muestra en la imagen, accederemos a ella desde *File* > *New* > *Mule Project*.

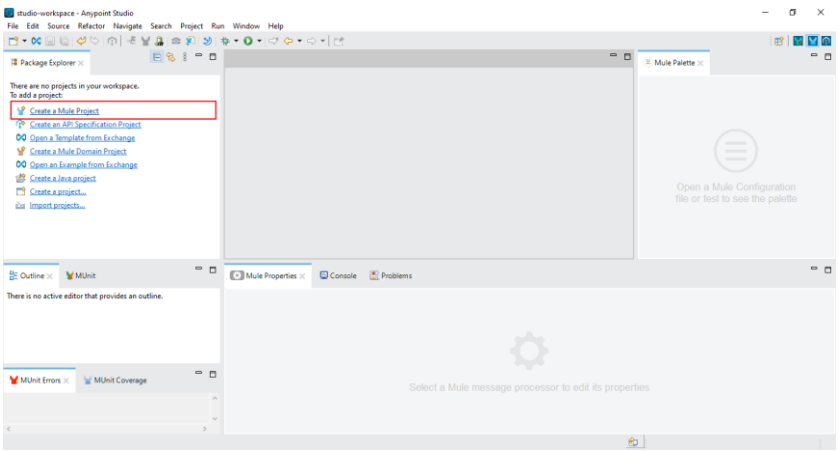

Figura 1. Ventana inicial de Anypoint Studio, que presenta la opción para crear un nuevo proyecto Mule

Esto abrirá una ventana de configuración del proyecto, donde deberemos especificar el nombre del proyecto, en este caso, *SPSMarketingAPI*. Como el proyecto lo crearemos desde cero, la sección de *API Implementation* la dejaremos con sus valores

por defecto. Lo mismo para la sección de *Project Location*, pues no modificaremos la ubicación del proyecto. Finalmente, daremos clic en el botón *Finish*.

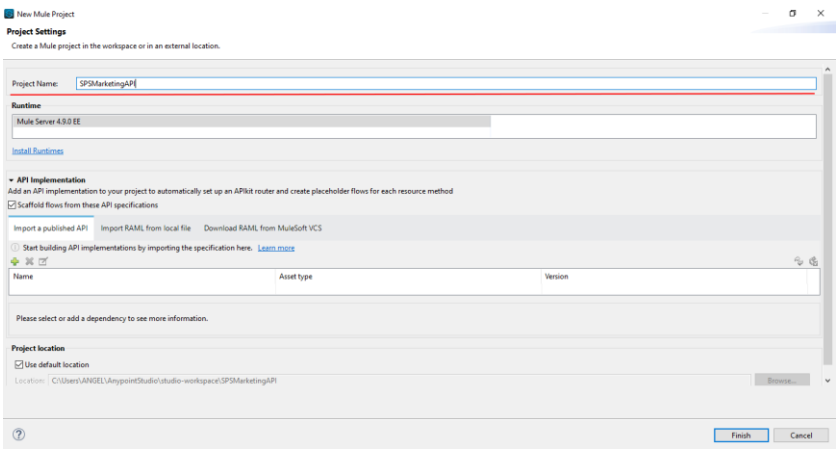

Figura 2. Ventana de configuración de nuevo proyecto Mule con valores por defecto, a excepción del nombre de proyecto

Con esto, habremos configurado el proyecto para implementar nuestra solución.

2.2.2 Configuración de archivos de propiedades

Como paso siguiente, declararemos algunas "variables" que serán de utilidad en la configuración de los elementos necesarios para la creación de la API. Esto, a través de archivos de propiedades. Para ello, daremos clic derecho en la carpeta *src/main/resources* dentro del explorador de paquetes y seleccionaremos *New* > *File*. Esto abrirá una ventana de configuración de archivo, donde le daremos el nombre *local.properties* y finalizaremos.

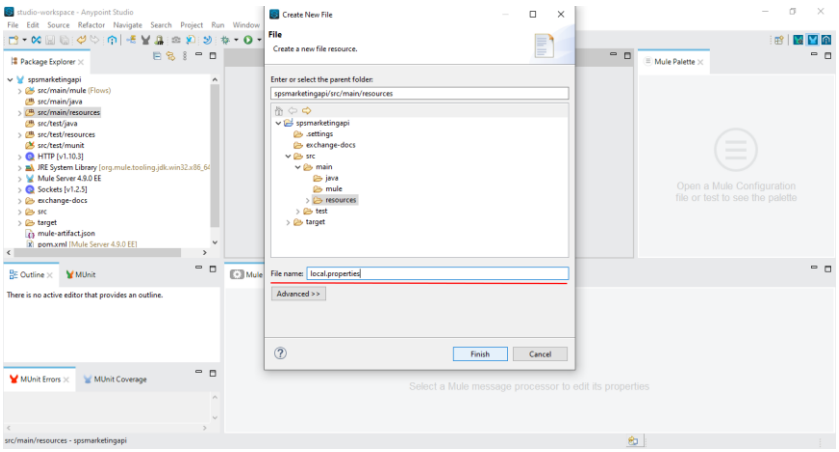

Figura 3. Ventana para la configuración de un nuevo archivo, en este caso, local.properties

Esto creará un archivo con el nombre especificado, dentro de la carpeta *src/main/resources*, en el cual especificaremos las propiedades de utilidad, no sin

antes repetir el proceso, pero ahora con un archivo nombrado *dev.properties*. Esto permitirá definir las mismas propiedades, pero con distintos valores según el entorno en el que se ejecuta la aplicación.

En este sentido, deberemos definir tres propiedades importantes en el contexto de esta práctica: *http.listener.host*, *http.listener.port* y *apiv1.path.base*. Cada una con los valores *0.0.0.0*, *8081* y */api/v1/sps*, respectivamente. Todas ellas serán de utilidad para configurar la exposición de la API, como lo veremos más adelante. Para esto, abriremos ambos archivos y escribiremos cada una de las propiedades siguiendo la sintaxis *propiedad=valor*, cada una de ellas en su propia línea.

En este caso específico, todas las propiedades tendrán el mismo valor en ambos archivos, sin embargo, esto se debe a que ambas configuraciones no cambian según el entorno.

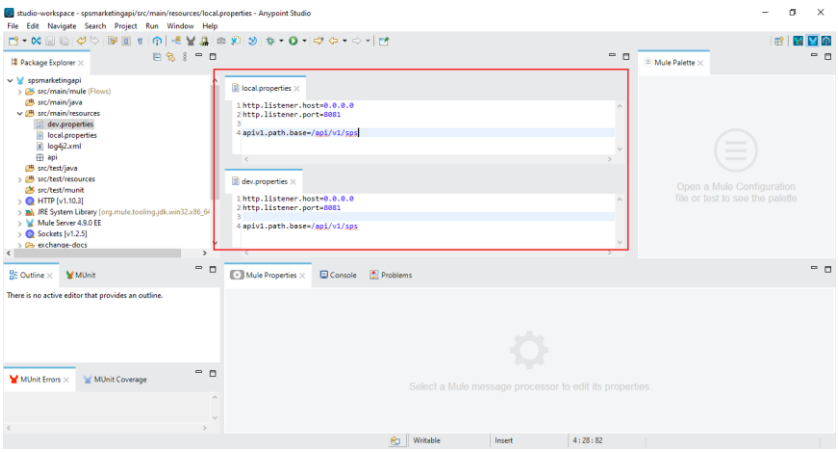

Figura 4. Configuración de archivos de propiedades

2.2.3 Definición de configuración global general

Con el proyecto creado y las propiedades declaradas, prepararemos nuestro entorno de trabajo, separando nuestras definiciones globales de las implementaciones particulares. Para ello, daremos clic derecho en la carpeta *src/main/mule* dentro del explorador de paquetes y seleccionaremos *New* > *Mule Configuration File*. Esto abrirá una ventana de configuración de archivo, donde le daremos el nombre *global.xml* y finalizaremos.

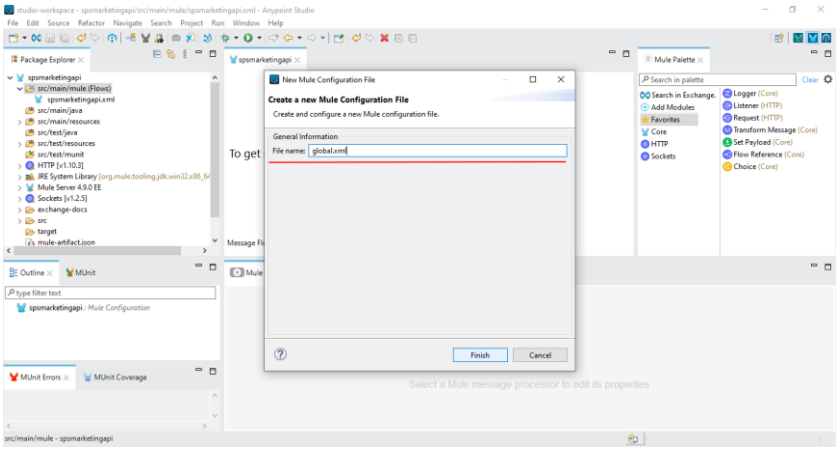

Figura 5. Ventana para la creación de un nuevo archivo de configuración Mule, para crear el archivo global.xml

Como paso siguiente, abriremos el archivo *global.xml* previamente creado y seleccionar la opción *Global elements* en el *Canvas* (parte central de la interfaz). Ahí, se nos presentará el botón *Create*, al que daremos clic y que abrirá una ventana de configuración global. Esto, para crear una nueva configuración global.

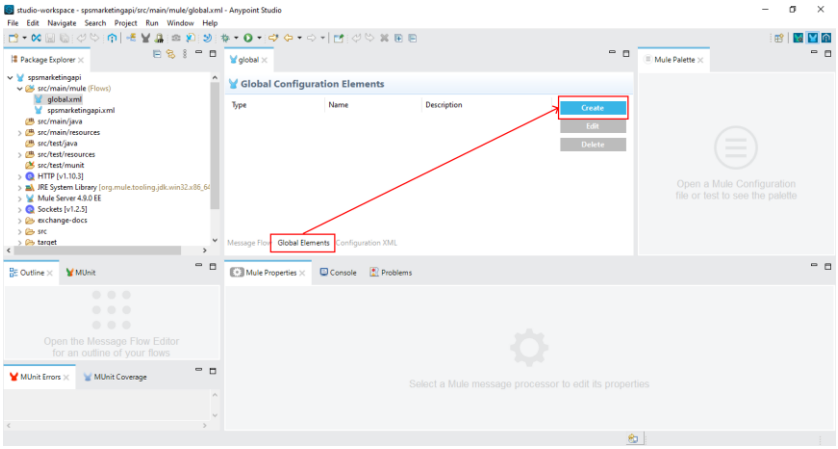

Figura 6. Interfaz de configuración global, para el registro de una nueva configuración

En la ventana de registro de nueva configuración global, deberemos buscar *Global property*, seleccionar el resultado y dar clic en *Ok*.

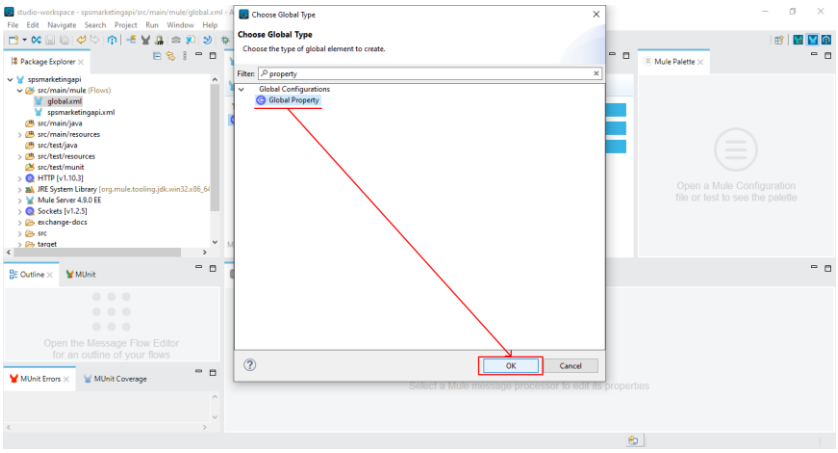

Figura 7. Ventana de registro de una nueva configuración global, en este caso, Global Property

El paso anterior abrirá una nueva ventana con dos campos: *Name* y *Value*, en los que deberemos ingresar los valores *env* y *local*, para después dar clic en *Ok* dentro de la ventana y finalizar el registro de esta propiedad. Así, habremos creado nuestra primera configuración global.

El siguiente paso será crear otra configuración global, pero esta vez deberemos buscar *Configuration properties*, seleccionar el resultado y dar clic en *Ok*. Esto abrirá una nueva ventana, en la que aparecerá el campo *File*. En este campo, deberemos ingresar el valor *${env}.properties*. Esta sintaxis es muy importante y la estaremos usando en pasos posteriores de esta práctica. Básicamente es un acceso y una concatenación: la sintaxis *${env}* le "indica" a Mule que queremos acceder al valor de la propiedad llamada *env* (la que creamos previamente).

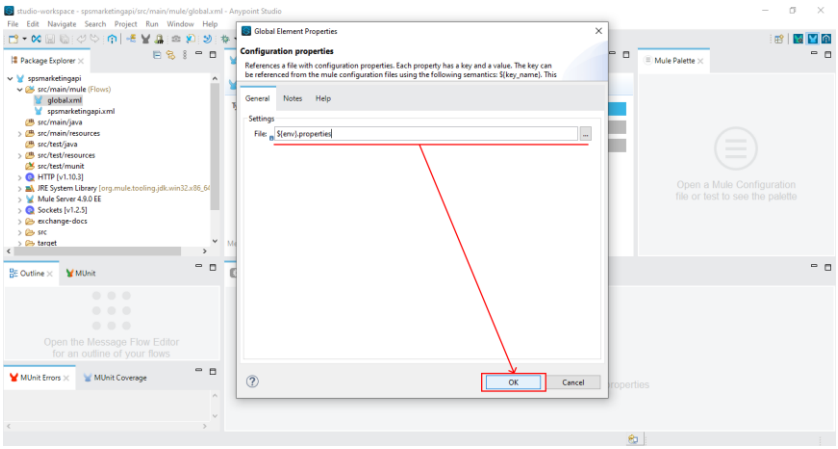

Figura 8. Ventana de configuración de propiedades globales

Estos dos pasos anteriores permiten que realmente podamos acceder a las propiedades definidas en nuestros archivos *env.properties* y *dev.properties*.

2.2.4 Configuración de archivos de propiedades seguras

Las propiedades previamente creadas son accesibles para todos, más al subir el proyecto a un repositorio de GitHub. Sin embargo, nuestras configuraciones de conexión a la base de datos no deberían estar expuestas de esta forma, por lo que deberemos configurar propiedades seguras para evitar su exposición.

El primer paso es agregar el módulo *Mule secure configuration property Extension* utilizando *Exchange*. Para esto, abriremos el archivo de configuración del proyecto *spsmarketingapi.xml* y daremos clic en *Search in Exchange* en la barra lateral derecha.

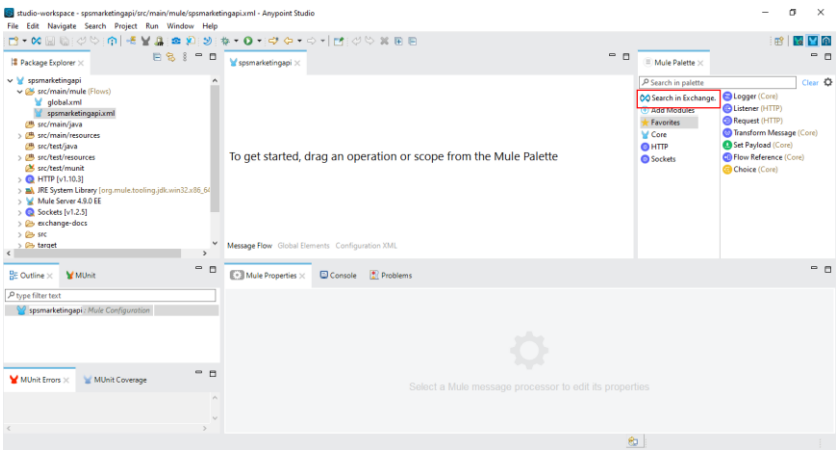

Figura 9. Opción para abrir Exchange y buscar los módulos de interés

El paso anterior abrirá una nueva ventana para agregar dependencias al proyecto, en la que deberemos buscar *Mule secure configuration property Extension*, seleccionarla, agregarla y finalizar.

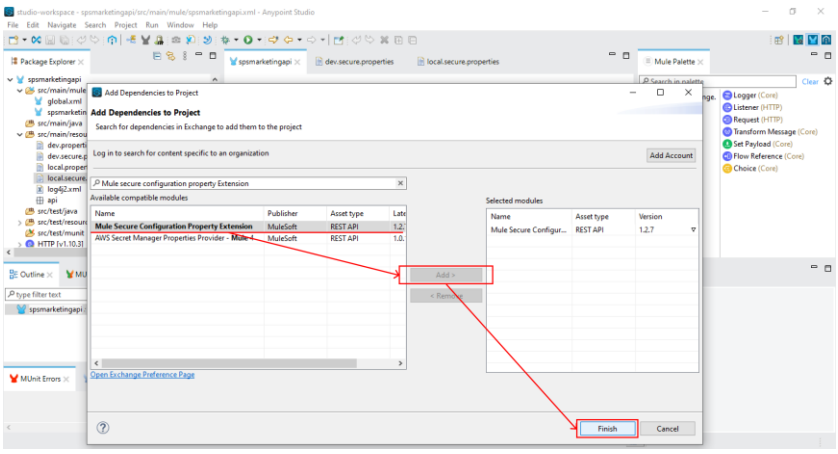

Figura 10. Ventana para agregar dependencias al proyecto

Seguido de esto, deberemos crear una nueva configuración global. Buscaremos *Secure properties configuration*, seleccionaremos la opción y daremos clic en *Ok*. Esto

abrirá la ventana para crear una nueva configuración global asociada con las propiedades seguras.

En la ventana de configuración de propiedades seguras, colocaremos *${env}.secure.properties* en el campo *File* (similar a la configuración de propiedades previa), en el campo *Key*, el valor *${secure.key}* y seleccionaremos *Blowfish* como algoritmo (campo *Algorithm*). La propiedad *secure.key* la definiremos posteriormente al momento de ejecutar la aplicación.

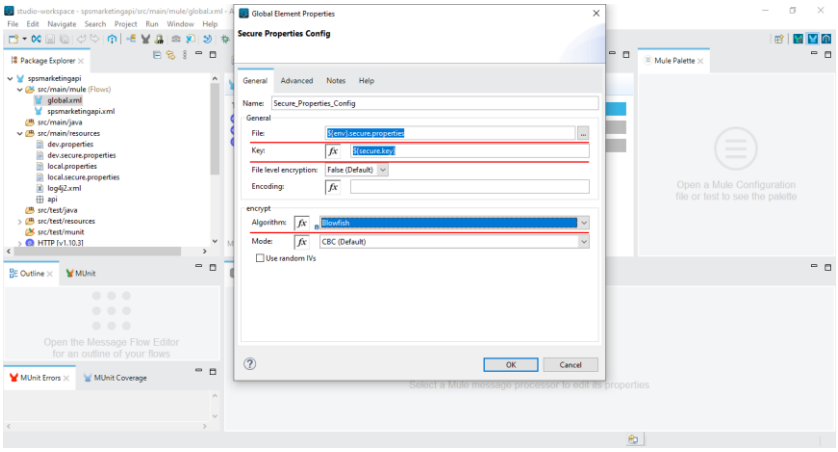

Figura 11. Ventana de configuración de propiedades seguras

Lo siguiente es generar los archivos que almacenarán las propiedades seguras. Siguiendo los mismos pasos que ya abordamos en la creación de archivos de propiedades, crearemos dos archivos: *local.secure.properties* y *dev.secure.properties*. En ellos, colocaremos las propiedades *db.host*, *db.port*, *db.user*, *db.password* y *db.name*, pero sin colocarles ningún valor aún.

Hasta ahora, ya tenemos los archivos que almacenarán las propiedades seguras, así como la configuración dentro del proyecto para trabajar con ellas. Sin embargo, las propiedades no tienen un valor asignado. Para esto, utilizaremos el archivo *.jar* para encriptar los valores de las propiedades.

Deberemos abrir una terminal en donde se encuentra el archivo .jar y ejecutar:

```
java -cp secure-properties-tool.jar com.mulesoft.tools.SecurePropertiesTool \
string \
encrypt \
Blowfish \
CBC \
password \
"some value to encrypt"
```

Esto lo haremos para cada propiedad que tenemos en nuestros archivos *.secure.properties*, pasándole entre las comillas, sus valores reales. El valor *password*, puede ser una contraseña segura propia; al final, es este el valor el que se trazará a la propiedad *secure.key* y que servirá para desencriptar las propiedades dentro de la aplicación.

El comando previo, producirá una cadena con el valor encriptado. Estos valores, deberemos colocarlos en nuestros archivos *.secure.properties*, en las propiedades respectivas, pero esta vez rodeados de *![]*.

Ejemplo: *db.host=![AfRLjmhnzSje8nZ5Ym/0c8Jsa3cr1FDa]*

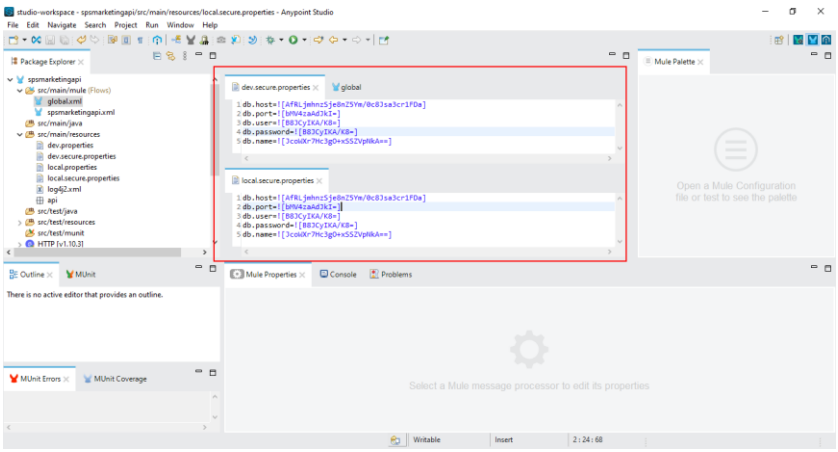

Figura 12. Archivos de propiedades seguras tras aplicar la encriptación con el archivo .jar

Como paso final de esta configuración de propiedades seguras, configuraremos la propiedad *secure.key*. Para ello, deberemos dar clic derecho en el nombre del proyecto y seleccionar *Run as* > *Run configurations*…. Esto abrirá una ventana de gestión de configuraciones de ejecución, donde deberemos dar doble clic en *Mule applications*, colocar el nombre del proyecto (spsmarketingapi) en el campo *Name* y seleccionar la opción *Environment*. Así, se mostrará un menú para registrar propiedades, por lo que ahí deberemos dar clic en *Add*….

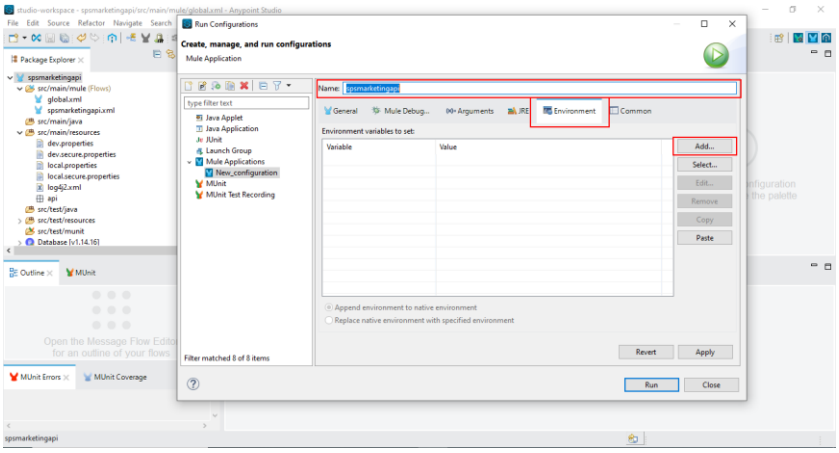

Figura 13. Ventana para la gestión de configuraciones de ejecución

El paso previo abrirá un pequeño diálogo con dos campos: *Name* y *Value*. En el primero, registraremos *secure.key* y en el segundo, *password* o cualquiera que haya sido la contraseña utilizada al ejecutar el archivo .jar.

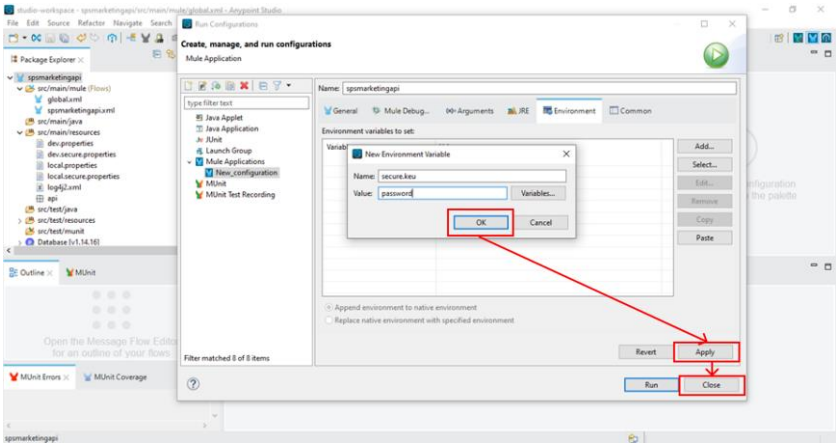

Figura 14. Ventana de registro de variables de entorno

Con estos valore especificados, ya solo queda hacer clic en *Ok*, después en *Apply*, para aplicar los cambios y finalmente en *Close*, para cerrar la ventana de gestión de configuración. Así, la variable de entorno quedará registrada y las propiedades seguras podrán ser accedidas al ejecutar el proyecto.

2.2.5 Preparación del entorno para la implementación (configuraciones globales)

Pasando a las configuraciones globales propias del funcionamiento de la API, deberemos crear dos de ellas. Primero, una *HTTP Listener Configuration* para exponer la API; segundo, una *Database Configuration* para conectarnos a MySQL y poder recuperar la información de los clientes. ¿Por qué declaramos estas configuraciones de forma global? Porque son elementos que seguramente estarán siendo usados

repetidamente en toda la aplicación. Aunque en este momento no hay muchos *endpoints* para implementar, esta separación cobra sentido al pensar a futuro, preparando bases sólidas y estructuradas para futuras implementaciones.

Comenzando con la *HTTP Listener Configuration*, deberemos crear una nueva configuración global, en nuestro archivo *global.xml*, como lo hicimos con *Global property* y *Configuration properties*. Pero esta vez, deberemos buscar *HTTP Listener config* y seleccionar esa opción.

El paso anterior abrirá una ventana de configuración de *HTTP Listener*, donde deberemos configurar tres valores: *Name*, *Host*, *Port* y *Base path*. Les deberemos asignar los valores *HTTP_Listener_spsmarketingv1*, *${http.listener.host}*, *${http.listener.port}* y *${apiv1.path.base}*, respectivamente (un nombre y las tres propiedades de los archivos previamente definidos).

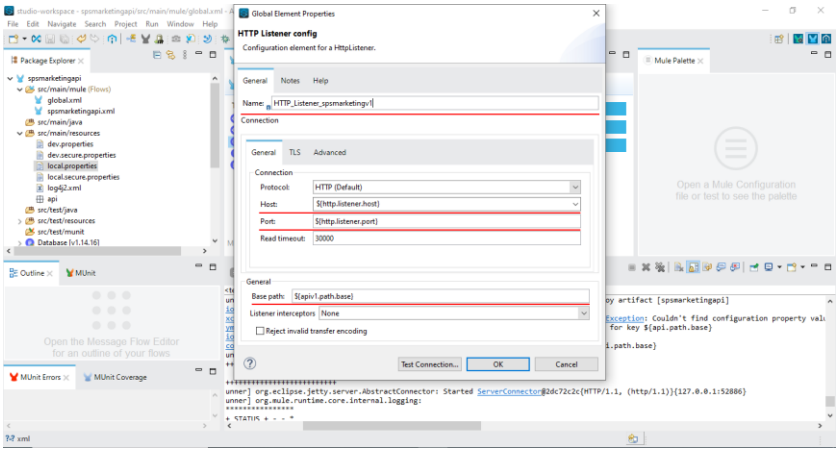

Figura 15. Ventana de configuración de HTTP Listener

Pasando a la configuración global de *Database Configuration*, primero debemos agregar las dependencias de *Database Connector* desde *Exchange*, tal como lo hicimos con *Mule secure configuration property Extension*.

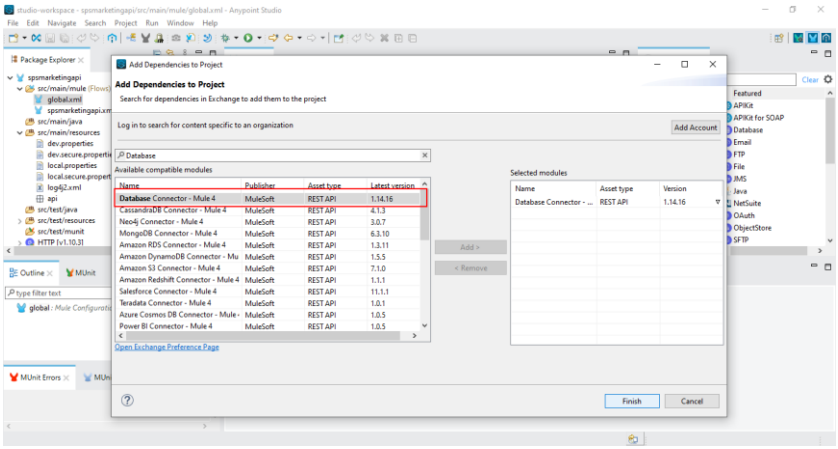

Figura 16. Uso de Exchange para agregar la dependencia Database Connector

Una vez agregada la dependencia, ya podremos ir a nuestro archivo de configuración global y registrar una nueva configuración global. Buscaremos *Database config*, la seleccionaremos y daremos clic en *Ok*.

El paso previo abrirá la configuración de la base de datos, donde seleccionaremos *MySQL Connection* en el campo *Connection*, instalaremos la biblioteca recomendada para el conector de *MySQL* y estableceremos los campos *Host*, *Port*, *User*, *Password* y *Database* con las propiedades seguras que definimos previamente. Estas propiedades se acceden también con la sintaxis *${}*, pero se les debe anteponer *secure::* al nombre para poder accederlas. Ejemplo: *${secure::db.host}*.

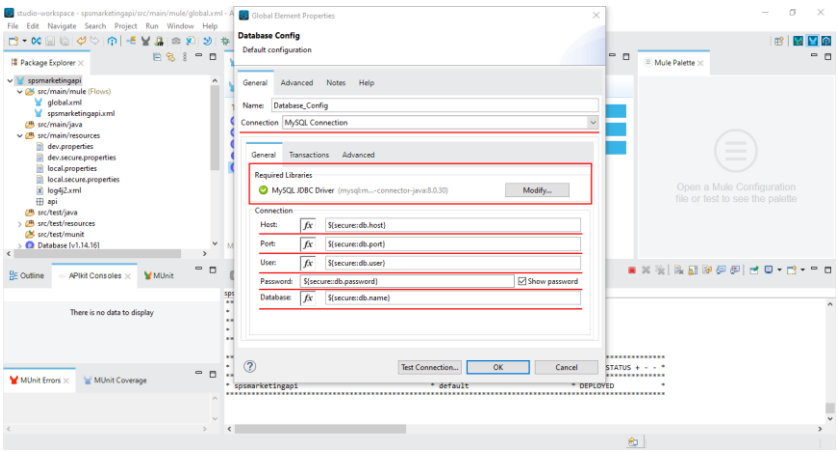

Figura 17. Ventana de configuración de base de datos

Con todas estas configuraciones, el proyecto está preparado para implementar nuestro primer flujo: un *endpoint* REST que permita recuperar la lista de clientes desde la base de datos.

2.2.6 *Implementación de endpoint base (GET /customers)*

Para comenzar con la implementación, abriremos el archivo de configuración del sistema, el que se creó en un inicio al crear el proyecto (*spsmarketingapi.xml*). Una vez abierto, buscaremos la opción *Listener (HTTP)* en el editor de propiedades (barra lateral derecha) y la arrastraremos hacia el *Canvas*. Cuando se muestre el icono de este elemento en el *Canvas*, daremos clic en él y se mostrará su configuración en la parte inferior.

Para la configuración del *Listener (HTTP)*, estableceremos las propiedades *Display Name*, *Connector Configuration* y *Path*. Para la primera y última, colocaremos los valores *GET Customer* (o un nombre autodescriptivo) y */customers*, respectivamente. Para la segunda propiedad mencionada, seleccionaremos la *HTTP Listener Configuration* previamente creada.

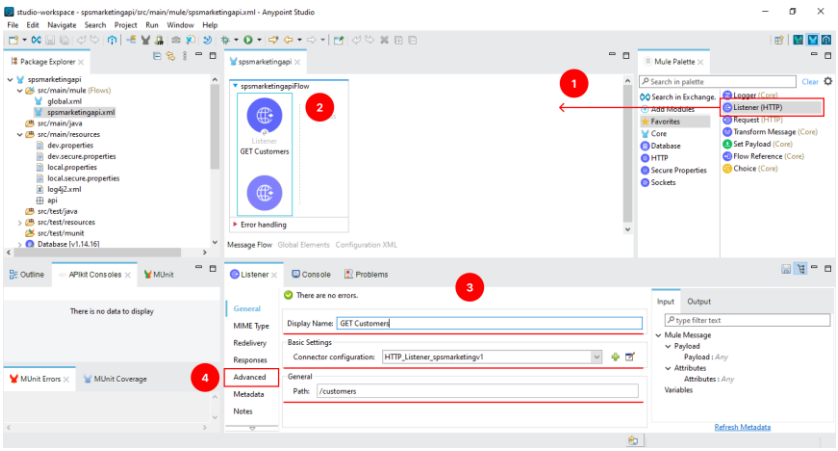

Figura 18. Configuración inicial del Listener

Seguido de esto, seleccionaremos la opción *Advanced* dentro de la configuración del *Listener (HTTP)*. En esta pestaña se mostrará el campo *Allowed Methods*, donde deberemos especificar solamente el valor *GET*, pues el endpoint tiene ese propósito, el de "obtener" o consultar los recursos de tipo *customer*.

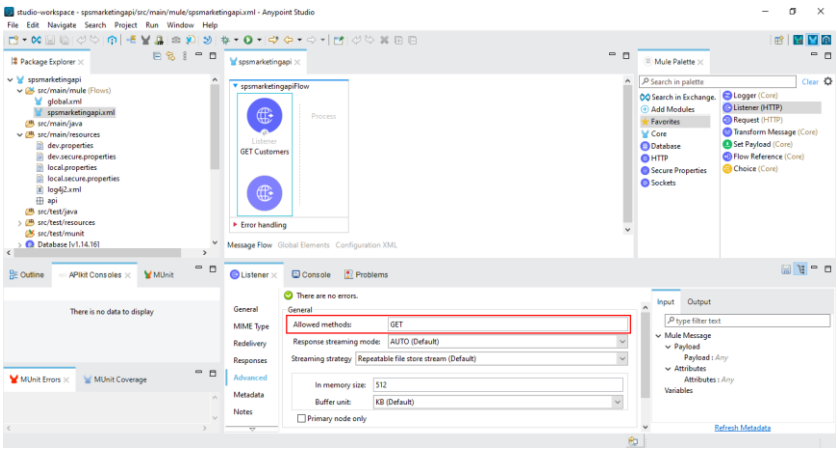

Figura 19. Configuración avanzada del Listener para especificar los métodos aceptados

Este es el primer elemento de nuestro flujo, que expone nuestro *endpoint* HTTP para su consumo. Lo siguiente en el flujo es la consulta hacia la base de datos para recuperar los clientes. Para ello, agregaremos el elemento *Select* tal como lo hicimos con el *Listener*. De igual modo, una vez en el *Canvas*, lo seleccionaremos y aplicaremos configuraciones sobre sus valores.

Modificaremos tres campos de la configuración de este nuevo elemento: *Display Name, Connector Configuration* y *SQL Query Text*. En el primero de los campos, colocaremos el valor *Select Customers* (pues esto es lo que hace el componente), en el segundo, seleccionaremos la configuración de base de datos previamente definida; finalmente, en el tercer campo, especificaremos la consulta SQL para recuperar la información de los cliente (SELECT * FROM customers).

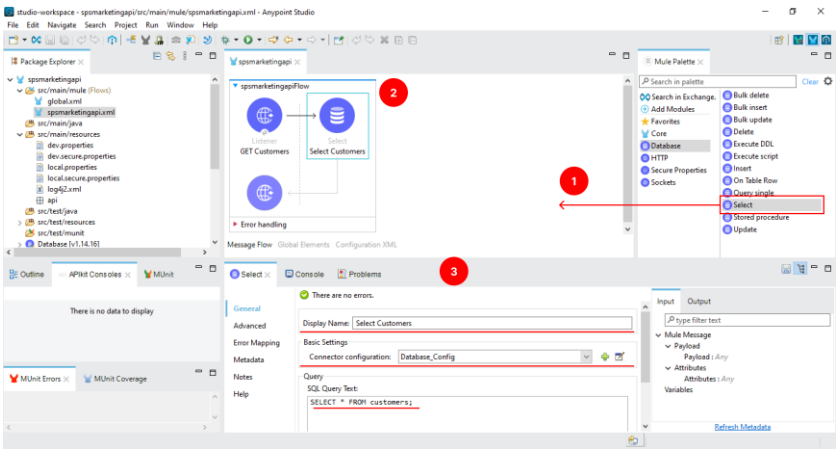

Figura 20. Configuración del elemento Select de la base de datos

Como paso final de nuestro flujo, tenemos que pasar esa información recuperada desde SQL hacia un formato JSON para poderla exponer en nuestro *endpoint*. En

este sentido, utilizaremos el componente *Transform Message*, configurado como se muestra en la siguiente imagen.

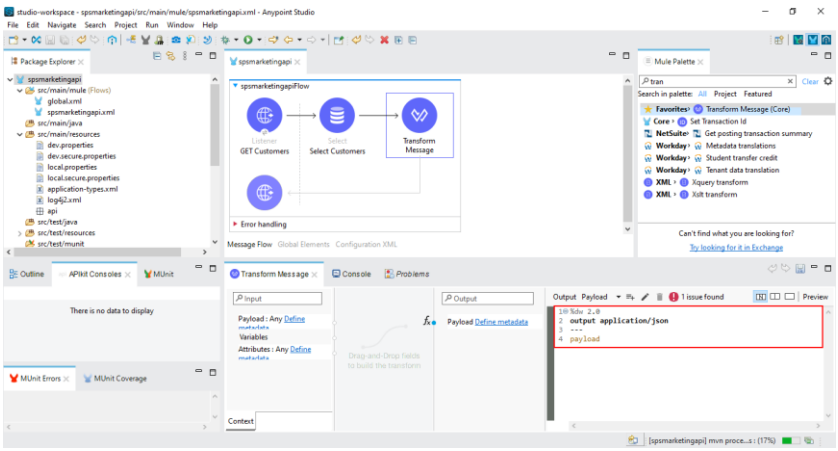

Figura 21. Configuración del componente Transform Message

El componente de la base de datos recupera la información y la organiza en objetos de Java. Lo que hace el componente *Transform Message* es tomar esa información como objetos de Java, almacenada en la variable *payload* (salida de la base de datos), y convertirla a formato JSON. La línea *output application/json* en la configuración del componente indica este cambio en formato, mientras que la línea *payload* especifica que se tome toda la información recuperada y se mande al siguiente componente, que en este caso ya es la respuesta.

Con toda esta configuración, podemos correr la aplicación desde *Run as* > *Run Configurations…*, seleccionando la configuración que realizamos previamente y dando clic en *Run*. Esto para mantener la variable *secure.key* y que todo funcione correctamente.

Finalmente, podemos probar que el *endpoint* funcione como esperamos, con algún cliente HTTP como Postman, haciendo una petición GET hacia *0.0.0.0:8081/api/v1/sps/customers*.

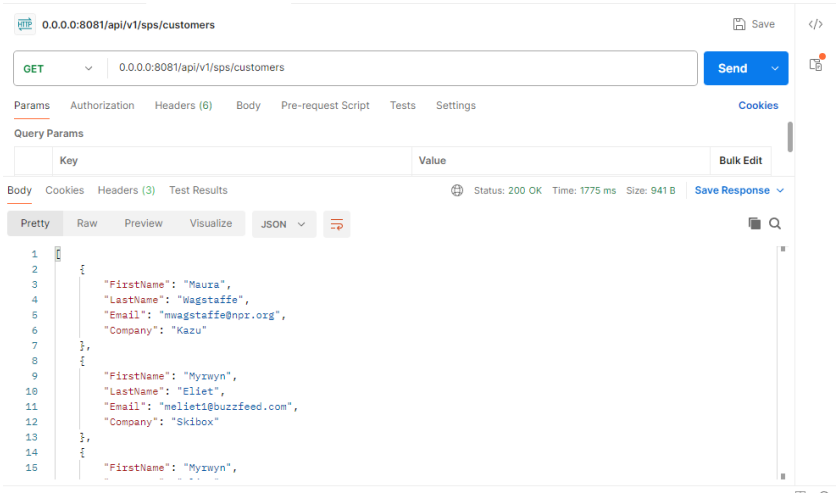

Figura 22. Petición GET desde Postman hacia el endpoint implementado con Mule

2.3 Despliegue de la aplicación en CloudHub

Antes de desplegar la solución previamente creada, debemos hacer una configuración para ocultar los valores de propiedades sensibles en la plataforma de despliegue. Para ello, abriremos el archivo *mule-artifact.json* y agregaremos *"secureProperties": ["secure.key"]*. Con esto, ocultaremos el valor de *secure.key* de forma visual dentro de la plataforma de despliegue.

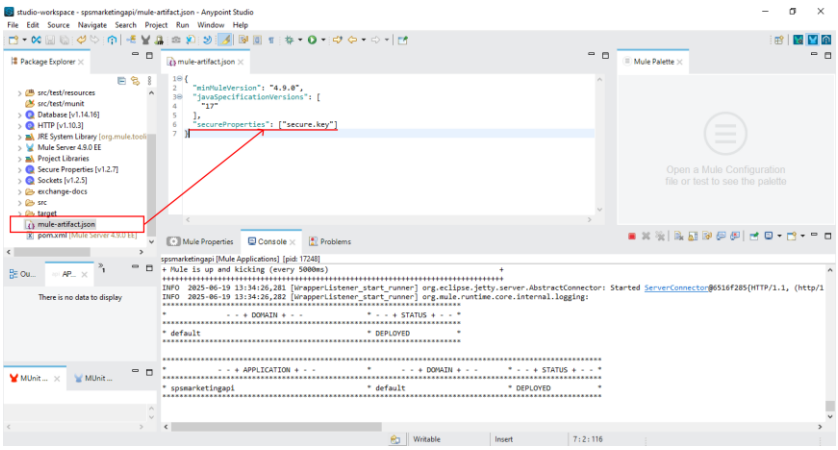

Figura 23. Configuración para ocultar propiedad segura secure.key

Una vez realizada la configuración previa, es momento de desplegar utilizando el servicio de *Anypoint CloudHub*. Para ello, daremos clic derecho en el proyecto y después seleccionaremos *Anypoint Platform* > *Deploy to CloudHub*.

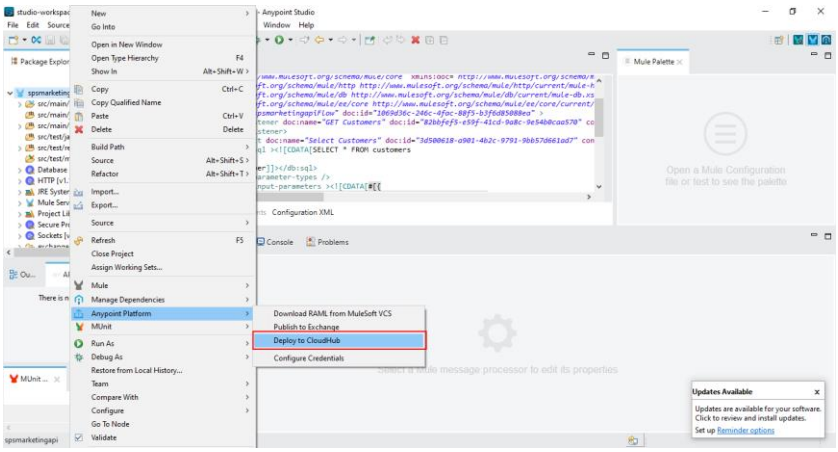

Figura 24. Opción en el menú para abrir la ventana de despliegue

Cuando realicemos el paso previo, tendremos que iniciar sesión con nuestra cuenta de *CloudHub*, para después pasar a una ventana de despliegue de la aplicación, donde se mostrarán dos opciones: *Design* y *Sandbox*. Seleccionaremos la segunda y esto nos llevará a la ventana de configuración de despliegue, donde dejaremos todas las configuraciones por defecto, a excepción del nombre, donde buscaremos uno válido y asociado con nuestro sistema; en este caso, *spsmarketingapi*.

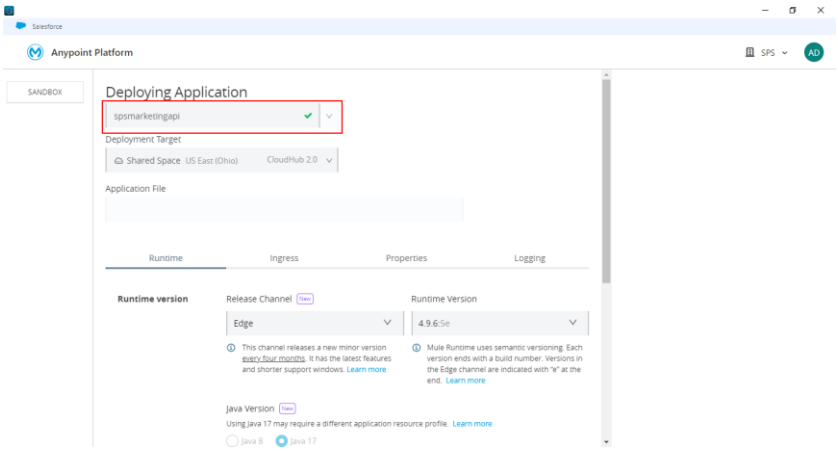

Figura 25. Ventana de configuración de despliegue, con el campo de nombre resaltado

Seguido de esto, pasaremos a la pestaña *properties* y definiremos *env* con el valor *dev* y *secure.key* con el valor *password* o cualquiera que haya sido el asignado en pasos previos. Finalmente, daremos clic en *Deploy application* y esperaremos que el proceso se complete.

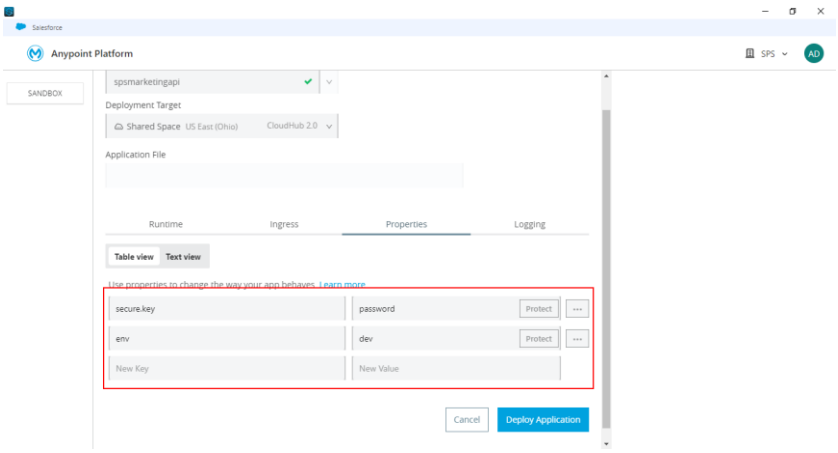

Figura 26. Configuración de properties previo al despliegue

Esto desplegará nuestra aplicación en *CloudHub*, que podremos administrar desde *Anypoint platform*, en el apartado de *Runtime Manager* > *Sandbox*. Ahí, podremos consultar el *Public Endpoint*, que además de servir para probar la aplicación, será de utilidad en configuraciones posteriores. En este ejemplo, este es *https://spsmarketingapi-ar49hp.5sc6y6-4.usa-e2.cloudhub.io/*.

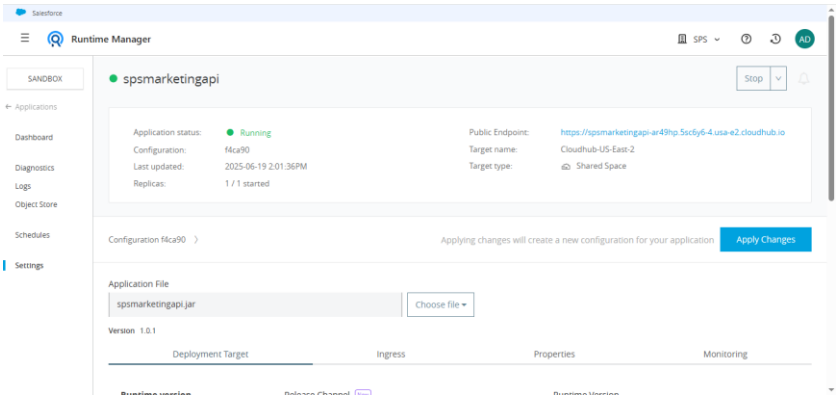

Figura 27. Ventana de Runtime Manager mostrando el despliegue de la aplicación

2.4 Especificación de la API

Con el despliegue de la API, debemos especificar cómo es que se comporta, para que los empleados del área de marketing sepan cómo utilizarla y sacarle el mayor provecho. Así, crearemos la especificación de la API con el uso de *Design Center* en *Anypoint Platform*.

Como primer paso, deberemos entrar a *Anypoint Platform* y seleccionar la opción *Design Center*, que abrirá el panel de gestión de especificaciones. Ahí, deberemos dar clic en el botón *Create* y seleccionar la opción *New API Specification*.

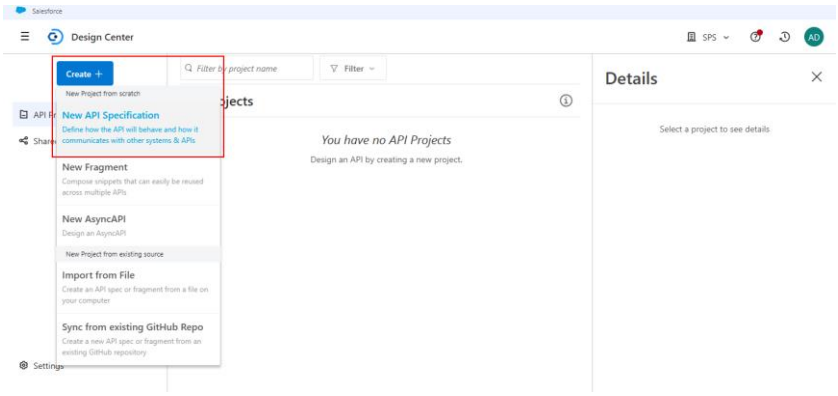

Figura 28. Panel de gestión de especificaciones de Anypoint Platform

Esto abrirá una nueva ventana de creación de especificación, donde en el campo *Project Name* deberemos colocar un nombre descriptivo para la especificación; en este caso *SPS marketing API Spec*. Además, deberemos seleccionar *Guide me trough it* y dar clic en el botón *Create API*, lo que nos abrirá una nueva ventana de configuración de la especificación.

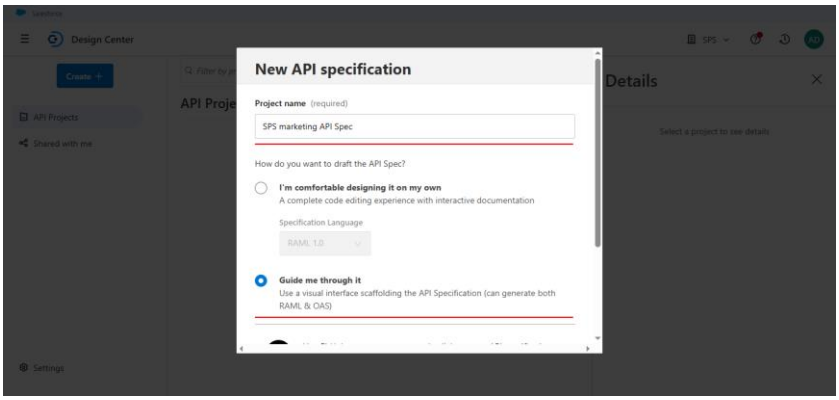

Figura 29. Ventana de creación de especificación

En esta nueva ventana de configuración, llenaremos los campos *Title*, *Version*, *Protocols*, *Media Type* y *Base URI*. Esto, con los valores que muestra la siguiente imagen, o adaptándolos a las necesidades del proyecto; es importante destacar el valor de *Base URI*, pues hace referencia al despliegue de nuestra aplicación realizado previamente.

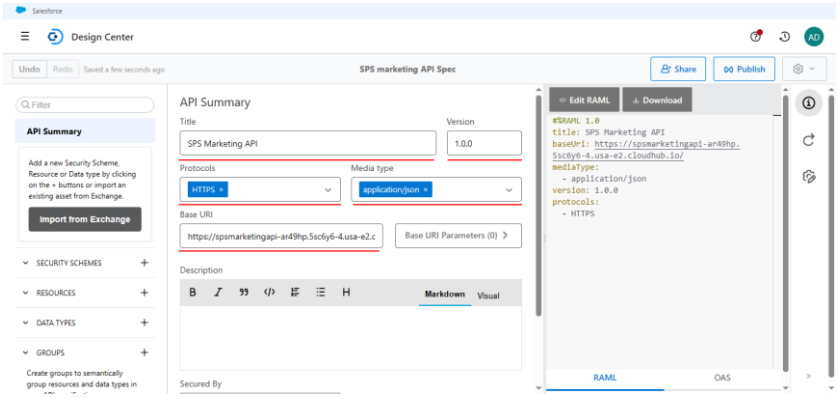

Figura 30. Configuraciones generales de la especificación de la API

Seguido de esto, configuraremos un nuevo tipo de dato desde la barra lateral izquierda, en la sección *Data Types*. Asignaremos el *Name* con el valor *Client* (o *Customer*) y el *Type* con *Object*. Esto permitirá registrar las propiedades, donde especificaremos *FirstName, LastName, Email* y *Company*, todas ellas de tipo *string*.

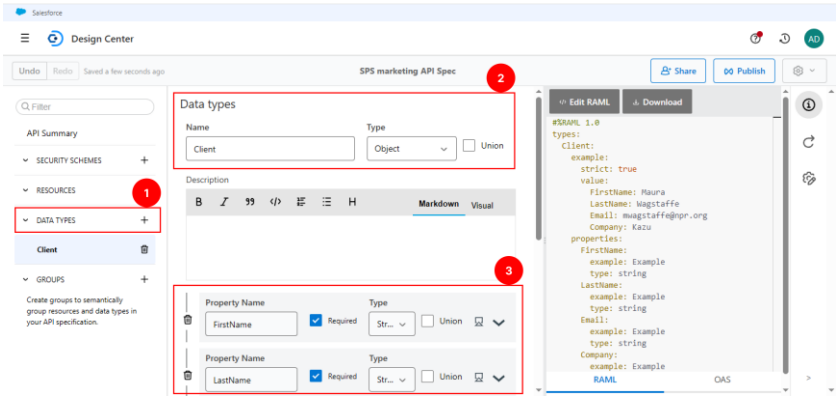

Figura 31. Configuración del tipo de dato Client en la especificación de la API

De forma similar, configuraremos un nuevo recurso (o *endpoint*). Para ello, en la sección de *Resources*, crearemos un nuevo recurso con valor /api/v1/sps/customers de *Resource path*, *GET* como método HTTP y *Get customers list* de *Name*. Todo esto en el apartado *Summary* de la configuración del recurso.

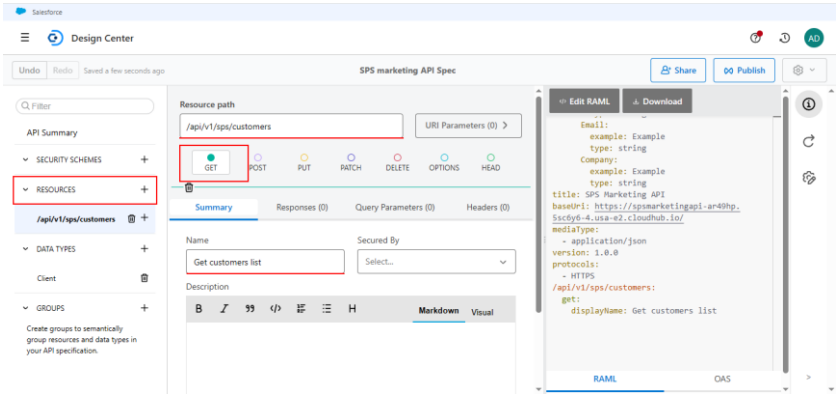

Figura 32. Configuración general del endpoint de customers

Seguido de esto, pasaremos al apartado *Responses* del recurso que estamos configurando, donde especificaremos un nuevo *Body* con *Status* 200, *Media Type* de tipo *application/json* y como *Type* un *Array* de *ítems Client* (el tipo que definimos previamente). Esto, pues es esta la única respuesta que hemos configurado hasta el momento en nuestro *endpoint*.

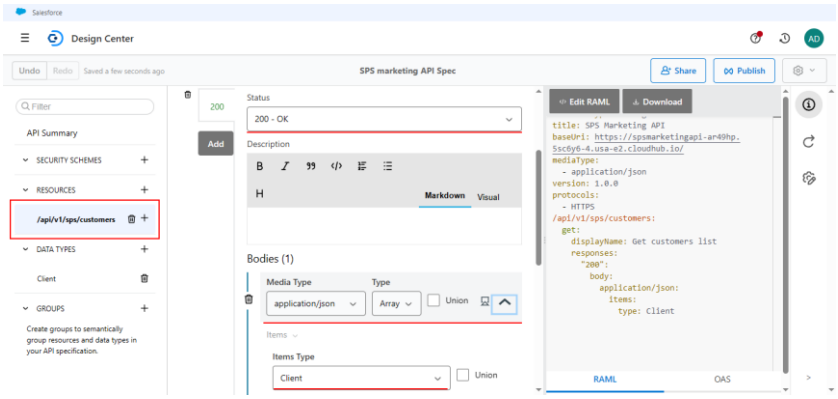

Figura 33. Configuración de respuestas del endpoint de customers

Finalmente, desde la interfaz donde hemos venido trabajando esta configuración, buscaremos el botón *Publish* en la parte superior derecha y daremos clic, lo que abrirá una ventana de configuración para publicar en *Exchange*. Ahí, deberemos especificar las *Asset version* y *API version*, que especificaremos como *1.0.0* en ambos casos y en el campo *Lifecycle State* indicaremos *Development*. Esto último, para indicar que la API se encuentra en la etapa de desarrollo. Con estas configuraciones, daremos clic en *Publish to Exchange* y nuestra especificación quedará cargada en la plataforma.

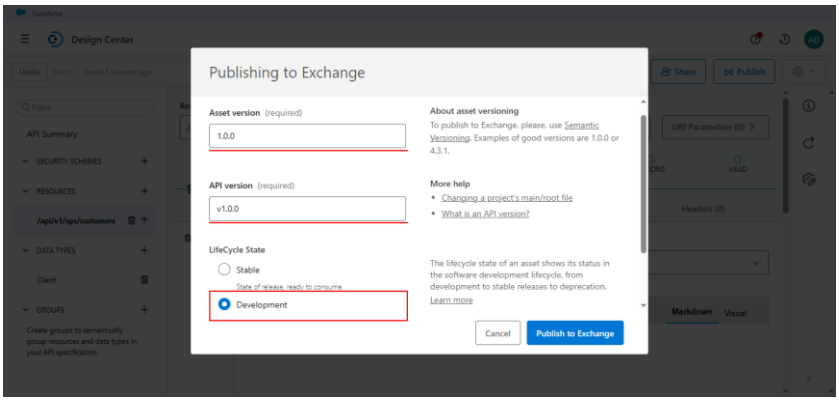

Figura 34. Ventana de configuración de publicación hacia Exchange

Realizado esto, nuestra especificación de API estará publicada en un link parecido al siguiente: https://anypoint.mulesoft.com/exchange/e44e5b60-6c33-4cc8-9258-0b9d7d117f5f/sps-marketing-api-spec/minor/1.0/pages/home/.

2.5 Mejoras en la solución

2.5.1 Configuración de seguridad con client-id Enforcement (API Management y discovery)

Hasta ahora, nuestra API se encuentra desplegada en *CloudHub*, sin embargo, esta es accesible por cualquier persona. Esto podría llegar a ser un problema para el equipo de marketing de SPS, pues usuarios malintencionados podrían saturar la API.

Con el fin de evitar esto, configuraremos *client-id Enforcement policy* en nuestra API desplegada. Pero antes de hacerlo, debemos configurar dos cosas: una nueva API dentro de *API Manager* y *API Autodiscovery*. La primera, para habilitar ciertas capacidades como la política que queremos implementar; la segunda, para conectar la API de *API Manager* con nuestra aplicación.

El primer paso será configurar una nueva instancia dentro de *API Manager*. Para ello, abriremos *Anypoint Platform* y seleccionaremos la opción *API Manager*. Esta abrirá una ventana donde encontraremos el botón *Add*. Al darle clic, desplegará un menú de opciones, del que seleccionaremos *Add new API*.

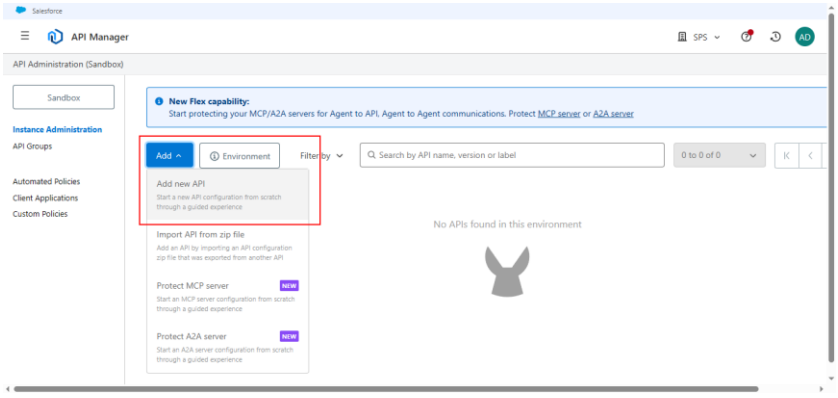

Figura 35. Ventana principal de API Manager

Esto nos mandará a una nueva serie de pasos para crear la API. En la primer ventana que aparece, seleccionaremos la opción *API Gateway*, y en los campos *API Proxy* y *Mule version*, ingresaremos los valores *Connect to existing application* y *Mule 4*. Con esta configuración, pasaremos al siguiente paso dando clic en *Next*.

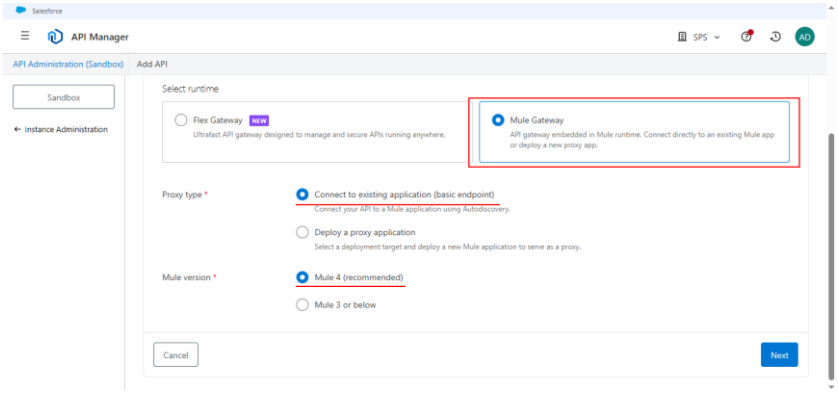

Figura 36. Ventana de configuración de ejecución de la instancia en API Manager

La siguiente ventana nos pedirá configurar la información general de la API. Ahí, seleccionaremos la opción *Create New API* y le asignaremos un *Name* adecuado según el proyecto; en este caso, *SPS Marketing API*. Finalmente, seleccionaremos *HTTP API* como *Asset Type* y daremos clic en *Next*.

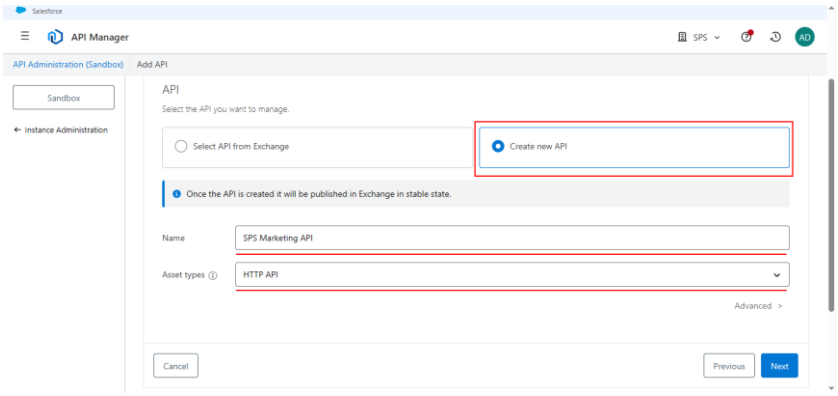

Figura 37. Ventana de configuración general de la API en API Manager

Las ventanas de configuración siguiente (*Downstream* y *Upstream*) las dejaremos con los valores por defecto y pasaremos directamente a la ventana de configuración final, que nos dará un resumen de la configuración y solamente deberemos dar clic en *Save*. Con esto, ya tendremos configurada la API dentro de *API Manager*.

Como siguiente paso, la configuración de *API Autodiscovery*. Para ello, necesitaremos el ID de la instancia previamente creada en *API Manager*, que podremos consultar accediendo directamente a la instancia.

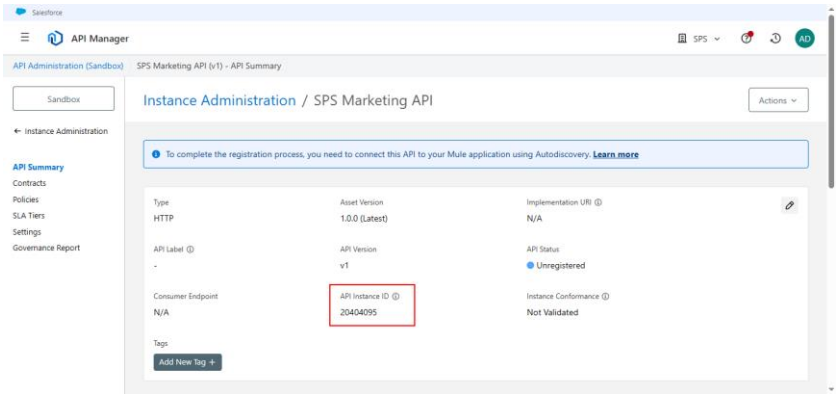

Figura 38. Ventana de información de instancia en API Manager, donde se muestra su API Instance ID

Con este valor a la mano, regresaremos a *Anypoint Studio* y, en nuestro proyecto, modificaremos los archivos *dev.properties* y *local.properties*, para agregar la propiedad *api.id* con el valor del ID de la instancia (*api.id=20404095*).

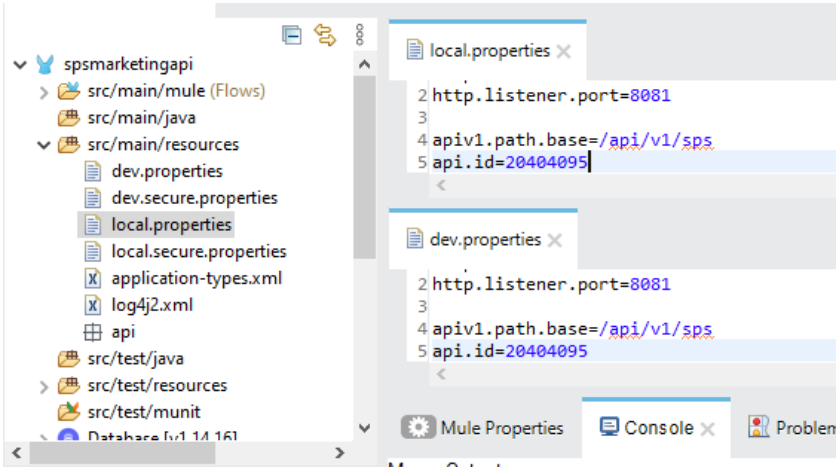

Figura 39. Configuración en archivos de propiedades para configurar API Autodiscovery

Seguido de esto, crearemos una nueva configuración global (gobal.xml). Buscaremos *API Autodiscovery*, seleccionaremos el resultado y en la ventana emergente, especificaremos los valores *${api.id}* y *spsmarketingapiFlow* (el flujo de nuestro endpoint), en los campos *API ID* y *Flow name*, respectivamente.

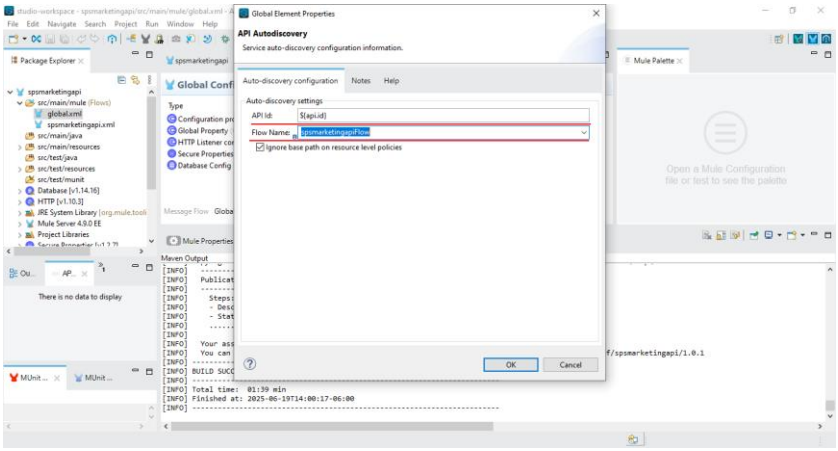

Figura 40. Ventana de configuración de API Autodiscovery

Con esta configuración realizada, regresaremos a *Anypoint Platform* > *Access Management* > *Business Groups* > Seleccionar la organización principal > *Environments* > *Sandbox*. Esto abrirá la ventana con los valores de *Client ID* y *Client Secret*, que deberemos recuperar y guardar, pues se usan en el paso posterior.

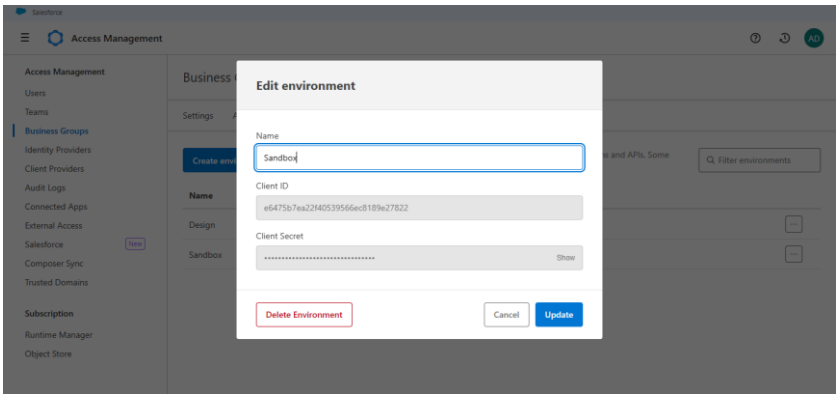

Figura 41. Ventana de ambiente con los valores de Client ID y Client secret

Como siguiente paso, regresaremos a *Anypoint Studio* y en el menú superior, en la opción *Window*, abriremos *Preferences*.

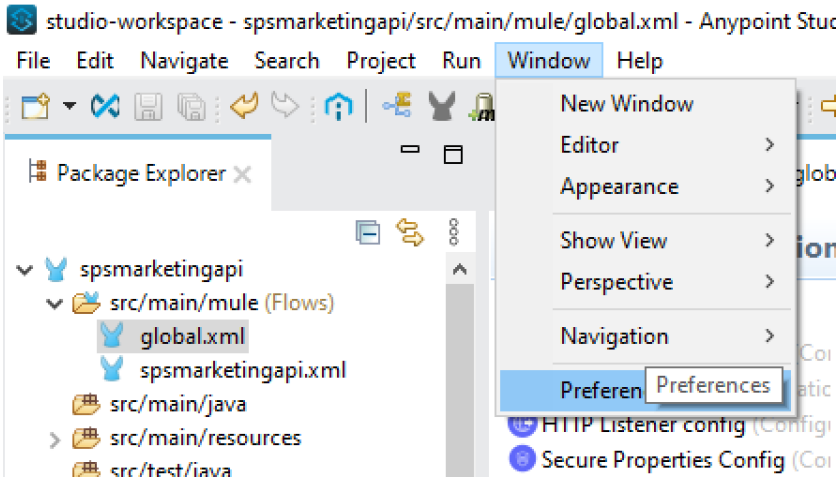

Figura 42. Opción preferences en Anypoint Studio

El paso previo abrirá una nueva ventana, donde iremos a la sección *Anypoint Studio > API Manager*. Con esto, se mostrará la configuración de *API Manager* y, en la sección *Environment Credentials*, deberemos pegar los dos valores que copiamos con anterioridad. Cuando los peguemos en los campos apropiados, daremos clic en *Validate*, verificaremos que no hay errores y finalizaremos con *Apply and Close*.

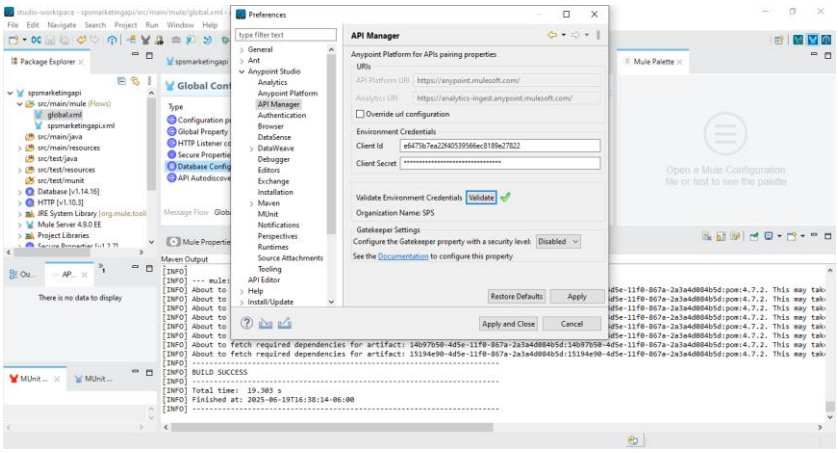

Figura 43. Ventana de configuración de API Manager desde Anypoint Studio

Después, deberemos modificar una vez más el archivo *mule-artifact.json*, agregando las propiedades *anypoint.platform.client_id* y *anypoint.platform.client_secret*.

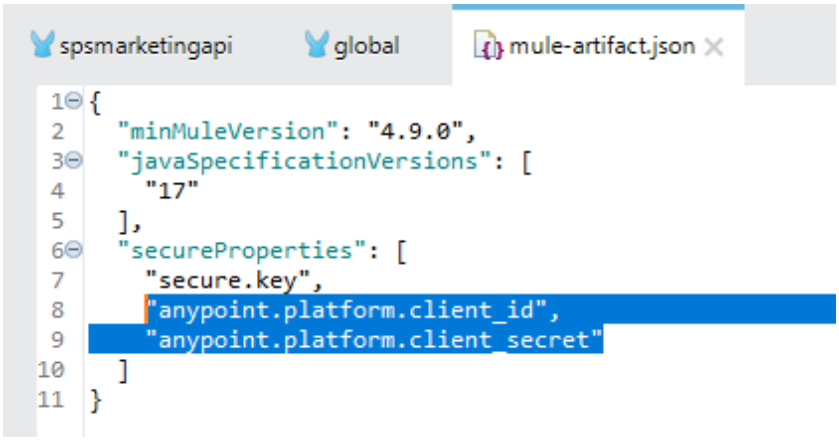

Figura 44. Configuración de mule-artifact.json para incluir propiedades asociadas con Anypoint

Con esta configuración, deberemos aplicar los cambios en nuestro despliegue (de nuevo desplegar a *CloudHub*) y especificar, antes de desplegar, estas propiedades previas tal como lo hicimos con *secure.key*. Así habremos conectado API Manager con nuestro proyecto. Esto lo podemos confirmar en *Anypoint Platform*, en *API Manager*, donde la instancia ahora aparecerá como *Active*.

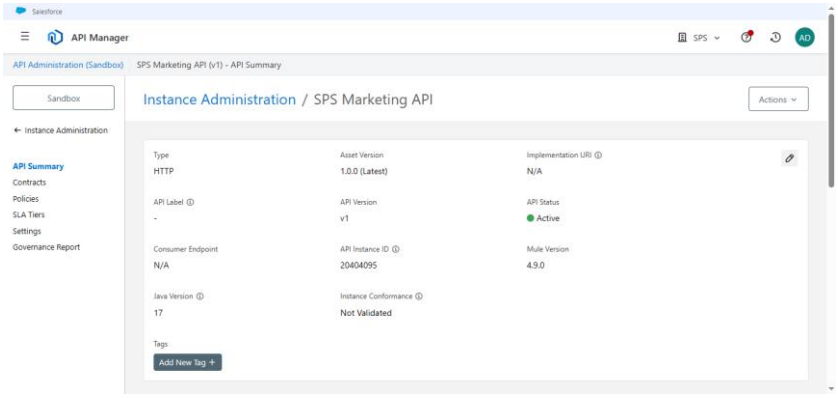

Figura 45. Detalles de la instancia en API Manager después de configurar Autodiscovery. Ahora aparece como activa

Como paso final, ya directamente relacionado con el establecimiento de la política, abriremos la instancia previamente creada dentro de *Anypoint Platform* y seleccionaremos la sección *Policies*, para luego dar clic en el botón *Add policy*.

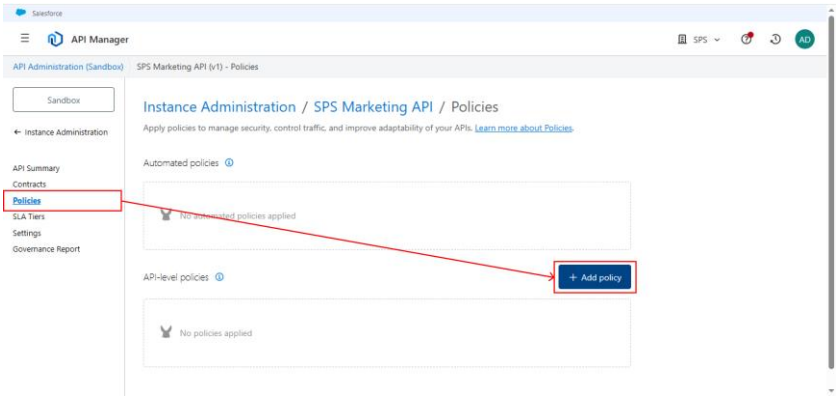

Figura 46. Ventana principal de configuración de políticas

En la ventana resultante, buscaremos *Client id*, seleccionaremos la opción proporcionada y daremos clic en *Next*.

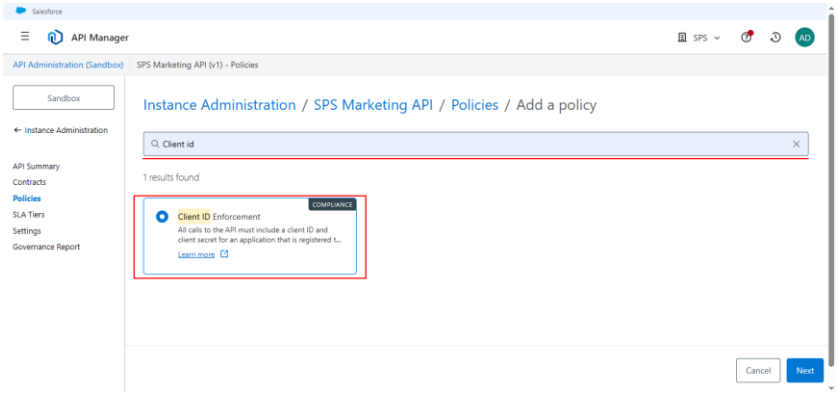

Figura 47. Ventana de selección de política

Después, seleccionaremos la opción *HTTP Basic Authentication header* para utilizar la autenticación por *usuario:contraseña* con el *header Authorization*. Y finalizaremos aplicando dicha política con el botón *Apply*.

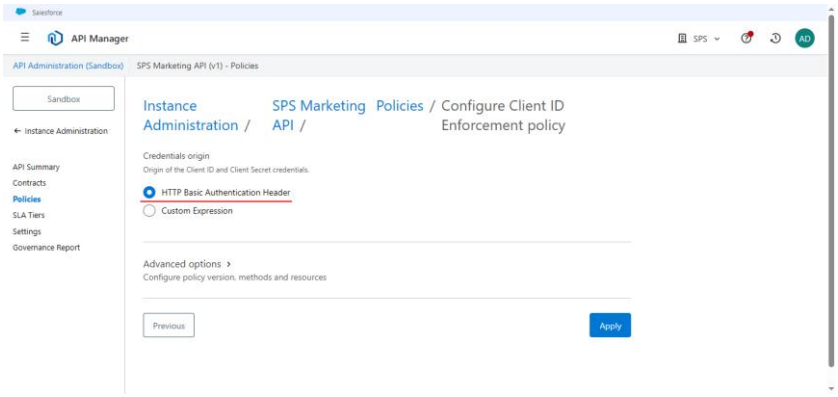

Figura 48. Ventana de configuración de Client ID Enforcement policy

Con esto, la configuración de autenticación ya estaría configurada y no habría que hacer ningún cambio más en la aplicación. Sin embargo, ahora el *endpoint* ya no es accesible sin permisos adecuados. Para otorgarlos, abriremos *Anypoint Platform* > *Exchange* y seleccionaremos *SPS Marketing API* (o el proyecto que configuramos desde *API Manager*).

Dentro del proyecto en *Exchange*, daremos clic en *Request Access*, lo que abrirá una nueva ventana de configuración.

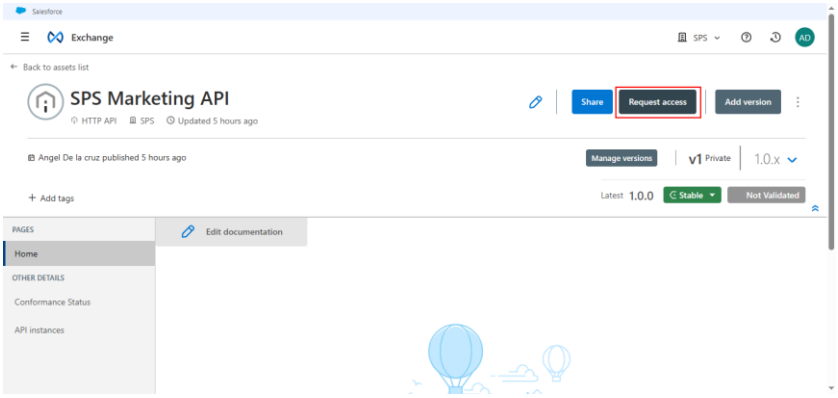

Figura 49. Ventana de detalles del proyecto en Exchange

En la nueva ventana seleccionaremos la única *API Instance* disponible, crearemos una nueva *Application* nombrada *SPS Marketing App* y la seleccionaremos. Con esta configuración, daremos clic en *Request Access*, lo que nos permitirá acceder a la API nuevamente, a través de *Basic Authentication* de HTTP.

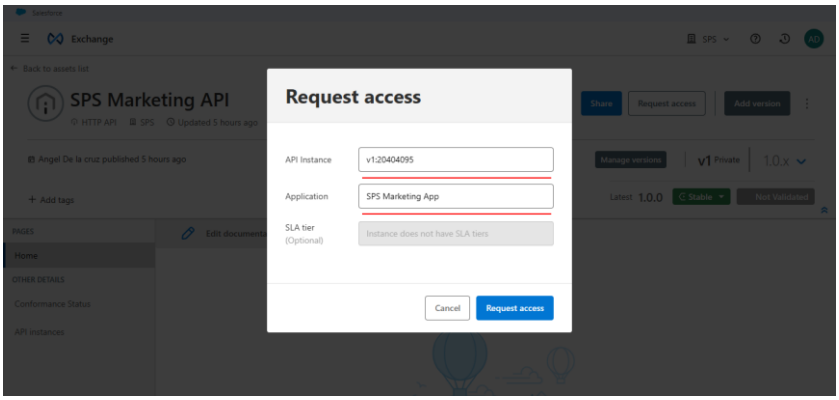

Figura 50. Ventana de configuración de acceso a la API

Con fines demostrativos y para probar la API, el acceso es:

***Client ID*:** 31886cf1cbeb486d9b4fe3229fe42133

***Client secret*:** C2D642035Ba84025bc074a497668FFc1

2.5.2 *Integración de paginación*

Recuperar todos los recursos de cierto tipo desde una base de datos es una tarea muy costosa. Utilizar paginación permite que las consultas sean más rápidas, controladas y escalables. Aunque en este caso la cantidad de clientes dentro de la base de datos no representan un problema, es valioso considerar este escenario y explorar este tipo de técnicas.

Con esto en mente, se contempla el uso de 2 parámetros: *page_size* y *page_number*. Para ello, el primer paso es modificar la consulta SQL asociada, para tomar en cuenta estos parámetros. En caso de no especificarse, se asume una página de tamaño 50 (máximo 50 clientes), retornando los recursos de la primera página.

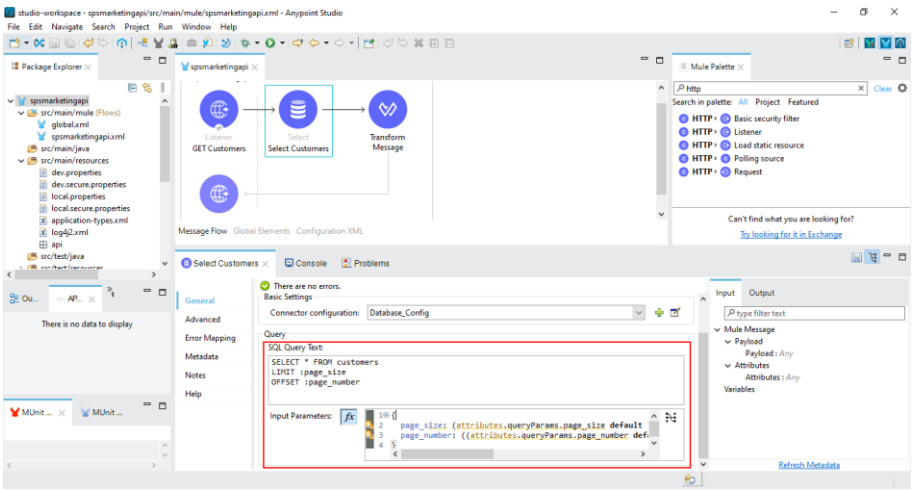

Figura 51. Nueva configuración de componente Select para contemplar parámetros de paginación y ordenamiento

Como se ve en la imagen, se hizo uso de parámetros de entrada, calculados a partir de los *queryParams*. Esta es una forma segura de inyectar la información de los parámetros, sin vulnerar la integridad de la base de datos por inyecciones SQL.

Ahora bien, con esta implementación, se agrega un problema. Tal como se encuentra el flujo del *endpoint*, no se están validando los parámetros, y los usuarios pueden mandar valores que no son numéricos, o que lo son, pero negativos. Esto no debería permitirse y, como está la solución, falla. El siguiente paso es validar esta información y controlar este flujo de información que llega a la API.

2.5.3 Monitoreo básico

Como primer paso hacia un mejor control de la información que llega a la API, crearemos un elemento *Logger* que imprima mensajes *INFO* sobre las peticiones que llegan a la API y los valores de *query parameters* que traen.

Para lograr esto anterior, abriremos el flujo de nuestra API (el archivo spsmarketingapi.xml) y agregaremos un elemento *Logger* justamente después del *Listener (HTTP)*. Una vez agregado, lo configuraremos con el valor *Requests Logger* en *Display Name* y un mensaje descriptivo que muestre los valores de *page_number* y *page_size* en el campo *Message*.

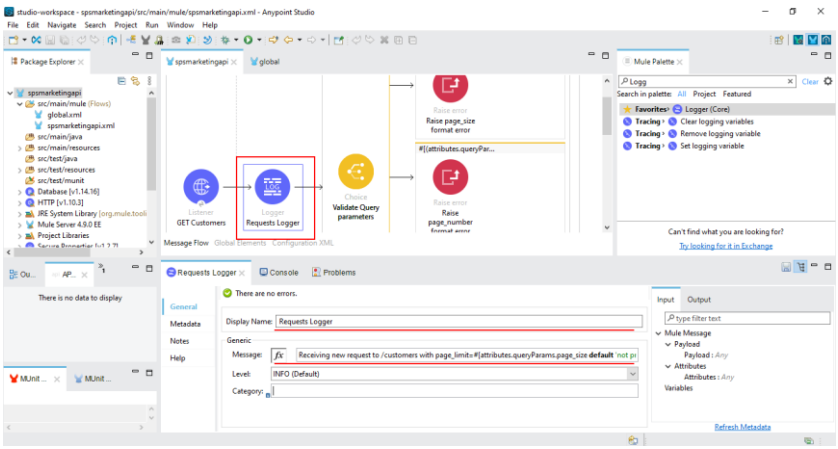

Figura 52. Configuración de Logger para peticiones

Con este simple cambio, tendremos un registro claro de las peticiones recibidas y los parámetros que están mandando los usuarios, lo cual puede ser de utilidad para detectar errores y visualizar el flujo de información.

2.5.4 Validación de parámetros

Pasando de lleno a la validación de parámetros, la idea es la siguiente: verificar que el parámetro *page_number* es un número entero positivo y que *page_size* es un

entero positivo menor o igual a 50. Además, retornar una respuesta con código de estado 400 en caso de que se infrinja alguna de estas dos condiciones.

Con esto en mente, el primer paso es agregar un elemento *Choice* con tres elementos internos: dos *Raise Error*, uno para cada caso en que alguno de los parámetros no viene en el formato adecuado y un tercero con nuestra lógica previa con los elementos *Select* y *Transform Message*. Este elemento *Choice*, siguiendo al *Logger* previamente creado.

El código de la condición del *Choice* para verificar el formato de *page_size* se muestra en la imagen debajo. Básicamente, se verifica que, si dicho parámetro no viene vacío, se revise si efectivamente es un número entre 0 y 50, incluyendo este último.

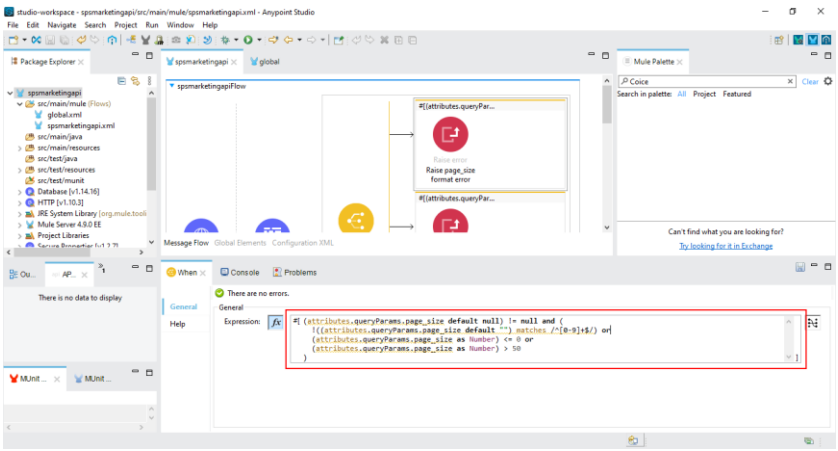

Figura 53. Configuración de la condición para la verificación de page_size en el Choice

En esta imagen previa, se puede visualizar el elemento *Raise Error* dentro de la condición de *Choice*, que se configura de forma muy sencilla. Establecemos las propiedades *Display Name*, *Type* y *Description*, con los valores *Raise page_size format error*, *CUSTOM:QUERY_PARAM_FORMAT* y *The query parameter page_size must be an integer between 0 and 50*, respectivamente.

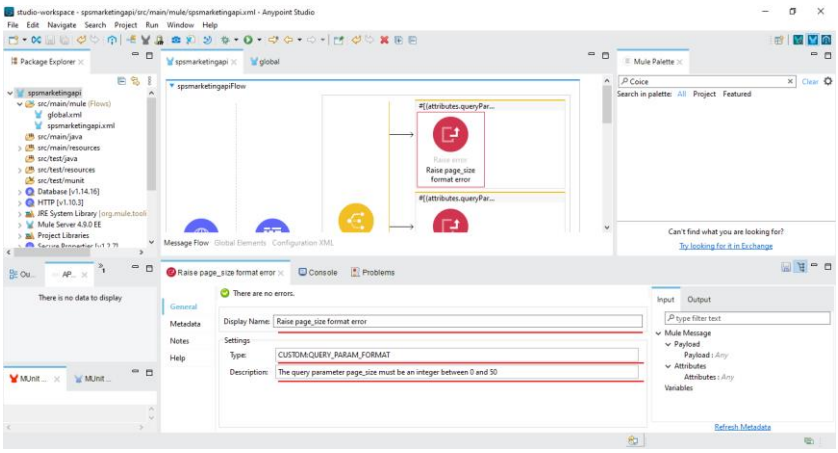

Figura 54. Configuración de elemento Raise Error para page_size

De forma análoga, deberemos configurar el *Raise Error* para *page_number*, al igual que la condición del *Choice*, pero adaptándolas al caso específico de este parámetro, tal como fue mencionado en un inicio.

En la tercera condición del *Choice* (default), deberemos colocar los elementos con los que ya contábamos, que eran el *Select* y *Transform Message*, sin realizar ninguna modificación a estos.

2.5.5 Manejo centralizado de errores y estados

Con la configuración anterior, los parámetros ya estarán validados en cuanto a formato. Sin embargo, si pasamos valores con formatos inadecuados, la API nos responderá con un formato de texto plano indicando el error. Lo mismo si, por ejemplo, la conexión con la base de datos falla. Esto, además de ser poco amigable con los usuarios finales, causa problemas de seguridad, al exponer los errores directamente, así como problemas de inconsistencia de formato, pues recordemos que nuestro *endpoint* de *customers* retorna JSON.

Para arreglar esto, centralizaremos el manejo de errores y le daremos formato a las respuestas. Para ello, en la sección *Error handling* de nuestro flujo, agregaremos un elemento *On error propagate*. A este primer elemento de manejo de errores, lo configuraremos indicando el valor *CUSTOM:QUERY_PARAM_FORMAT* en su propiedad *Type*. Recordemos que este tipo de error lo definimos al validar los formatos de los parámetros, por lo que este manejador de errores será de utilidad para retroalimentar a los usuarios sobre sus errores en las peticiones.

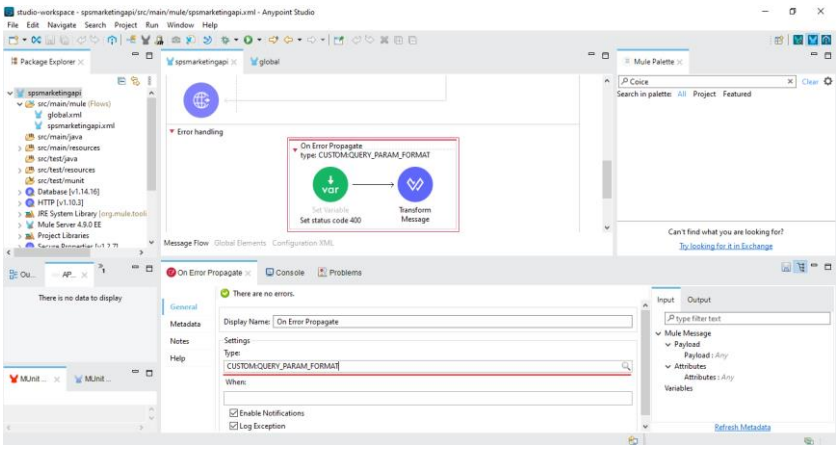

Figura 55. Configuración del elemento On Error Propagate encargado de manejar formatos erróneos en datos

Con este elemento configurado, deberemos agregar dos elementos dentro: *Set Variable* y *Transform Message*. El primero servirá para cambiar el código de respuesta HTTP, que será 400; el segundo, para establecer la respuesta JSON.

La configuración del elemento *Set Variable* queda como se muestra a continuación.

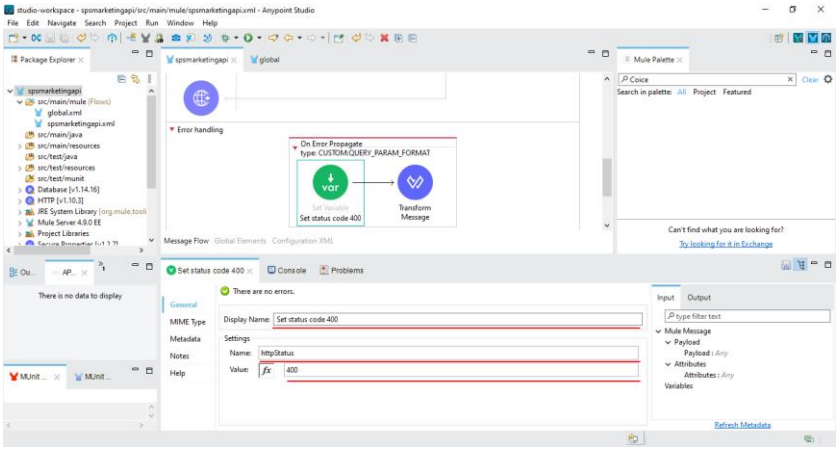

Figura 56. Configuración de Set Variable que establece un código de respuesta 400

Por su parte, la configuración del *Transform Message* accede a la descripción del error que establecimos previamente y la manda en un objeto JSON con una propiedad *message*. Esto se muestra en la imagen a continuación.

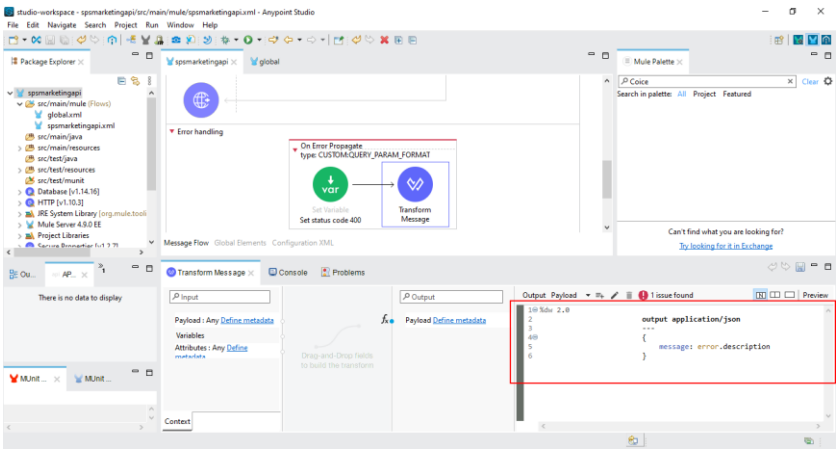

Figura 57. Configuración de Transform Message encargado de preparar las respuestas con código 400

En la misma sección de *Error handling*, agregaremos otro *On Error Propagate*, siguiendo una idea similar al anterior. Este no tendrá un *Type* específico, con el objetivo de manejar todos los errores inesperados. Además, dentro incluiremos un elemento *Logger* y uno *Transform Message*.

La configuración del Logger, deberá tener la propiedad *Message* con el valor *error.description* y *Level* con *ERROR*. La configuración es muy similar a la que ya realizamos con el otro *LOGGER*. La diferencia es que este tiene como objetivo registrar los errores no manejados para monitorear los fallos en el sistema.

Por otra parte, la configuración del *Transform Message* es la misma que la del anterior, pero cambiando el *message* del JSON por *Server out of service, try again later*.

Con estos dos elementos, manejamos una respuesta amigable para el usuario y dejamos registro del error real que ocurrió, para consultarlo después y poder solucionarlo.

Antes de finalizar y para que todo funcione como se espera, deberemos hacer unas últimas configuraciones en el *Listener (HTTP)*. En el apartado de *Responses* > *Error response* de su configuración, en *Body*, deberemos establecer *payload* como valor, para que pueda tomar los resultados de los *On Error Propagate* y mandarlos al usuario. Además, en ese mismo apartado, en el campo *Status code*, asignaremos *vars.httpStatus default 500* como valor, para que se recupere la variable definida en *On Error Propagate* y se cambie el estado de respuesta dependiendo el caso.

Como paso final, solamente deberemos desplegar nuevamente la solución en *CloudHub* como lo hemos venido realizando, así como actualizar la especificación de la API para incluir los detalles de parámetros, respuestas y autenticación.

3 CONCLUSIONES

Encontré muy interesante trabajar con este nuevo entorno de desarrollo. Hasta ahora, había desarrollado APIs REST desde cero, utilizando lenguajes de programación para ello. Sin embargo, este cambio de paradigma resultó muy interesante, eficiente y ágil. Aunque en un inicio fue complejo entender cada componente del entorno de desarrollo, su propósito y forma de uso, después de hacerlo, implementar *endpoints* y extender su funcionalidad se vuelve muy sencillo.

Así, puedo decir que:

1. **¿Me gustó esta herramienta?**

   Sí, me gustó la herramienta y además me pareció muy interesante y útil. Implementar APIs con estos flujos me parece un enfoque muy ágil, centrado en el negocio y no tanto en la tecnología, que a veces esta última es la que hace los desarrollos más lentos.

2. **¿Qué tan bien me familiaricé con Mulesoft?**

   Creo que me familiaricé realmente bien con la herramienta. Si bien las implementaciones me tomaron mucho tiempo, en su mayoría fue debido a la documentación paralela del proceso.

   El primer acercamiento que realicé, consultando los recursos recomendados e investigando otros por mi cuenta, me permitió tener muy clara la función de cada componente y el camino a seguir para implementar la API solicitada.

   A nivel de implementación, aunque hubo algunas dificultades, después de investigar un poco y con mis conocimientos previos sobre APIs REST, fue fácil llevar esos conceptos hacia este nuevo entorno de desarrollo.

3. **¿Qué dificultades tuve al usar la herramienta?**

   Hubo dos partes que se me complicaron entender: primero, el rol de cada herramienta dentro del desarrollo como un todo; y segundo, entender el flujo de información y cómo manipularlo ya en la implementación del *endpoint*.

   Fue un poco complejo comprender cada herramienta dentro del entorno de desarrollo, principalmente entender cuál era el rol de *API Management* y las especificaciones de APIs desde *Design Center*. Cómo esto se relacionaba con mi implementación en *Anypoint Studio* y qué rol tenía *Autodiscovery*.

Después, ya a la hora de implementar, me fue un poco difícil entender cómo se pasa la información entre los diversos elementos, como *Listener (HTTP)*, *Select* (base de datos), *On Error Propagate*, etc. Sin embargo, después de investigar y entender más a fondo la herramienta, considero que logré comprender este concepto a un buen nivel.

4. **¿Qué cosas logré dominar de la herramienta?**

   Los aspectos que más me costó entender, creo que fueron los que mejor logré dominar. El cómo conectar cada una de las herramientas para que funcionen como un todo y el manejar la información entre los componentes, al investigarlos tan a profundidad para entenderlos completamente, me permitió llevarlos a la práctica sin ningún problema y dominarlos.

   De igual modo, el manejo de la interfaz de *Anypoint Studio* y *Anypoint Platform*, con sus elementos y funciones, logré dominarlos a un muy buen nivel. Al menos a un nivel básico y para dar soluciones como la aquí presentada.

En conclusión, haber realizado esta práctica me ayudó a entender muchísimo mejor una de las herramientas utilizadas en el entorno de desarrollo de SPS y debo destacar que lo disfruté mucho. Si bien hay muchas cosas por aprender, mejores prácticas que adoptar y aspectos por mejorar, este primer acercamiento me deja muy satisfecho con el resultado logrado.