



# Práctica Mulesoft Trainee

Fecha: 25/09/2025

## DESCRIPCIÓN

Documentación del procedimiento realizado para llevar a cabo el desarrollo de la API Clientes para el equipo de Marketing en SPS.

## AUTOR

Sebastián Martín González Escalera

# ÍNDICE

<b>1.-Introducción.....</b>	<b>1</b>
<b>2.-Herramientas necesarias .....</b>	<b>2</b>
2.1-Anypoint Platform.....	2
2.2-Anypoint Studio .....	3
2.3-Cliente REST Postman .....	4
2.4-GitHub Platform .....	5
2.5-Sistema de control de versiones Git .....	5
<b>3.-Desarrollo de la API REST .....</b>	<b>6</b>
3.1-Creación del proyecto .....	6
3.2-Definición de las propiedades, elementos y seguridad del proyecto .....	12
3.3-Despliegue en CloudHub .....	25
3.4-Especificación de la API .....	28
<b>4.-Extras .....</b>	<b>33</b>
4.1-Creación de especificación de API en API Manager .....	33
4.2-Configuración de API Autodiscovery .....	40
4.3-Protección de la API .....	45
<b>5.-Retos y problemas presentados .....</b>	<b>48</b>
5.1-Documentación de MuleSoft .....	48
5.2-Consulta a la Base de datos .....	48
5.3-Bloopers.....	50
<b>6.-Conclusiones.....</b>	<b>51</b>

## **1.-Introducción**

Antes de hablar de mi procedimiento realizado, consideró que es necesario dar un poco de contexto respecto a mis conocimientos previos en el desarrollo de API's y algunas de las herramientas utilizadas en esta práctica. Normalmente en este tipo de documentos no es algo apropiado hablar en primera persona, sin embargo, debido a que en este caso dar a conocer más de mi persona y mi experiencia con esta práctica cobra más relevancia, hare una excepción.

Como mencione, ya cuento con experiencia en el desarrollo de API's muy básicas, particularmente con Java y Spring Boot, también con herramientas como Postman y Git, esto permitió que me sintiera familiarizado con algunos de los puntos a desarrollar en la práctica, siendo para mí el mayor reto el familiarizarme con el entorno de MuleSoft y Anypoint Studio.

En el presente se verá desglosado punto por punto el procedimiento que decidí llevar a cabo para realizar la práctica teniendo en consideración lo anterior, desde el establecimiento de tareas, como las herramientas que decidí utilizar, así como los obstáculos o retos que se me presentaron durante el desarrollo.

## 2.-Herramientas necesarias

Como prerequisite dicho por la misma documentación de MuleSoft y el documento proporcionado por SPS, era necesario contar con algunas herramientas indispensables, le di prioridad a descargar dicho Software y registrarme donde fuera necesario.

### 2.1-Anypoint Platform

Lo primero que realice y que es indispensable para poder proceder, fue registrarme en Anypoint Platform, una plataforma de Mulesoft de integración empresarial híbrida que permite a las empresas crear, implementar, proteger y administrar sus API e integraciones. Para registrarse en dicha plataforma solo es necesario un correo electrónico, llenar un pequeño formulario y verificar la cuenta por medio del email proporcionado.

MuleSoft announces solutions to build a future-ready foundation for AI. [Learn more](#)

MuleSoft from Salesforce Products Solutions Services Resources Developers Partners [Contact Us](#) 1-800-596-4880 [Login](#) [Free trial](#)

## Download Anypoint Code Builder, Studio, or Mule

**New** Anypoint Code Builder and Mule 4

- Design APIs across OAS and RAML specifications
- Test and validate API functionality with built-in mocking service
- Automatically implement API logic from existing specification
- Build integrations to connect any app, system, or data together
- Jumpstart integration development with the power of generative AI

Anypoint Studio 7 and Mule 4

Mule 4 standalone

Previous versions

### Get started for free today

Anypoint Studio and Mule

Latest

Windows

Sebastián Martín

González Escalera

dev.sebastian.gonzalez@gmail.com

\*Please, use a valid email. We'll send the download link to the provided address.

GOES970220HDFNSB08

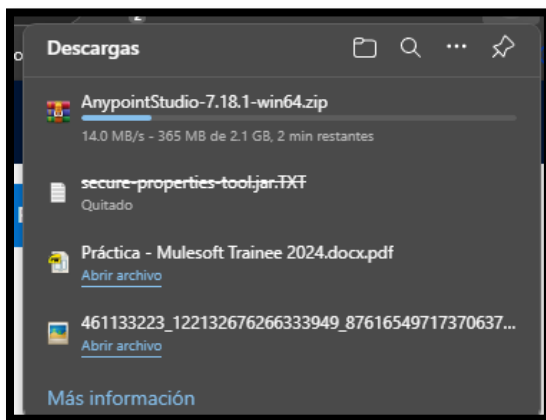
Backend Developer

5573371691

La plataforma cuenta con una cantidad muy grande de herramientas tanto para desarrollar API's como para poder hacer uso de ellas, por mencionar algunas contiene lo necesario para implementar pequeñas bases de datos en ellas, seguridad, test rápidos, logs, e incluso diseñarlas de manera más grafica comparado a lo convencional.

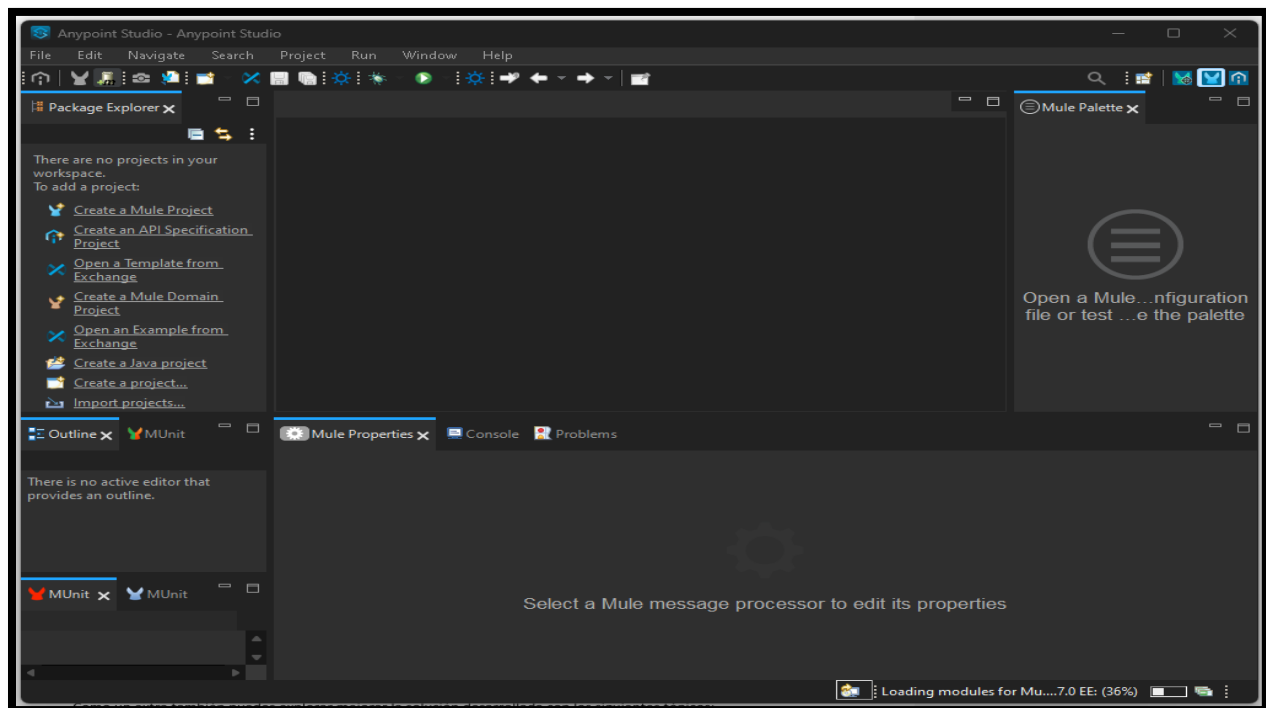
## 2.2-Anypoint Studio

Este es el IDE proporcionado por MuleSoft para poder desarrollar en conjunto con Anypoint Platform, se puede descargar en el mismo sitio web una vez completado el registro, cuenta con una interfaz gráfica similar a la de otros IDE como IntelliJ y está totalmente pensado para trabajarse con el ecosistema de MuleSoft.



Instalarlo es una tarea rápida y sencilla, pues con la configuración que viene por defecto es suficiente para comenzar a utilizarlo, sin embargo, más adelante será necesario hacer algunos ajustes pequeños para poder llevar a cabo la práctica.

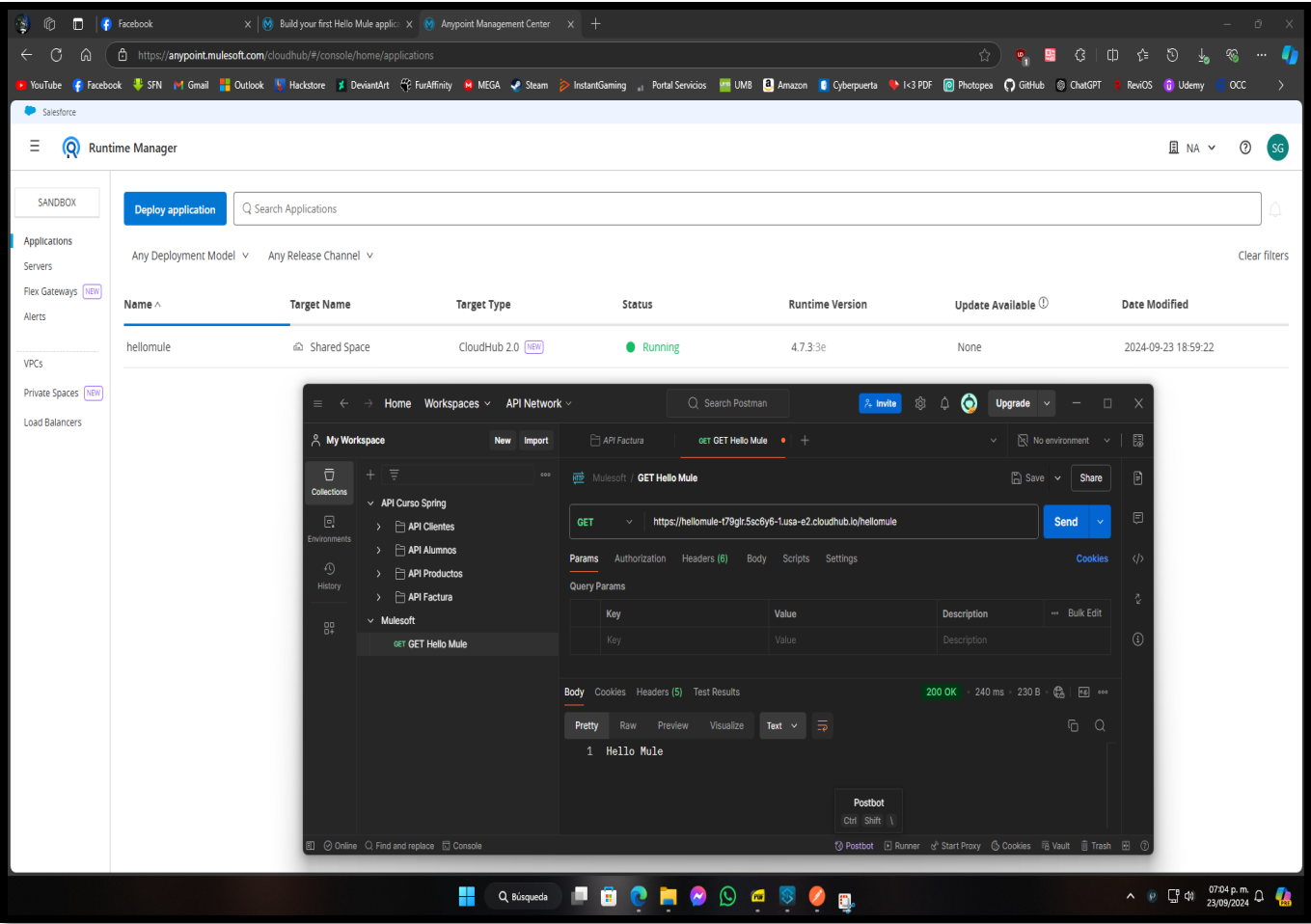
Un vistazo rápido nos confirma que tiene una interfaz similar a la de cualquier otro IDE, contando con una consola, un editor y un explorador de archivos, entre otras cosas.



### 2.3-Cliente REST Postman

Postman es una popular herramienta utilizada para probar APIs, permitiendo a los desarrolladores enviar peticiones a servicios web y ver respuestas, existen otras herramientas de software que permiten realizar esto, sin embargo, yo decidí utilizar Postman debido a que ya contaba con él en mi equipo, además de haberlo utilizado con anterioridad y ser el que se utiliza también en la documentación de MuleSoft.

Postman cuenta con diversas herramientas para hacer solicitudes http del tipo GET, POST, PUT, PATCH y DELETE de forma organizada, detallando la respuesta de la solicitud y considerando diversos formatos de respuesta como JSON y XML, así como posibles características como el pase de parámetros o la autenticación. A continuación, puede verse una pequeña prueba con la práctica ofrecida en la documentación de MuleSoft “Hello Mule”, haciendo uso de una solicitud GET.



## 2.4-GitHub Platform

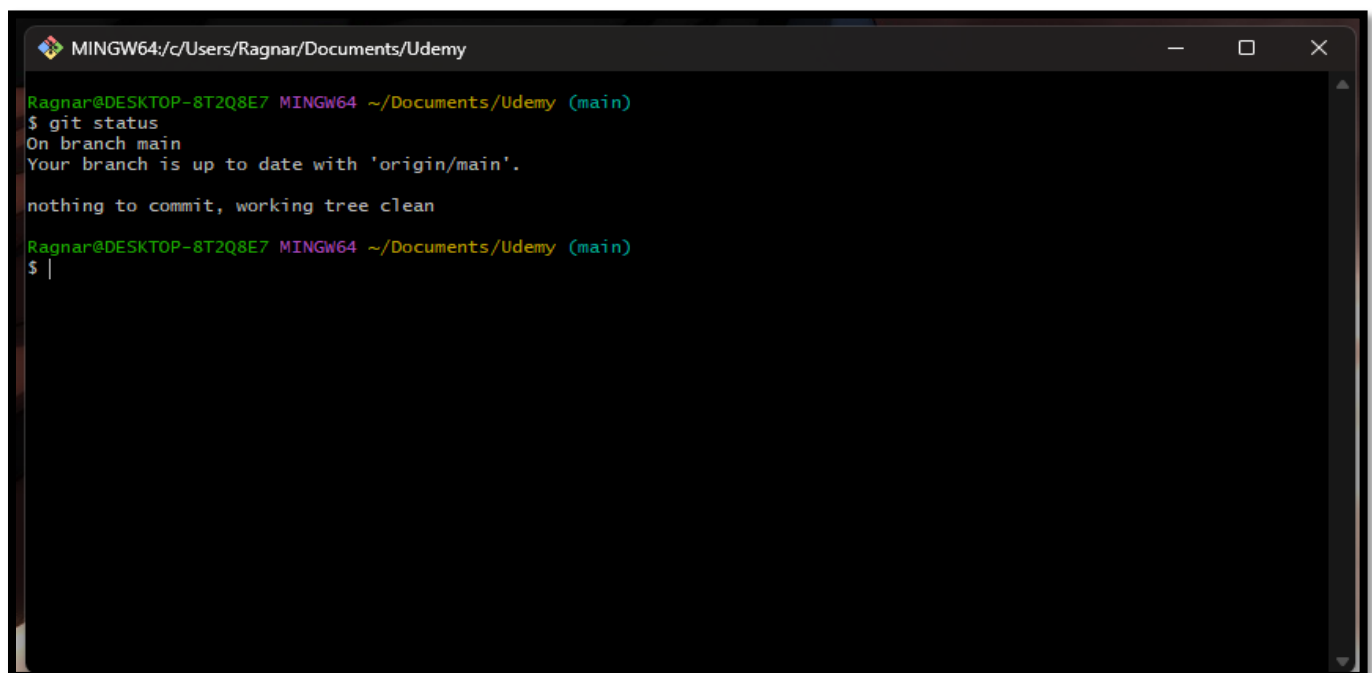
GitHub es una plataforma web que permite almacenar, compartir y trabajar en proyectos de código abierto, es por mucho la más utilizada de su tipo y será necesario tener una cuenta en ella para poder crear el repositorio de acceso público que alojará los entregables de la práctica, yo ya contaba con una cuenta en dicha plataforma.

## 2.5-Sistema de control de versiones Git

Git es un sistema de control de versiones distribuido que permite a los desarrolladores administrar los cambios en su código fuente, esta herramienta se utiliza de la mano de GitHub por lo general, aunque no es indispensable para realizar la práctica, si será útil, pues permite gestionar los repositorios de forma remota de una forma más completa y sencilla que hacerlo directamente desde GitHub.

Esta herramienta tampoco necesite instalarla pues ya contaba con ella en mi equipo, funciona por medio de comandos en la terminal, aunque existen herramientas para trabajar con ella con una interfaz gráfica.

Aquí se puede ver un ejemplo de su uso, consultando el estado de un repositorio con el comando “git status”.

A screenshot of a terminal window titled "MINGW64/c/Users/Ragnar/Documents/Udemy". The prompt is "Ragnar@DESKTOP-8T2Q8E7 MINGW64 ~/Documents/Udemy (main)". The user enters "\$ git status". The output is: "On branch main", "Your branch is up to date with 'origin/main'.", and "nothing to commit, working tree clean". The prompt then changes to "Ragnar@DESKTOP-8T2Q8E7 MINGW64 ~/Documents/Udemy (main)" followed by "\$ |".

```
MINGW64/c/Users/Ragnar/Documents/Udemy
Ragnar@DESKTOP-8T2Q8E7 MINGW64 ~/Documents/Udemy (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

Ragnar@DESKTOP-8T2Q8E7 MINGW64 ~/Documents/Udemy (main)
$ |
```

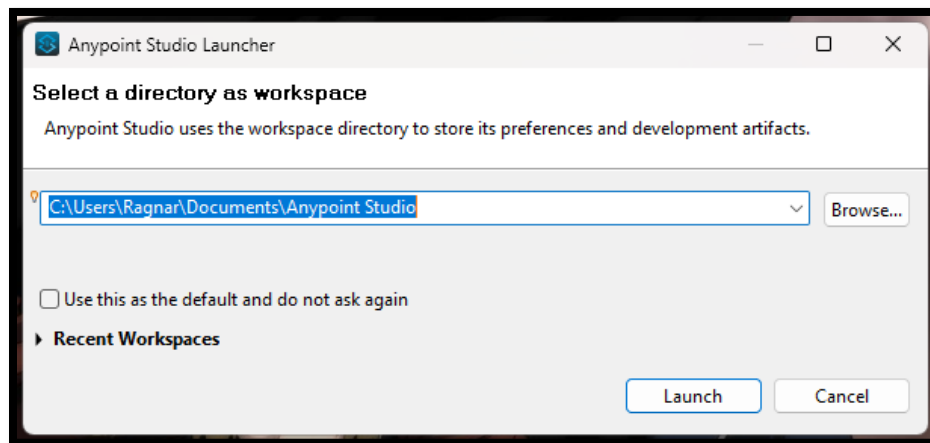
### 3.-Desarrollo de la API REST

Comenzando con el desarrollo de la práctica, en el documento proporcionado por SPS se solicita tener la información de los clientes para el área de Marketing, esto con la intención de realizar campañas personalizadas para los clientes, dicha información debe estar almacenada en una base de datos.

Para este punto ya estoy preparado con las herramientas previamente mencionadas y conocimientos esenciales adquiridos al consultar la documentación y otras fuentes de información.

#### 3.1-Creación del proyecto

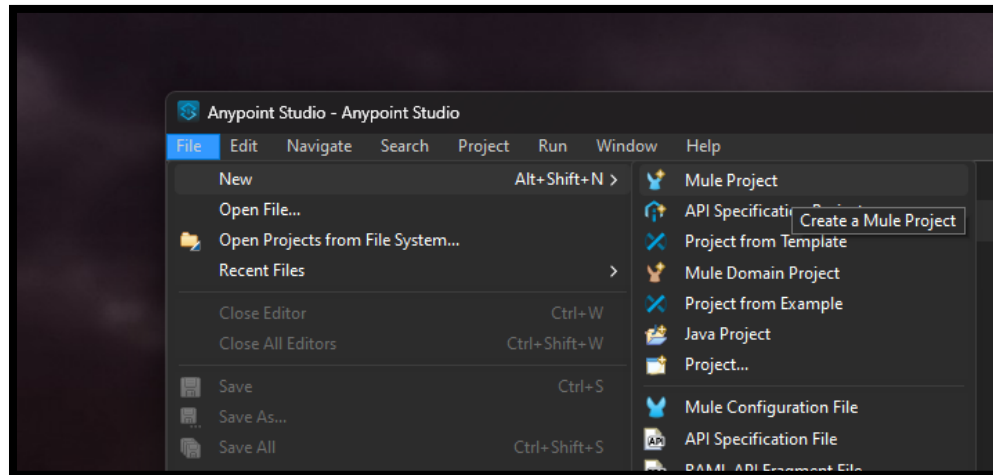
Lo primero que se solicita es que la información de los clientes quede expuesta en el endpoint *“/api/v1/sps/customers”*, esto puede realizarse desde Anypoint Studio, comencé creando un nuevo proyecto, al iniciar el IDE este solicitara que se defina un espacio de trabajo, es decir, un directorio donde el programa guardara los proyectos y sus archivos de configuración, en mi caso escogí el siguiente:



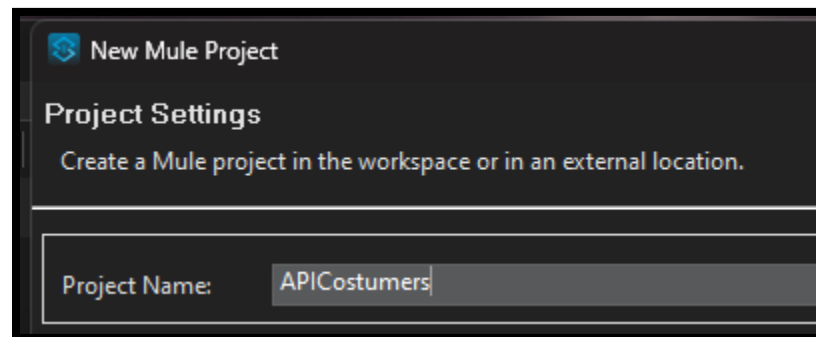
Una vez seleccionado un directorio, se da clic en “Launch” para continua, una vez aparezca la ventana principal del IDE podremos continuar con la creación del proyecto nuevo.



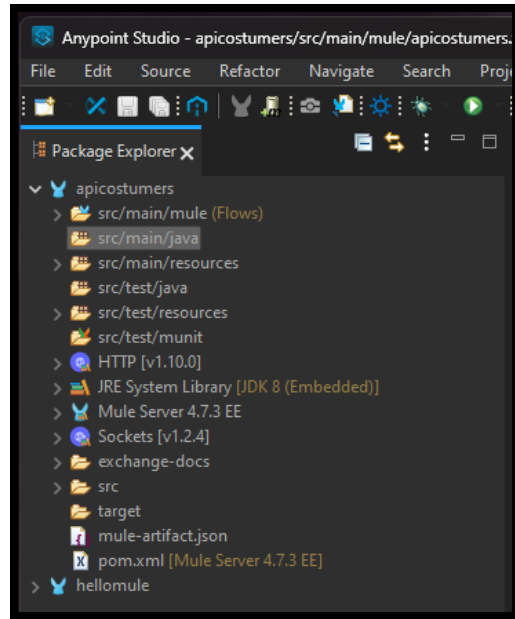
Para crear un nuevo proyecto, se debe dar en “File” ubicado en la parte superior de la ventana (la barra de herramientas) y a continuación, posicionarse en “New”, después, dar clic en “Mule Project”:



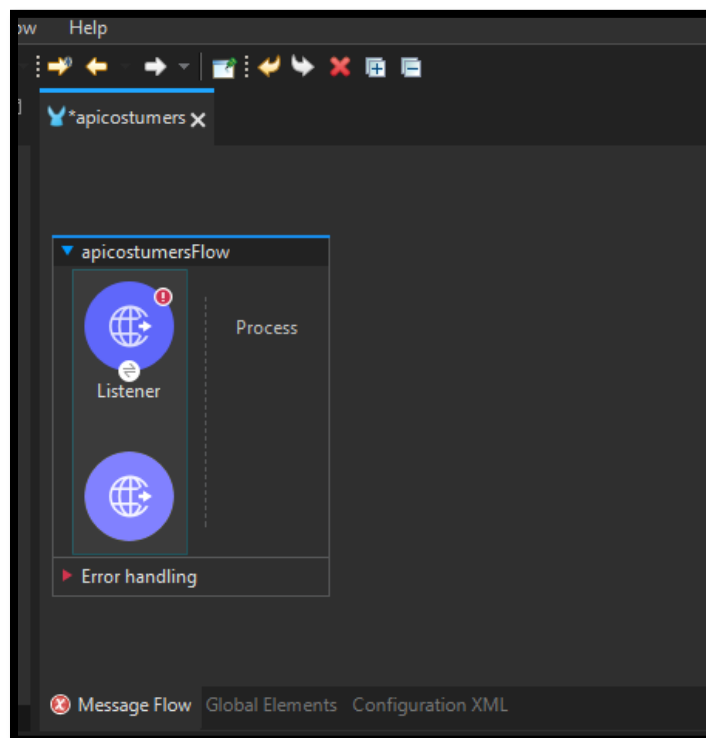
En la siguiente ventana, debemos dejar todos los ajustes por defecto, asignarle un nombre a nuestro proyecto y, por último, dar clic en “Finish”, en mi caso le puse el siguiente nombre:



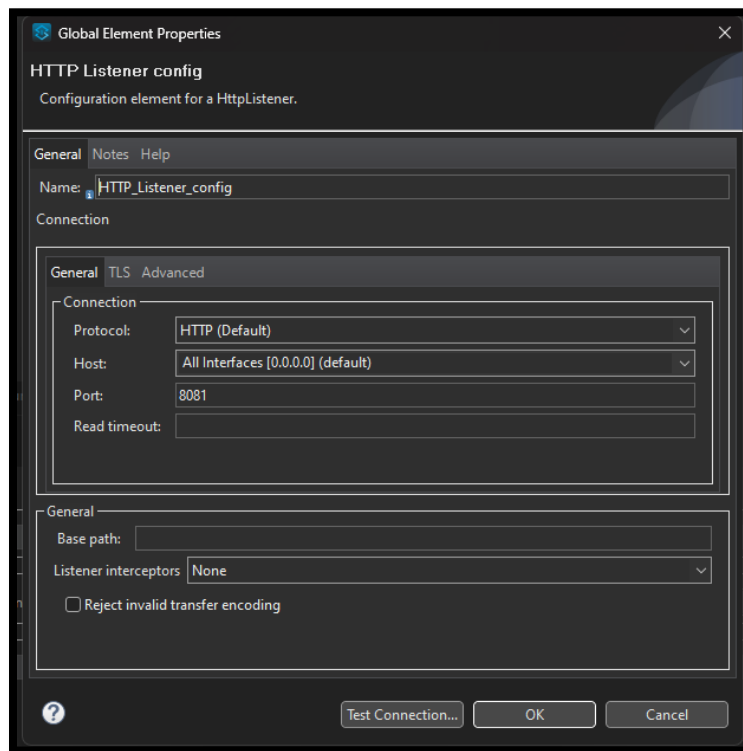
Una vez creado el proyecto, nos aparecerá en la barra lateral izquierda, los proyectos de Mule por lo que pude observar tienen una arquitectura muy similar a la de un Proyecto Java con Maven, mezclado con algunas dependencias del propio Mule y de Spring.



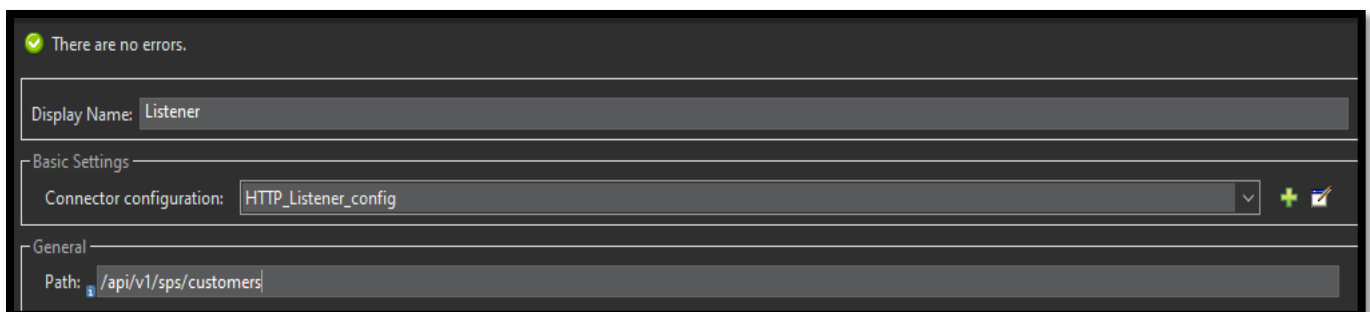
Entre estas dependencias o módulos, se encuentra un módulo http, a este ocupamos añadirle un módulo “listener”, este será el que permitirá recibir solicitudes del tipo GET, para añadirlo solo se debe ir a la paleta de módulos ubicada en la parte lateral derecha y arrastrar el módulo al editor, de manera que quede así:



Ahora, falta configurar algunas propiedades del listener, para ello se debe hacer clic en él y después, en la parte inferior del editor, se verá una sección con sus propiedades, las propiedades que es necesario definir son la dirección del Host y el Puerto en el que se ejecutara el servicio, debemos añadir estas propiedades con el botón “+” de color verde y definir las en la siguiente ventana:

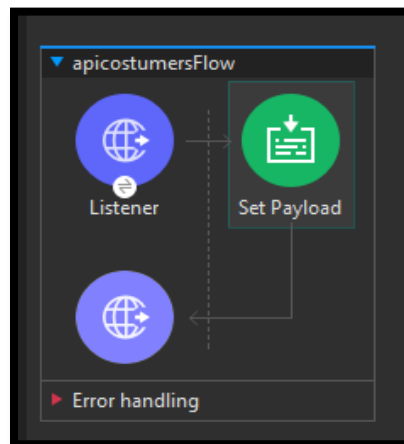


Además, también es necesario modificar la propiedad “path” de la sección, en ella vamos a definir el endpoint de nuestro servicio o recurso, como se solicitó, el endpoint debe ser el siguiente:

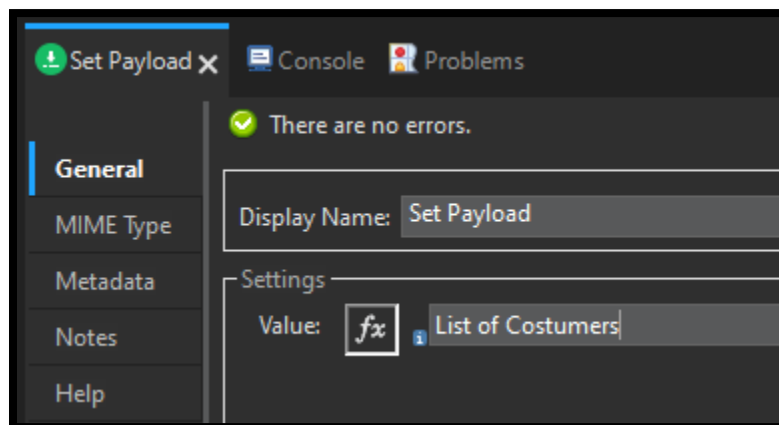


Cabe mencionar que este endpoint es general, lo cual es funcional para una práctica como esta que ofrece acceso a un único servicio o recurso, sin embargo, para una API más realista sería más correcto definir una URL general y una URI por recurso partiendo de esta, en cuanto a la definición de este endpoint cumple con buenas prácticas al mostrar que es una API, su versión, la organización y el recurso sin utilizar verbos, se deben guardar los cambios hechos en el listener.

Antes de proseguir con el siguiente punto, toca probar la API de manera local, para ello es posible apoyarse del módulo “Set Payload” este permitirá generar un valor de respuesta para las solicitudes que lleguen al listener, es posible agregarlo igual que el listener desde la paleta de módulos al editor, debe agregarse al flujo de “proceso” del listener de forma que quede así:



Además, en las propiedades del Set Payload, se debe modificar la propiedad “value”, aquí debe ir la respuesta que arrojará el listener, en mi caso le pondré “List of costumers”:



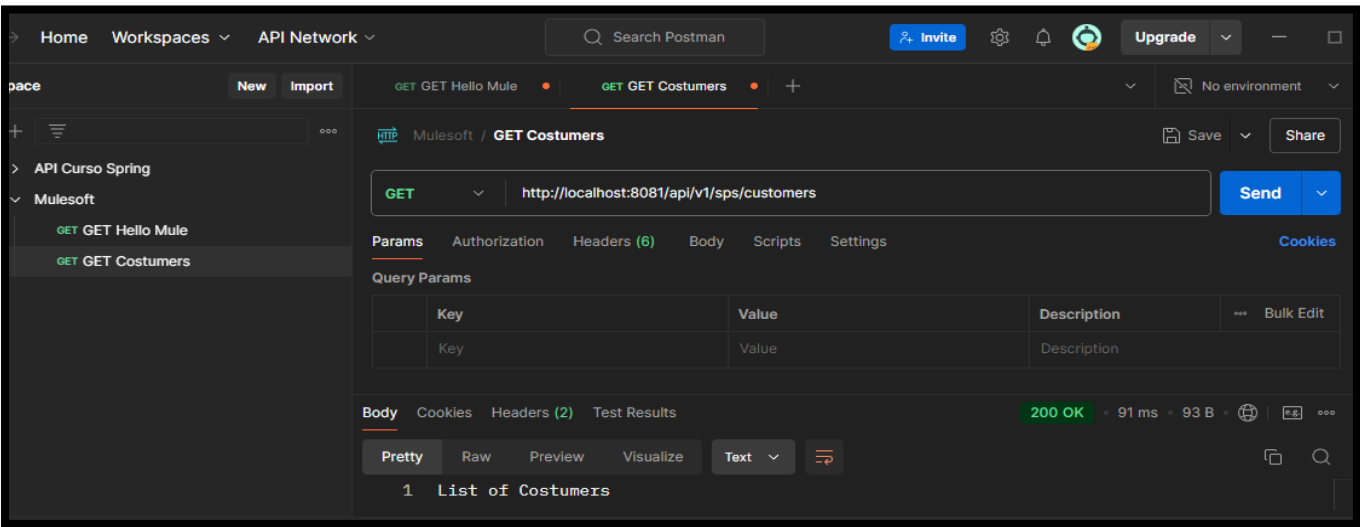
Una vez hecho este cambio, ya es posible probar la API usando Postman, toca ejecutar nuestra aplicación dando clic al botón “Run” de la barra de herramientas o dando clic derecho en el editor y luego a la misma opción en el menú contextual.

Es necesario esperar unos 30 segundos para que el servicio se ejecute, si todo está correcto, en la consola debe aparecer el status de la aplicación como “DEPLOYED”:

```
apicostumers [Mule Applications] [pid: 5024]
INFO 2024-09-24 23:11:47,159 [WrapperListener_start_runner] org.eclipse.jetty.server.AbstractConnector
INFO 2024-09-24 23:11:47,161 [WrapperListener_start_runner] org.mule.runtime.core.internal.logging:
*****
* - - + DOMAIN + - - * - - + STATUS + - - *
*****
* default * DEPLOYED *
*****

*****
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
*****
* apicostumers * default * DEPLOYED *
*****
```

Si es correcto, toca ir a y crear una carpeta de solicitudes solo para este proyecto, por cuestión de orden, o bien, crear una única solicitud GET con un nombre que la diferencie, después, se manda la solicitud a la dirección del localhost con el puerto 8081 y el endpoint definido, de esta forma:

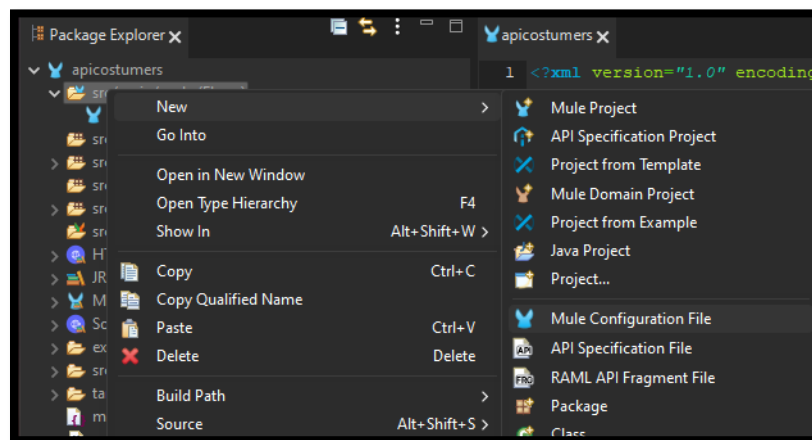


Ya verificado el funcionamiento local, pasamos al siguiente punto.

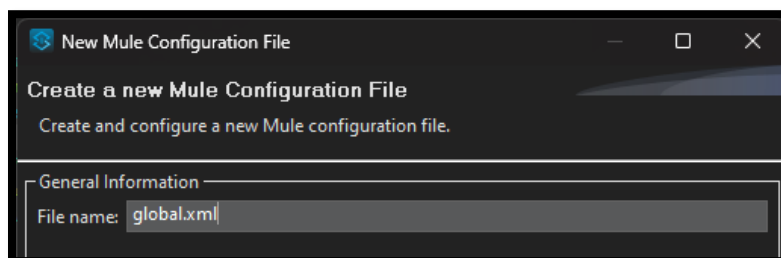
### 3.2-Definición de las propiedades, elementos y seguridad del proyecto

Como parte de las buenas prácticas es a menudo necesario tener más de un archivo de configuración en el proyecto, por defecto, generalmente los proyectos manejan un único archivo de configuración global, lo cual puede llegar a ser problemático a medida que crecen las implementaciones, es por ello que se recomienda tener separadas configuraciones globales de otras más particulares.

Es eso lo que toca hacer ahora, hay que crear un archivo “global.xml” para las configuraciones globales, este debe ser creado en el directorio “src/main/mule” del proyecto, haciendo uso del menú contextual para crearlo de la siguiente forma:



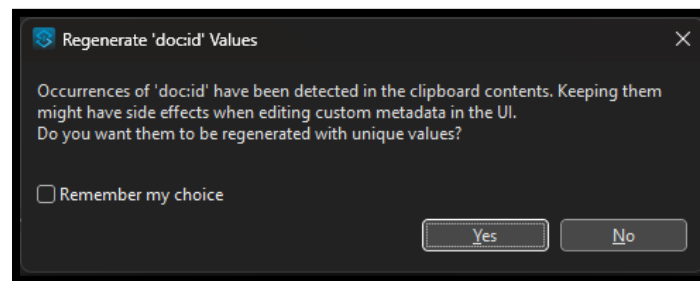
Es importante crear el archivo con su nombre y extensión correctos:



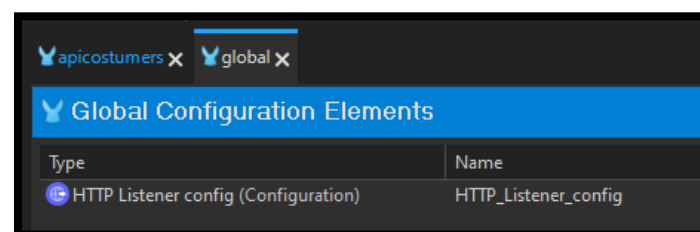
Ahora, hay que pasar la configuración del listener al nuevo archivo que fue creado, para ello es necesario ubicarse en el archivo “apicostumers.xml” o bien, el nombre que tenga según se le haya nombrado al proyecto y de allí, toca extraer (cortar) el código del listener, desde la etiqueta de apertura hasta la de cierre:

```
apicostumers x global.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <mule xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns="http://www.mulesoft.org/schema/mule/core"
4     xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
5     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.mulesoft.org/schema/mule/core http
6     http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd">
7     <http:listener-config name="HTTP Listener config" doc:name="HTTP Listener config" doc:id="e0cb75b5-7f8e-49aa-857a-3766
8     <http:listener-connection host="0.0.0.0" port="8081" />
9     </http:listener-config>
10    <flow name="apicostumersFlow" doc:id="088951a1-6bf3-4f31-8f8f-a452cb540214" >
11        <http:listener doc:name="Listener" doc:id="3372d85c-0264-4d00-9aae-fcc0464ff667" config-ref="HTTP Listener config"
12        <set-payload value="List of Costumers" doc:name="Set Payload" doc:id="a3041ef4-f8f1-4bdf-b0ac-20b3d7daae11" />
13    </flow>
14 </mule>
```

Y este bloque, ahora se ubicará en global.xml, al copiarlo el IDE nos avisara que, si no copiamos los identificadores de los objetos, estos se duplicaran al copiar el código, preguntara si deseamos conservar los mismos identificadores o no, en este caso hay que indicarle que sí, pues solo se desea separar la configuración, pero mantener los objetos (el listener) intactos:

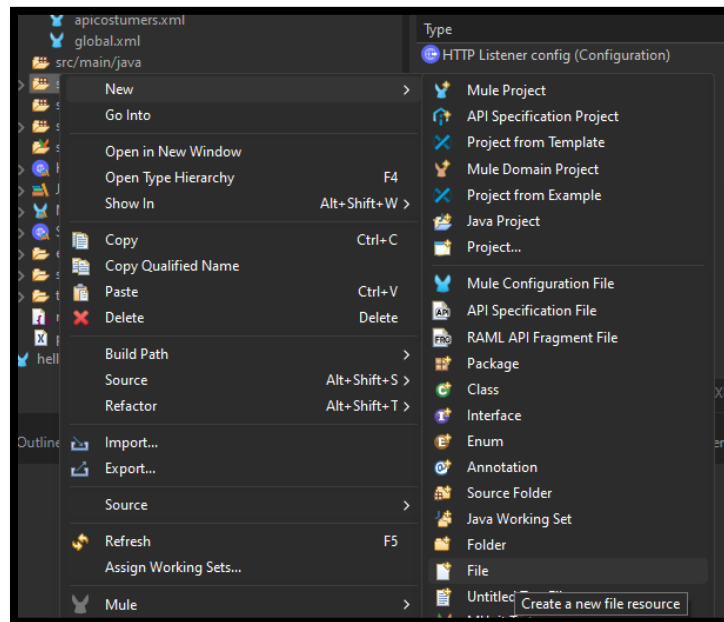


Se guardan cambios y si se hizo correctamente, el IDE no debe marcar errores y ahora el objeto http listener debe ser reconocido en el nuevo archivo de configuración global:

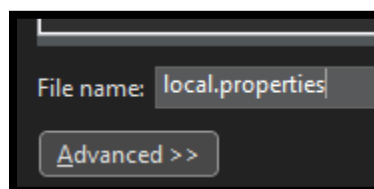


Ahora, otra recomendación que se hace es externalizar los valores codificados en las propiedades, valores como el HOST y el puerto, esto debido a que a medida que crece un proyecto, si desean cambiarse, se vuelve complicado encontrar el objeto correcto y hacer el cambio sin cometer errores.

Para poder hacerlo, vamos a proceder creando un nuevo archivo de propiedades en “src/main/resouces”, de la siguiente manera:



A este archivo se le nombrara local y deberá tener la extensión “properties”, importante que tenga dicha extensión, de la siguiente forma:



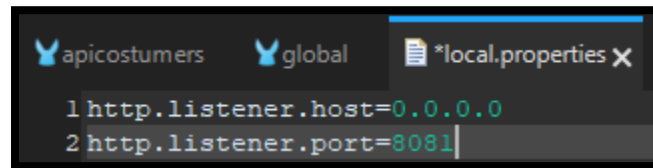
Una vez creado el archivo, se le agregaran las siguientes propiedades:

http.listener.host=0.0.0.0

http.listener.port=8081

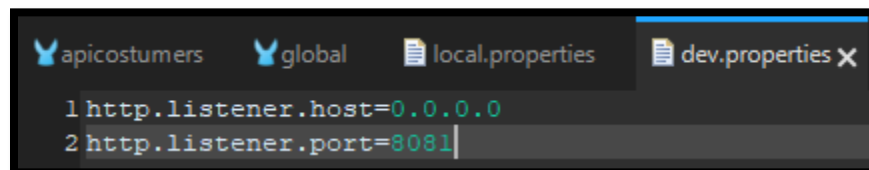


Debe quedar de la siguiente forma:



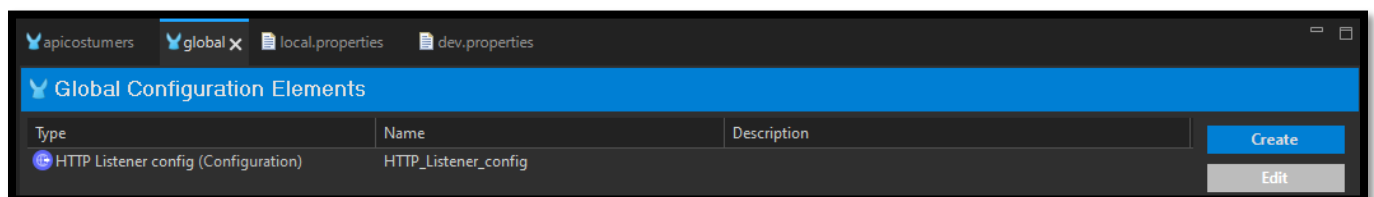
A screenshot of a text editor window with tabs for 'apicostumers', 'global', and '\*local.properties'. The 'local.properties' tab is active and contains two lines of text: '1 http.listener.host=0.0.0.0' and '2 http.listener.port=8081'.

Se guardan los cambios y debe repetirse exactamente el mismo procedimiento, pero ahora con un archivo llamado dev, es decir “dev.properties” el cual tendrá las mismas propiedades y valores.



A screenshot of a text editor window with tabs for 'apicostumers', 'global', 'local.properties', and 'dev.properties'. The 'dev.properties' tab is active and contains two lines of text: '1 http.listener.host=0.0.0.0' and '2 http.listener.port=8081'.

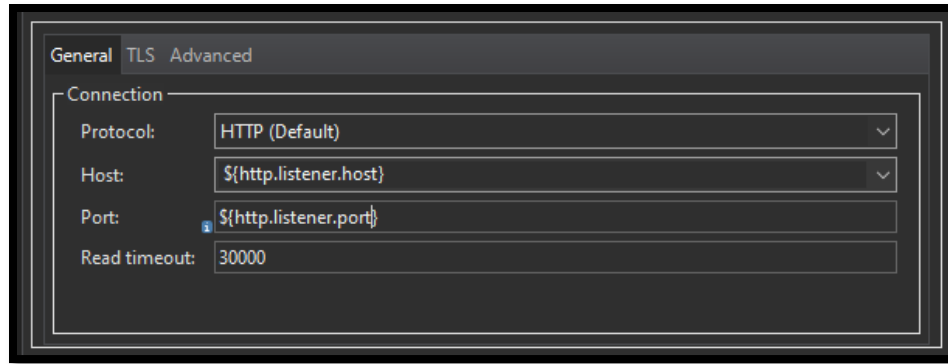
Ahora que están las propiedades de HOST y puerto en los archivos de configuración, ahora hace falta hacerles referencia desde el listener, para ello toca ubicarse en el archivo global.xml, seleccionar nuestro listener y dar clic a “Edit”:



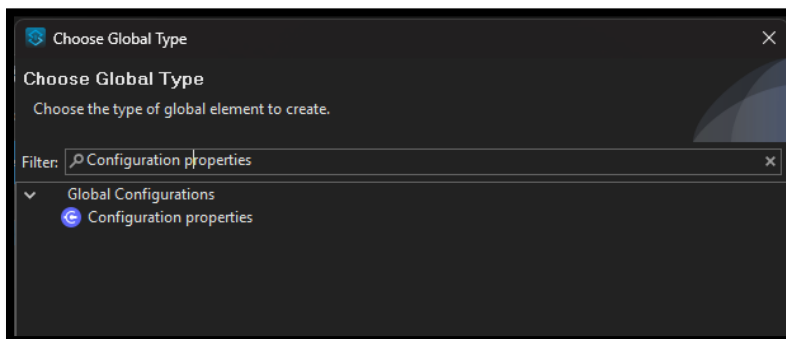
Una vez aquí, en la ventana que se muestra a continuación, podemos referenciar las nuevas propiedades como variables usando la siguiente sintaxis:

`${your.property.name}`

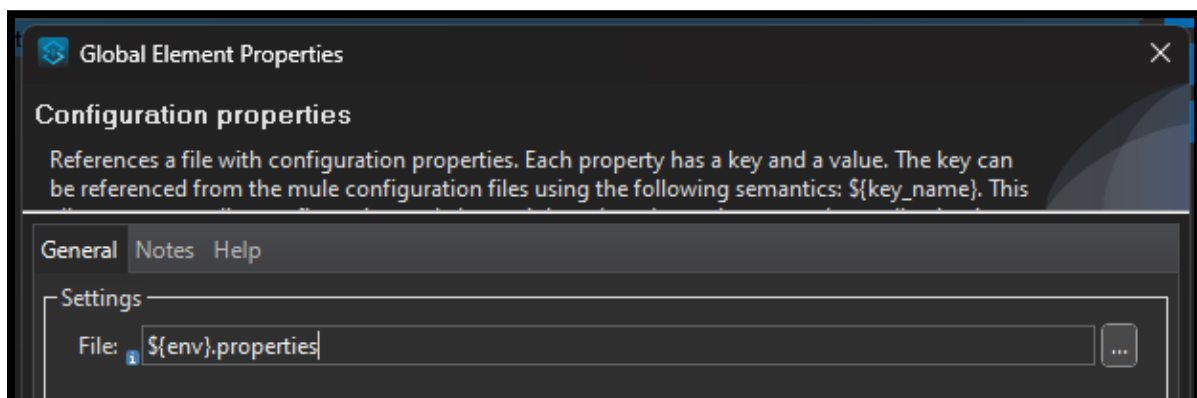
En el caso de esta práctica, adaptando la sintaxis quedaría de la siguiente forma:



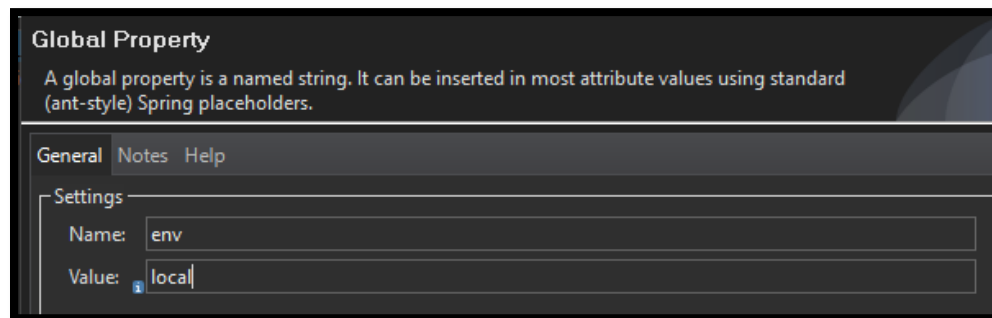
Vamos bien hasta este punto, sin embargo, hace falta indicarle a la aplicación Mule donde encontrar las nuevas propiedades, pues no las reconocerá solo añadiéndolas, para poderlo hacer hay que ir al global.xml en vista de elementos globales y luego damos clic en “Create”, en la ventana que sale debemos buscar “Configuration Properties”:



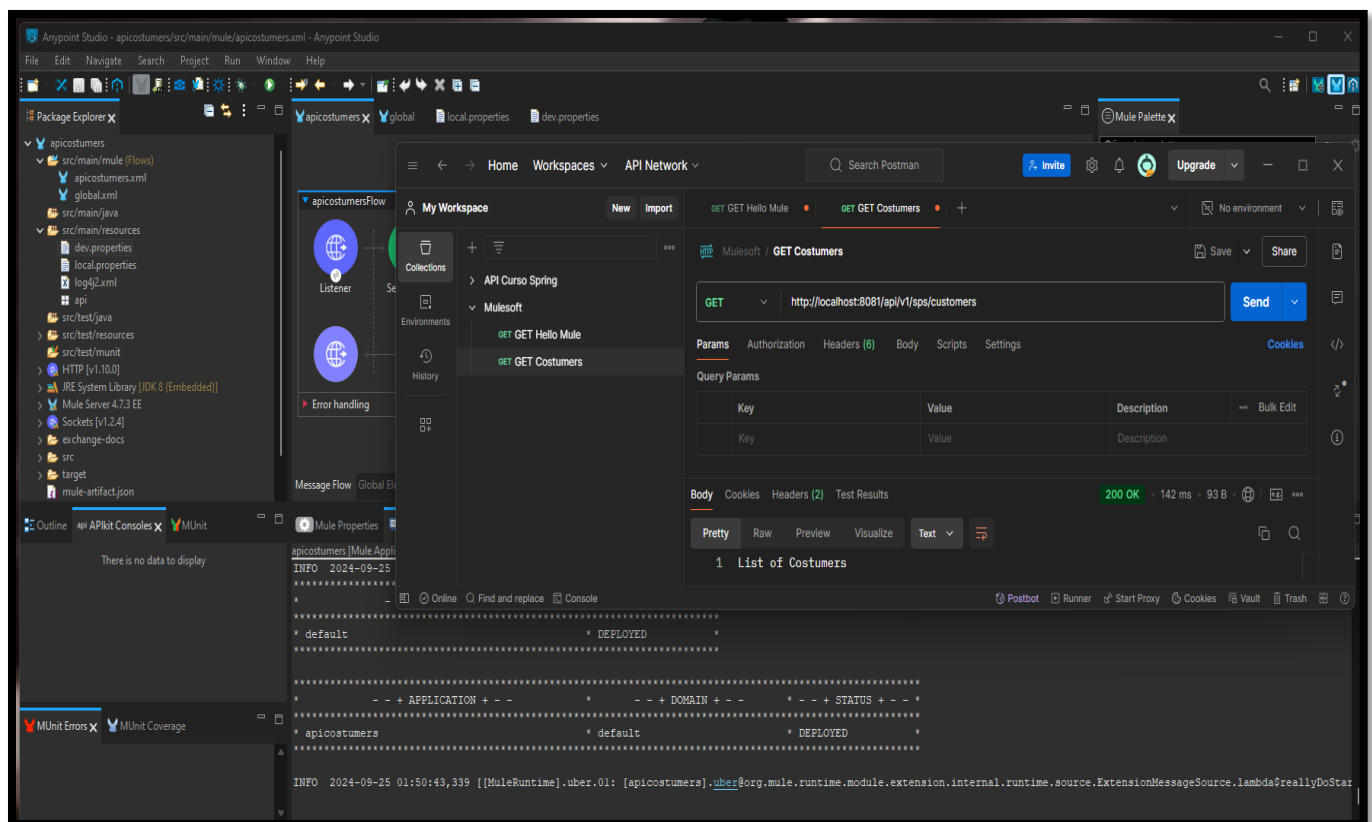
Lo seleccionamos y damos clic en “Ok”, una vez hecho nos saldrá otra ventana en la que tendremos que indicar el archivo de configuración que queremos utilizar, tenemos dos según el entorno, dev y local, para poder alternar entre uno y otro según las necesidades debemos indicarlo como variable, tal como se hizo con las propiedades en el listener, de esta manera:



Sin embargo, la variable especificada aún no existe, vamos a crearla como una propiedad global, para ello damos clic en “Create” nuevamente y la buscamos como “Global Property”, después procedemos a seleccionarla, una vez hecho solicitara un nombre y un valor para la propiedad, en este caso asignamos lo siguiente:



Aceptamos y ahora, antes de continuar con lo siguiente, hay que verificar que nuestra API siga funcionando correctamente, solo hemos realizado cambios considerados buenas prácticas en cuanto a los archivos de configuración del proyecto, por lo que, si todo se realizó correctamente, la API debe seguir funcionando como antes.



Ahora que verificamos que hasta este punto todo es correcto, podemos proceder a crear otros nuevos dos archivos de propiedades para el proyecto, estos serán para propiedades seguras y contendrán las siguientes propiedades:

-Para el archivo local.secure.properties:

*example.username=myUsernameLocal*

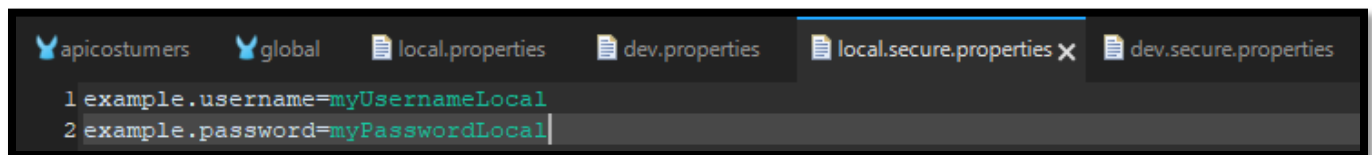
*example.password=myPasswordLocal*

-Para el archivo dev.secure.properties:

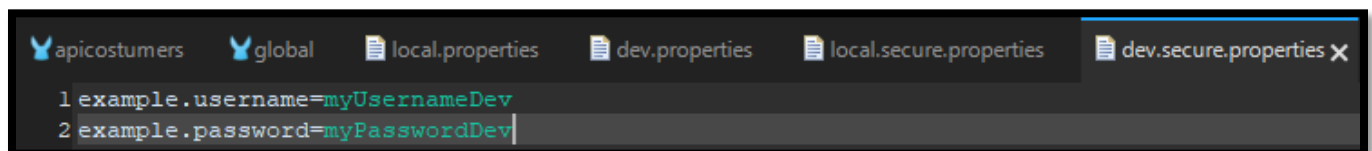
*example.username=myUsernameDev*

*example.password=myPasswordDev*

Deben quedar de la siguiente forma:



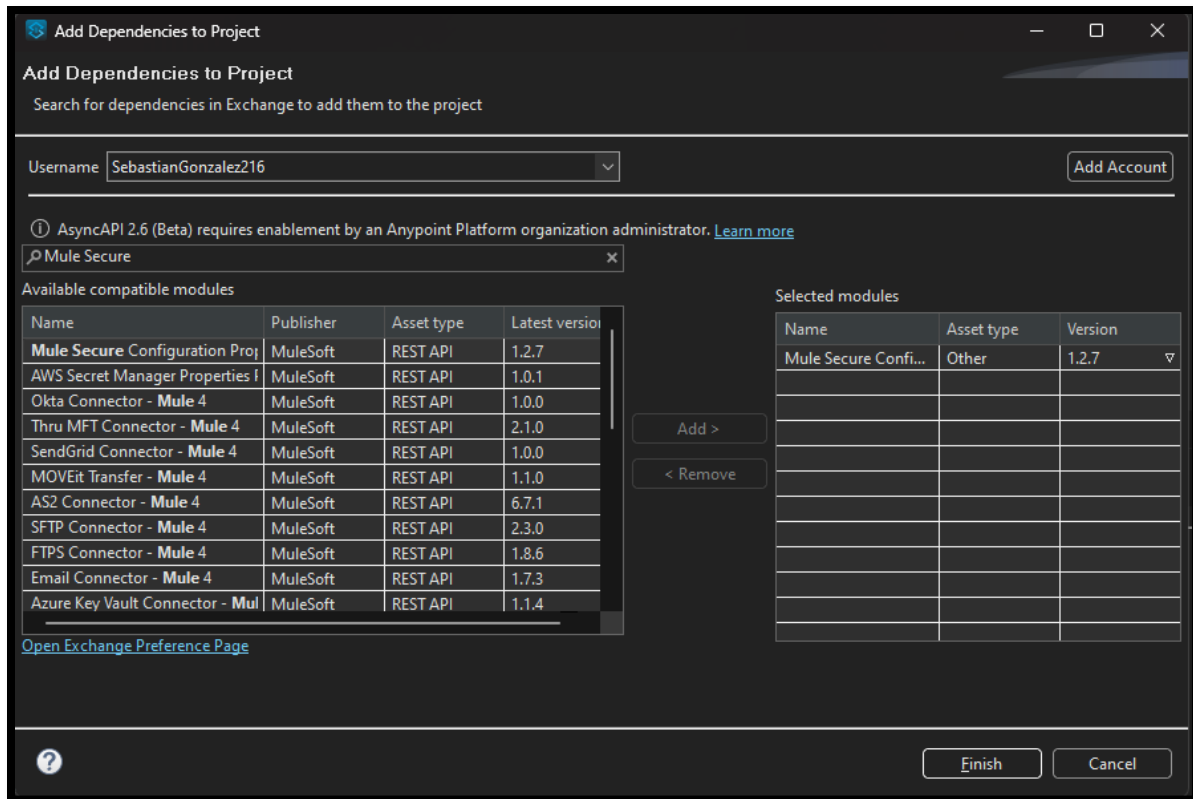
```
1 example.username=myUsernameLocal
2 example.password=myPasswordLocal
```



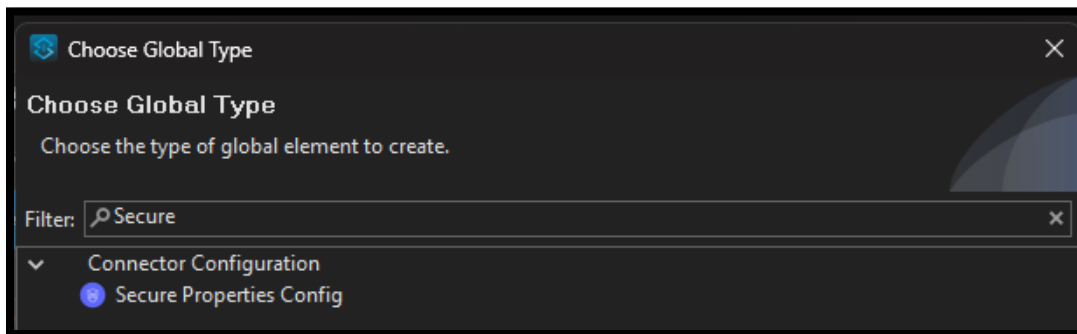
```
1 example.username=myUsernameDev
2 example.password=myPasswordDev
```

Luego de haberlos añadido, ocupamos un nuevo módulo llamado “Mule Secure Configuration Properties” para buscarlo y añadirlo debemos ubicarnos en el archivo global.xml, en el modo “Message Flow” y después, visualizar el panel lateral derecho (Mule Palette), una vez allí daremos clic en “Search in Exchange”.

Ya realizado el paso anterior, se desplegará una ventana de búsqueda de módulos, allí buscaremos el módulo anteriormente mencionado, lo seleccionaremos y daremos clic en “Add”, después a “Finish”.



Una vez importado el módulo Mule Secure Configuration Properties, se debe ir a la vista de elementos globales, hacer clic en el botón “Create” nuevamente y ahora se debe crear un “Secure Properties Config”:



Una vez allí, en File debe colocarse el siguiente valor:

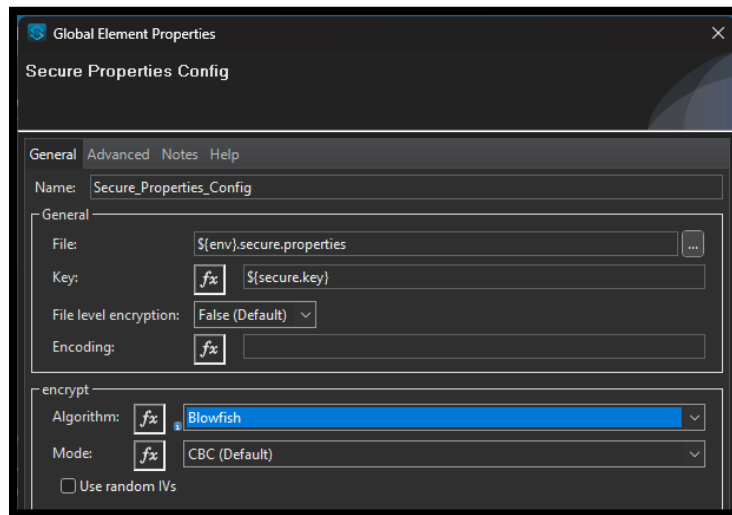
`${env}.secure.properties`

Que es el de nuestro archivo de configuración dinámico.

En Key, pasaremos la siguiente propiedad variable:

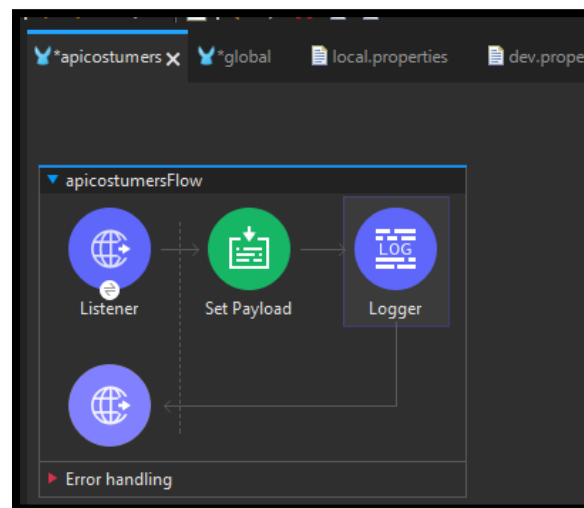
`${secure.key}`

Además, debe seleccionarse Blowfish como algoritmo de encriptación, quedando de la siguiente manera:



Una vez quede de esa forma, se deberán guardar los cambios.

Ahora, necesitamos un componente de registro para que reciba las credenciales de acceso y las devuelva por consola, el componente que necesitamos ahora es un “Logger” y puede encontrarse en el módulo “Core” en la paleta de módulos del IDE, este debe añadirse al flujo de proceso después de nuestro Set Payload, de la siguiente manera:



Una vez añadido, toca ir a las propiedades del objeto o componente Logger y en la propiedad “Message”, debemos dar clic al botón “fx”, esto abrirá una vista en la que debemos agregar el siguiente código:

```
output application/java
```

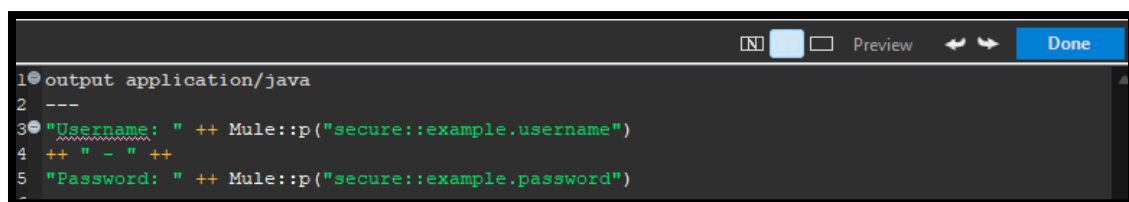
```
---
```

```
"Username: " ++ Mule::p("secure::example.username")
```

```
++ " - " ++
```

```
"Password: " ++ Mule::p("secure::example.password")
```

Quedando de esta forma:



```
1 output application/java
2 ---
3 "Username: " ++ Mule::p("secure::example.username")
4 ++ " - " ++
5 "Password: " ++ Mule::p("secure::example.password")
```

“secure::” indica que los valores deberán ser descifrados antes de ser usados por la aplicación, una vez hecho esto guardamos cambios y procedemos a cifrar los valores que creamos de user y password en los archivos de propiedades.

Para realizar dicha tarea, necesitaremos el archivo JAR “Secure Properties Tool”, el cual fue proporcionado por SPS, sin embargo, puede descargarse desde la documentación oficial de MuleSoft [AQUÍ](#).

Una vez descargado el archivo JAR, vamos a abrir una ventana de la terminal y a ubicarnos en el directorio en el que tengamos ubicado dicho archivo, ya sea navegando por la terminal o abriéndola directamente en la carpeta que contiene el archivo JAR, en mi caso utilizare PowerShell de Windows:

```
Administrador: Windows Pow X + v
PS C:\Users\Ragnar\Downloads> ls

Directorio: C:\Users\Ragnar\Downloads

Mode                LastWriteTime         Length Name
----                -
d-----          23/09/2024   06:14 p. m.      AnypointStudio
d-----          23/09/2024   12:33 p. m.          escl
d-----          23/09/2024   07:05 p. m.      resources
-a-----          24/09/2024   04:27 p. m.    809460 007422007402567644_202409.pdf
-a-----          24/09/2024   04:24 p. m.    463049 1562560610_202409.pdf
-a-----          23/09/2024   02:35 p. m.    172841 461133223_122132676266333949_8761654971737063751_n.jpg
-a-----          23/09/2024   06:00 p. m.  2281507830 AnypointStudio-7.18.1-win64.zip
-a-----          23/09/2024   10:57 p. m.    12538 comando_encryptacion.docx
-a-----          24/09/2024   06:59 p. m.    561504 integrated-circuit-board-wallpaper-technology-digital-motherboard-background-free-vector.jpg
-a-----          24/09/2024   06:45 p. m.    785524 MuleSoft-API-integration-in-finance.webp
-a-----          23/09/2024   03:35 p. m.    516595 Práctica - Mulesoft Trainee 2024.docx.pdf
-a-----          23/09/2024   03:35 p. m.    134342 secure-properties-tool.jar

PS C:\Users\Ragnar\Downloads> |
```

En mi caso, podemos ver el archivo JAR al final de la lista, haciendo uso del comando “ls” para generarla, como se puede apreciar yo estoy ubicado en el directorio “Downloads”, una vez ubicado utilizaremos el siguiente comando para hacer uso del archivo JAR para encriptar nuestro valor para el user:

```
java -cp secure-properties-tool.jar com.mulesoft.tools.SecurePropertiesTool string encrypt Blowfish CBC MyMuleSoftKey "myUsernameLocal"
```

Y el siguiente para el valor del password:

```
java -cp secure-properties-tool.jar com.mulesoft.tools.SecurePropertiesTool string encrypt Blowfish CBC MyMuleSoftKey "myPasswordLocal"
```

Hagámoslo con ambos valores, al ingresar el comando deberá devolvernos el valor cifrado directamente en la terminal, de esta forma:



```
Administrador: Windows Pow x + -
PS C:\Users\Ragnar\Downloads> java -cp secure-properties-tool.jar com.mulesoft.tools.SecurePropertiesTool string encrypt
Blowfish CBC MyMuleSoftKey "myUsernameLocal"
HbsuWJRjiubchmzQREGdsA==
PS C:\Users\Ragnar\Downloads> java -cp secure-properties-tool.jar com.mulesoft.tools.SecurePropertiesTool string encrypt
Blowfish CBC MyMuleSoftKey " myPasswordLocal"
d/gfosIOEPpofTvc2MX1SekjGN1UepI7
PS C:\Users\Ragnar\Downloads> |
```

Ahora, los valores encriptados que nos devuelve debemos pasarlos a nuestras propiedades seguras a sus respectivas propiedades que les corresponden, los valores encriptados deben ir entre la siguiente sintaxis:

*![encryptedValue]*

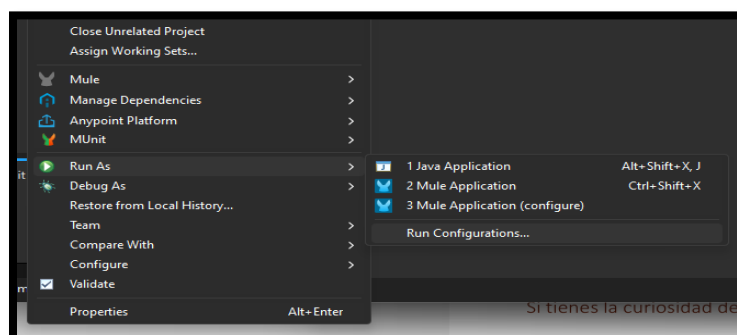
Reemplazando “encryptedValue” por los valores obtenidos en la terminal, debemos hacer esto en todas nuestras propiedades seguras tanto de dev como locales, quedando de esta forma:

```
apicostumers global local.properties dev.properties local.secure.properties x dev.secure.properties
1 example.username=![HbsuWJRjiubchmzQREGdsA==]
2 example.password=![d/gfosIOEPpofTvc2MX1SekjGN1UepI7]
```

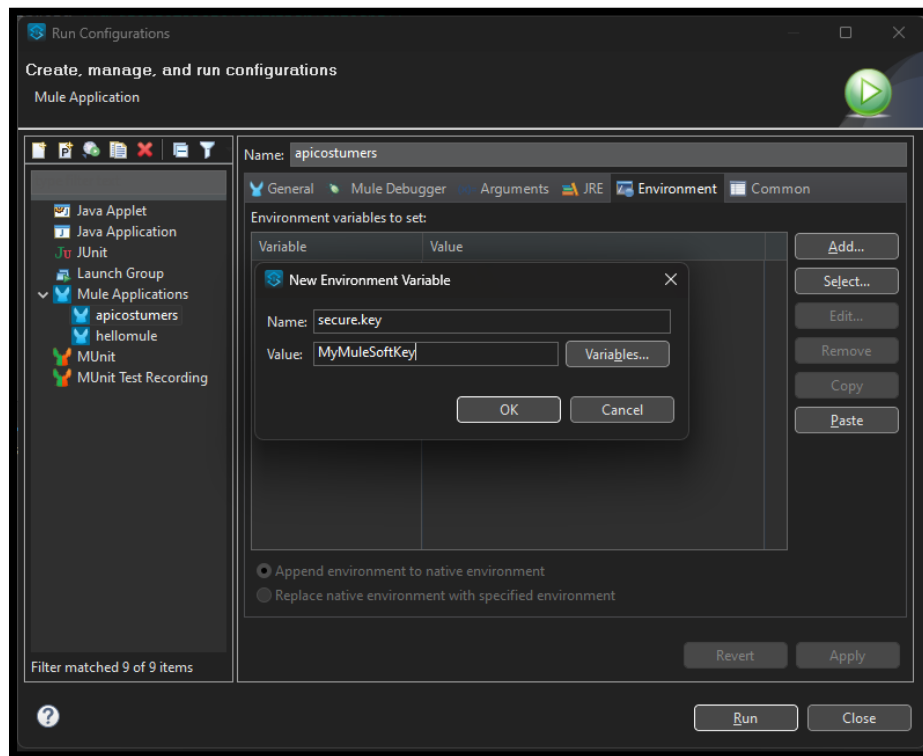
```
apicostumers global local.properties dev.properties local.secure.properties dev.secure.properties x
1 example.username=![HbsuWJRjiubchmzQREGdsA==]
2 example.password=![d/gfosIOEPpofTvc2MX1SekjGN1UepI7]
```

Una vez realizado, guardamos todos los cambios.

Ahora debemos probar nuevamente nuestra aplicación de forma local, primero debemos cambiar algunas configuraciones de ejecución de la aplicación, para ello debemos dar clic derecho sobre nuestro proyecto Mule, luego dar clic a “Run As” y después, a “Run Configurations”:



Una vez allí, nos ubicamos en la pestaña “*Environment*” y damos clic en “*Add*” y añadimos los siguientes valores:



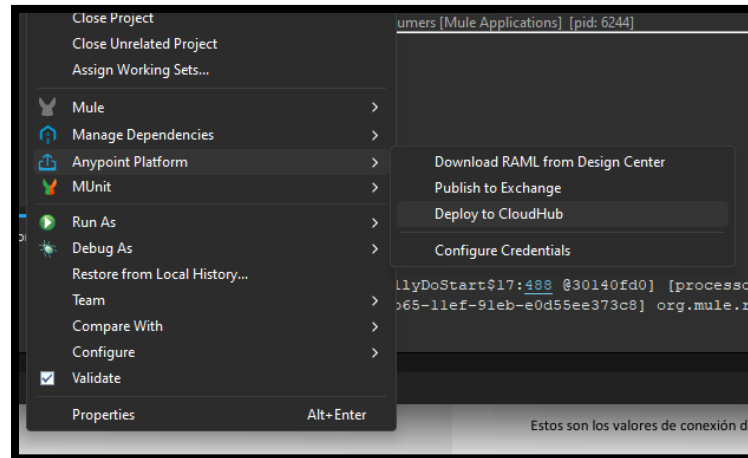
Hacer esto enviara la propiedad `secure.key` en tiempo de ejecución, evitando así el riesgo de exponerla en el código fuente. Una vez realizado guardamos cambios y damos clic en “*Run*” para correr la aplicación, si el estado de la aplicación es `DEPLOYED` abrimos Postman para enviar una solicitud, una vez enviada una solicitud y recibida una respuesta exitosa, debemos ver nuestros valores de user y password descriptados en la consola:

```
[processor: ; event: ] org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource: Message source '
org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Username: myUsernameLocal - Password: myPasswordLocal
```

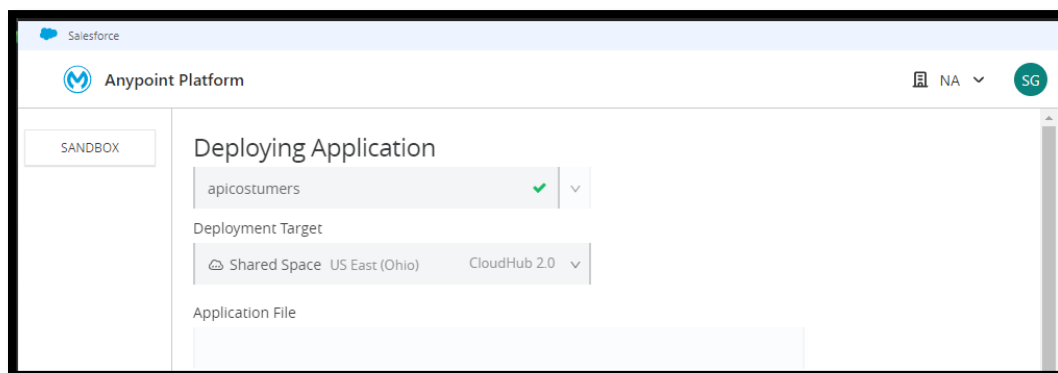
Si aparecen los valores que utilizamos originalmente, es decir, antes de realizar la encriptación con el archivo JAR, el proceso fue realizado correctamente.

### 3.3-Despligue en CloudHub

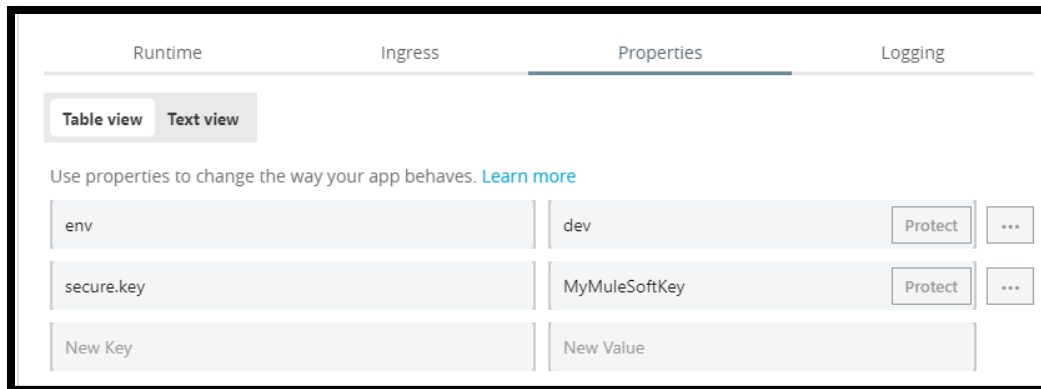
Para poder desplegar nuestra aplicación, debemos dar clic derecho al proyecto, luego posicionarse en “Anypoint Platform” y después dar clic a “Deploy to CloudHub”



En la ventana nueva, del lado izquierdo debemos seleccionar el entorno SANDBOX y asignar un nombre único a nuestra aplicación, cabe mencionar que, si no lo hemos hecho, el IDE nos solicitara iniciar sesión en Anypoint Platform con nuestra cuenta.



Antes de hacer el DEPLOY de la aplicación, es importante también añadir valores secure.key y env a las propiedades de la implementación, para ello nos vamos a la pestaña “Properties” en la parte inferior de la ventana y añadimos las propiedades anteriormente mencionadas, de la siguiente forma:



Y, por último, antes de desplegar, vamos a ocultar el valor de secure.key, para ocultarlo debemos ir al archivo “mule-artifact.json” de nuestro proyecto y agregar la siguiente línea:

*"secureProperties": ["secure.key"]*

Recordando respetar el formato del json (comas, corchetes y llaves) para evitar errores, quedando de esta forma:

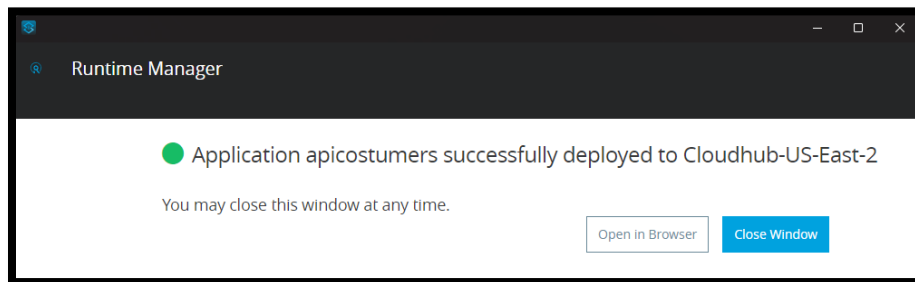
```

1 {
2   "minMuleVersion": "4.7.0",
3   "secureProperties": ["secure.key"],
4   "javaSpecificationVersions": [
5     "1.8"
6   ]
7 }
```

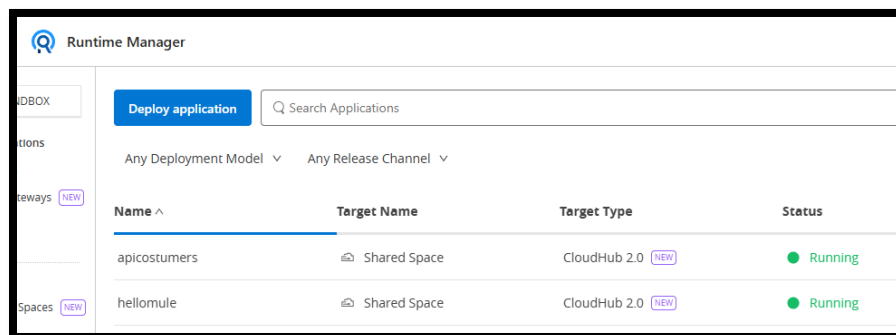
Una vez realizados todos estos pasos, ahora si procedemos a desplegar la aplicación, dicho proceso puede demorar unos minutos:



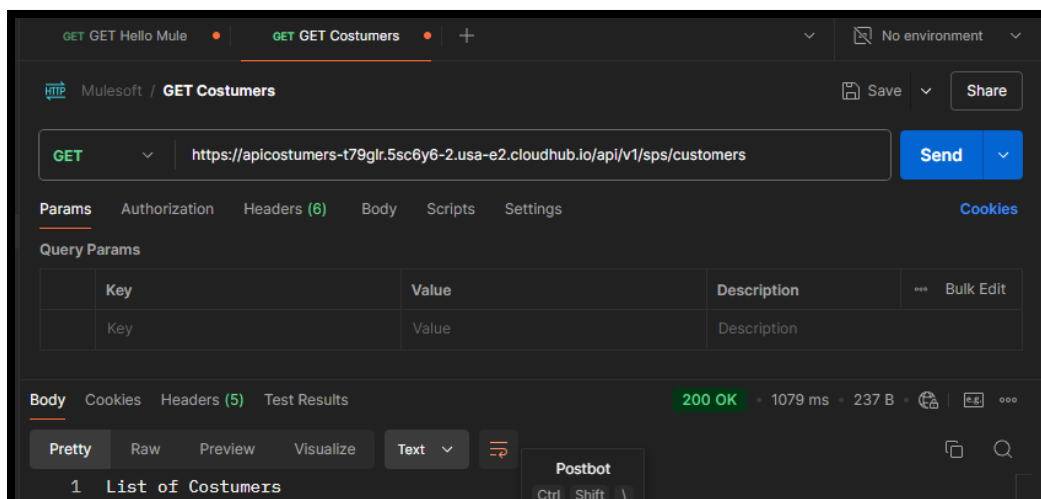
Una vez completado el proceso, el IDE nos notificara y podremos ver nuestra API en el navegador, en la sección Runtime Manager de Anypoint Platform:



Podemos ver el estado *"Running"* de nuestra API:



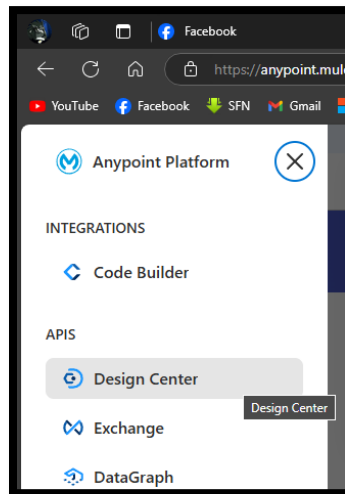
Hacemos clic en ella en la tabla anteriormente mostrada para poder visualizar su endpoint público, al cual le añadiremos el endpoint personalizado que nosotros creamos para poder hacer una solicitud, de la siguiente forma:



Con esta última solicitud verificamos que todo se realizó correctamente.

### 3.4-Espesificación de la API

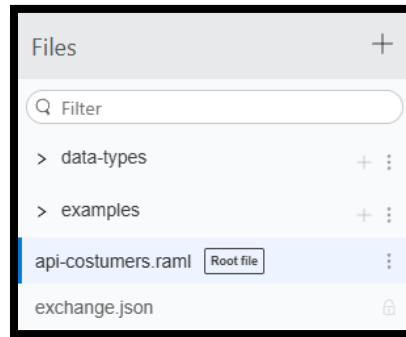
Para crear una especificación para la API recién creada Anypoint Studio también nos ofrece una herramienta, para comenzar debemos iniciar sesión en su plataforma e ir a su sección “*Design Center*”, la podemos encontrar en su página y menú principales:



Una vez allí, es posible crear un nuevo diseño dando clic en el botón “Create” ubicado en la parte superior izquierda y seleccionando “*New API Specification*”, una vez allí debemos darle un nombre al diseño y seleccionar un lenguaje de especificación, en este caso usare RAML 1.0

A screenshot of the 'New API specification' form in the Anypoint Platform. The form has a title bar 'New API specification'. Below it, there is a 'Project name (required)' field with the text 'API Costumers'. A question 'How do you want to draft the API Spec?' is followed by two radio button options. The first option, 'I'm comfortable designing it on my own', is selected and includes the subtext 'A complete code editing experience with interactive documentation'. Below this, a 'Specification Language' dropdown menu is set to 'RAML 1.0'. The second option, 'Guide me through it', is unselected and includes the subtext 'Use a visual interface scaffolding the API Specification (can generate both RAML & OAS)'. At the bottom, there is a GitHub logo and text about using GitHub to store and collaborate on API specifications, with an 'Authorize' button. At the very bottom right, there are 'Cancel' and 'Create API' buttons.

Una vez creado, tendremos un editor similar al IDE con un explorador de archivos, un editor y una consola, en el explorador vamos a crear dos carpetas, una llamada “*data-types*” y otra llamada “*examples*”:



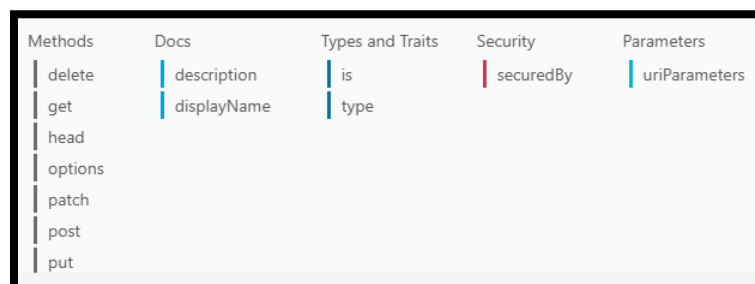
En el archivo `api-costumers.raml` podemos ir generando nuestro diseño a base de código, para generar nuestro endpoint, nos ubicamos al final del código, damos un espacio y escribimos nuestro endpoint tal cual lo deseamos, finalizando con un “.”, en este caso sería:

`/api/v1/sps/customers:`

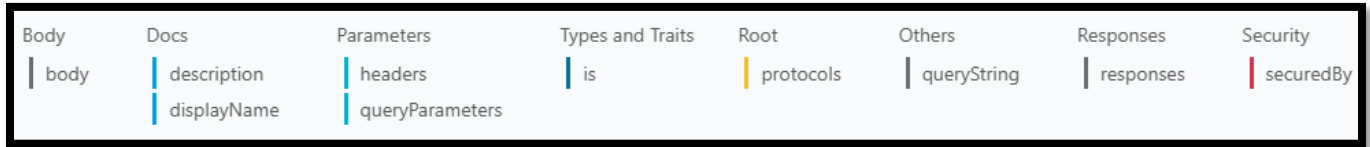
Y quedaría de la siguiente forma:

```
1  #%RAML 1.0
2  title: API Costumers
3
4  /api/v1/sps/customers:
5
```

Una vez escrito, damos Enter para pasar a la siguiente línea y automáticamente el editor nos sugerirá métodos que podría realizar este endpoint, en este caso seleccionare GET, pues lo que se necesita es obtener a los clientes.



Una vez seleccionado, el mismo editor nos sugiera sugiriendo formas de autocompletado dependiendo de las operaciones que deseemos hacer, en el caso de GET, nos arrojará lo siguiente:



En este caso, añadiré un nombre y una descripción para el método, de esta forma:

```
1  #%RAML 1.0
2  title: API Costumers
3
4  /api/v1/sps/customers:
5    get:
6      displayName: Get All Costumers
7      description: Displays a list of all customers
```

Después, podríamos añadir Query Parameters, sin embargo, para el caso de esta API no hacemos uso de pase de parámetros, por lo que omitiremos este paso y pasaremos directo a indicar el tipo de response que debe generar este método, por lo general para los métodos GET el tipo de respuesta es la del código 200, la añadimos de la siguiente manera:

```
1  #%RAML 1.0
2  title: API Costumers
3
4  /api/v1/sps/customers:
5    get:
6      displayName: Get All Costumers
7      description: Displays a list of all customers
8      responses:
9        200:
```

Después, debemos definir el formato en el que se devolverá la información, en este caso ocuparemos json y lo podemos hacer de la siguiente forma:

```
1  #%RAML 1.0
2  title: API Costumers
3
4  /api/v1/sps/customers:
5    get:
6      displayName: Get All Costumers
7      description: Displays a list of all customers
8      responses:
9        200:
10         body:
11           application/json:
```



Ahora, debido a que no contamos con una base de datos, vamos a crear un archivo json con una lista de clientes hipotética que respondería este método, vamos a crear este archivo dentro de la carpeta data-types creada anteriormente, nombraremos a este archivo costumers.json y lo llenaremos con los siguiente:

```
{
  "customers": [
    {
      "id": 1,
      "name": "John Smith",
      "email": "john.smith@example.com",
      "age": 28,
      "active": true
    },
    {
      "id": 2,
      "name": "Mary Johnson",
      "email": "mary.johnson@example.com",
      "age": 34,
      "active": true
    },
    {
      "id": 3,
      "name": "Charles Brown",
      "email": "charles.brown@example.com",
      "age": 22,
      "active": false
    },
    {
      "id": 4,
      "name": "Anna Davis",
      "email": "anna.davis@example.com",
      "age": 29,
      "active": true
    },
    {
      "id": 5,
      "name": "Louis Miller",
      "email": "louis.miller@example.com",
      "age": 40,
      "active": false
    }
  ]
}
```

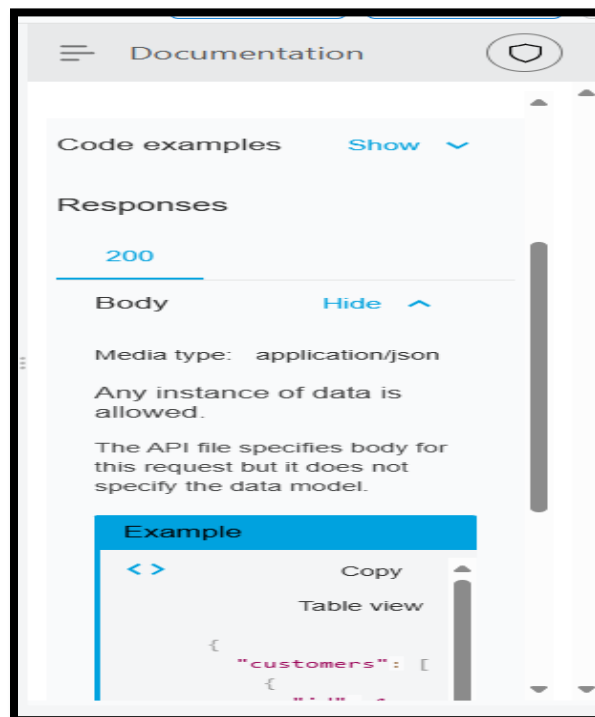
El archivo también puede importarse, debe quedar así:



Ahora, solo debemos indicarle al código en RAML que tome nuestro archivo json para la respuesta, podemos hacerlo añadiendo el siguiente código:

```
1 ##RAML 1.0  
2 title: API Costumers  
3  
4 /api/v1/sps/customers:  
5   get:  
6     displayName: Get All Costumers  
7     description: Displays a list of all customers  
8     responses:  
9       200:  
10        body:  
11          application/json: !include data-types/costumers.json
```

Si lo hicimos correctamente, podemos ver en la sección de documentación que ahora al ejecutar el método GET nos debe aparecer el ejemplo creado con el código de respuesta 200:



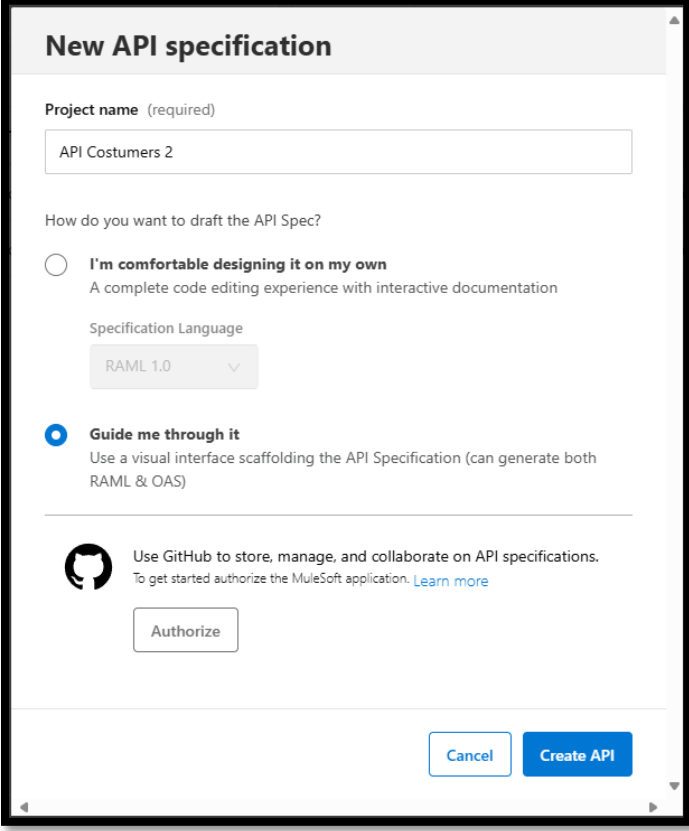
## 4.-Extras

El Conseguí realizar estos puntos extra con éxito, sin embargo, seré más breve en cuando al desarrollo de estos en comparación a los anteriores debido a lo mucho que se extendió mi documentación, comprendo el que se me está prestando un tiempo para revisar esto, tiempo que agradezco y aprecio, por ello decidí colocar un índice y en esta sección, tratare de ir más al grano.

### 4.1-Creación de especificación de API en API Manager

Ahora Diseñaremos nuestra primera especificación de API en el Diseñador de API de Anypoint Design Center, es esencialmente lo que ya se hizo anteriormente con RAML, pero esta vez de forma guiada por la plataforma.

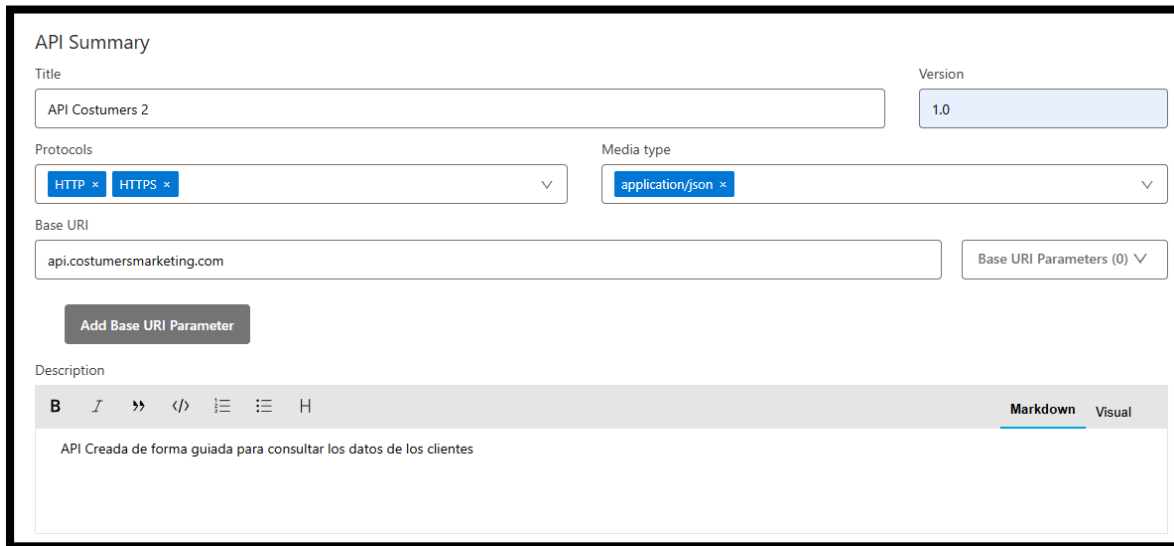
Debemos ir al Design Center de la plataforma, una vez allí vamos a Create New y luego a Create new API Spec, vamos a nombrarla API Costumers 2 para diferenciarla de la anterior y esta vez, usaremos “*Guide me through it*”:



The screenshot shows a modal window titled "New API specification". It contains the following elements:

- Project name (required):** A text input field containing "API Costumers 2".
- How do you want to draft the API Spec?**
  - ☐ **I'm comfortable designing it on my own**  
A complete code editing experience with interactive documentation
  - ☒ **Guide me through it**  
Use a visual interface scaffolding the API Specification (can generate both RAML & OAS)
- Specification Language:** A dropdown menu currently showing "RAML 1.0".
- GitHub Integration:** A section with the GitHub logo, text "Use GitHub to store, manage, and collaborate on API specifications. To get started authorize the MuleSoft application. [Learn more](#)", and an "Authorize" button.
- Buttons:** "Cancel" and "Create API" buttons at the bottom right.

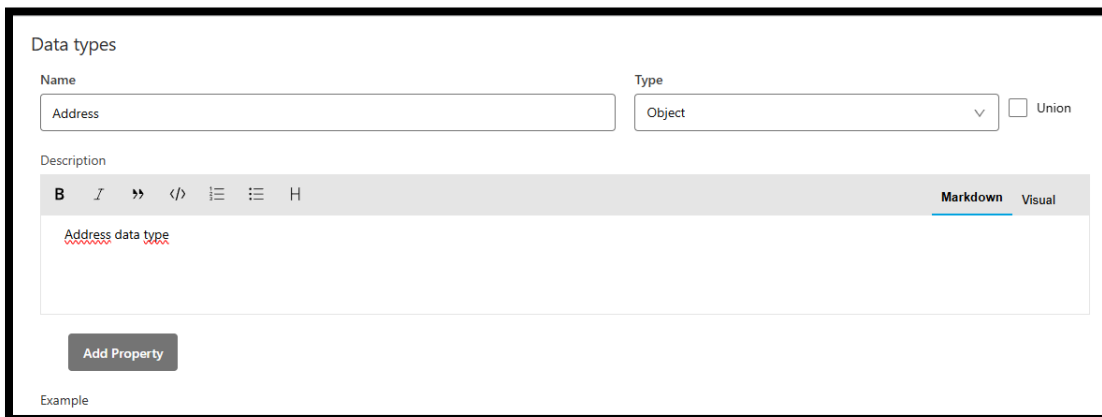
A continuación, vamos a utilizar los siguientes datos para la API:



The screenshot shows the 'API Summary' configuration interface. It includes a 'Title' field with the value 'API Costumers 2', a 'Version' dropdown set to '1.0', a 'Protocols' dropdown with 'HTTP' and 'HTTPS' selected, and a 'Media type' dropdown set to 'application/json'. The 'Base URI' field contains 'api.costumersmarketing.com', and there is a button 'Add Base URI Parameter'. Below these fields is a 'Description' section with a rich text editor containing the text 'API Creada de forma guiada para consultar los datos de los clientes'. The editor has tabs for 'Markdown' and 'Visual'.











A la derecha, podremos ver como se genera el código automáticamente.

En el panel izquierdo, podremos añadir datos a nuestra API dando clic en “+” del Data Types, Vamos a llenarlo con los datos que se muestran en la siguiente captura de pantalla:



The screenshot shows the 'Data types' configuration interface. It includes a 'Name' field with the value 'Address', a 'Type' dropdown set to 'Object', and a 'Union' checkbox. Below these fields is a 'Description' section with a rich text editor containing the text 'Address data type'. The editor has tabs for 'Markdown' and 'Visual'. At the bottom, there is a button 'Add Property' and an 'Example' label.

También vamos a añadir propiedades con los siguientes campos y propiedades:

Property Name	Type	Required	Union	Actions
street	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	 
city	String	<input type="checkbox"/>	<input type="checkbox"/>	 
postalCode	String	<input type="checkbox"/>	<input type="checkbox"/>	 
state	String	<input type="checkbox"/>	<input type="checkbox"/>	 
country	String	<input type="checkbox"/>	<input type="checkbox"/>	 

[Add Property](#)

Luego, llenaremos estos campos con un json de ejemplo, usaremos el siguiente:

```
{
  "street": "44 Shirley Ave.",
  "city": "West Chicago",
  "postalCode": "60185",
  "state": "IL",
  "country": "USA"
}
```

Example

Inherited

Edit

```
{
  "street": "44 Shirley Ave.",
  "city": "West Chicago",
  "postalCode": "60185",
  "state": "IL",
  "country": "USA"
}
```

Y listo, el siguiente paso es crear otro Data Type, este debe quedar de la siguiente manera:

JSON:

```
{
  "firstName": "Danny",
  "lastName": "Brookshire",
  "phone": "123-412-3412",
  "email": "danny.brookshire@example.com",
  "postalAddress": {
    "street": "44 Shirley Ave.",
    "city": "West Chicago",
    "postalCode": "60185",
    "state": "IL",
    "country": "USA"
  }
}
```

### Data types

Name

Costumer

Type

Object

☐ Union

Description

**B** *I* » </> ☰ ☷ H

Markdown Visual

Costumer data type

Property Name

firstName

☐ Required

Type

String

☐ Union

🔗 ▼

Property Name

lastName

☒ Required

Type

String

☐ Union

🔗 ▼

Property Name

phone

☐ Required

Type

String

☐ Union

🔗 ▼

Property Name

email

☐ Required

Type

String

☐ Union

🔗 ▼

Property Name

postalAddress

☐ Required

Type

Address

☐ Union

🔗 ▼

Add Property

Example

```
"firstName": "Danny",
"lastName": "Brookshire",
"phone": "123-412-3412",
"email": "danny.brookshire@example.com",
"postalAddress": {
  "street": "44 Shirley Ave.",
```

Y listo, el siguiente paso es crear el recurso para el método, podemos hacerlo dando clic en el “+” de “Resources” y vamos a darle las siguientes propiedades:

Resource path

/api/v1/sps/customers

URI Parameters (0) >

GET POST PUT PATCH DELETE OPTIONS HEAD

Summary Responses (0) Query Parameters (0) Headers (0)

Name

Get list of costumers

Secured By

Select...

Description

**B** *I* » </> ≡ ≡ H

Markdown Visual

Vamos a añadirle un response con las siguientes características:

Summary Responses (1) Query Parameters (0) Headers (0)

200

Add

Status

200 - OK

Description

**B** *I* » </> ≡ ≡ H

Markdown Visual

Bodies (1)

Media Type

application/json

Type

Array

Union

Items

Items Type

Costumer

Union

Vamos a hacer lo mismo ahora para un endpoint que permita obtener un cliente en específico, primero, estas serán sus propiedades (Incluya el parámetro URI):

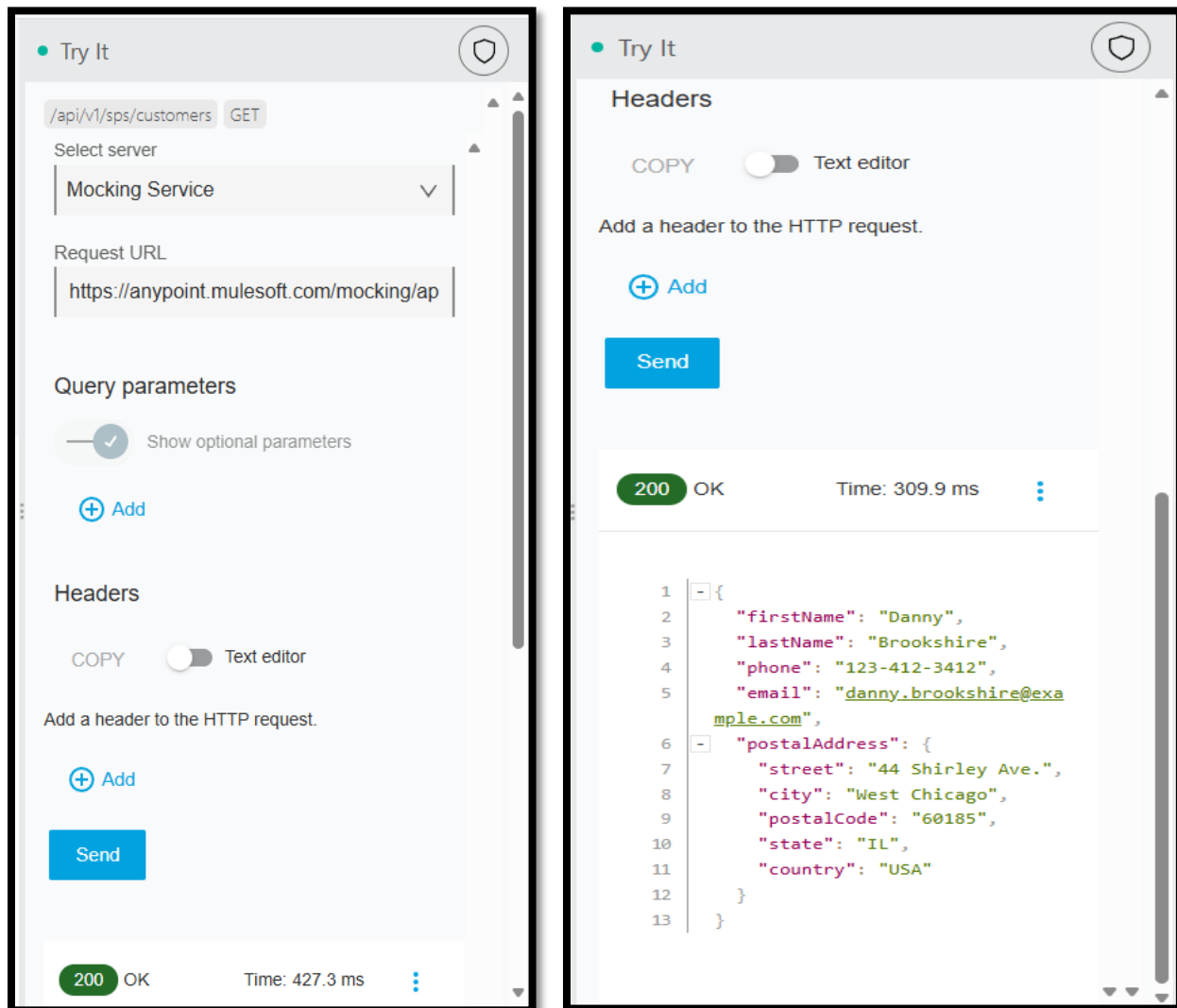
The screenshot shows the configuration for a new API endpoint. The 'Resource path' is set to `/api/v1/sps/customers/{customerId}`. Under 'URI Parameters (1)', a parameter named `customerId` is defined as a required 'String'. The HTTP method is set to 'GET'. The 'Name' field is 'Get customer by ID', and 'Secured By' is set to 'Select...'. The interface includes tabs for 'Summary', 'Responses (0)', 'Query Parameters (0)', and 'Headers (0)', with 'Summary' currently selected.

Y este será su Response:

The screenshot shows the 'Responses (1)' tab for the endpoint. The 'Status' is set to '200 - OK'. The 'Description' field is empty, with a rich text editor toolbar visible. Under 'Bodies (1)', a body is defined with 'Media Type' as 'Default (application/json)' and 'Type' as 'Costumer'. The interface includes tabs for 'Summary', 'Responses (1)', 'Query Parameters (0)', and 'Headers (0)', with 'Responses (1)' currently selected.



Ahora podemos probar la API en un servicio de simulación que nos ofrece la plataforma, he aquí la prueba de ambos métodos GET:



Por último, podemos publicar la API haciendo clic en “*Publish*” en la parte superior derecha de la plataforma.

**Publishing to Exchange**

**Asset version** (required)  
1.0.0

**API version** (required)  
1.0

**LifeCycle State**

☒ **Stable**  
State of release, ready to consume

☐ **Development**  
In Process of Design and Development

> Advanced options

**About asset versioning**  
To publish to Exchange, please, use [Semantic Versioning](#). Examples of good versions are 1.0.0 or 4.3.1.

**More help**

- [Changing a project's main/root file](#)
- [What is an API version?](#)

The lifecycle state of an asset shows its status in the software development lifecycle, from development to stable releases to deprecation.  
[Learn more](#)

Cancel Publish to Exchange

Una vez publicada otros usuarios podrán acceder a ella, yo omitiré este paso debido a que ya tengo la API anterior publicada, la intención de este punto era mostrar esta forma alternativa de creación, más completa y sencilla.

## 4.2-Configuración de API Autodiscovery

Ahora veremos como configurar autodescubrimiento de API en Anypoint Studio, vinculando la aplicación de Mule con la aplicación disponible en el API Manager.

Primero vamos a crear una API nueva en el API Manager, podemos hacerlo con el botón “Add API” ubicado en la parte superior izquierda de dicha sección, le damos clic y luego seleccionamos “Add New API”, debemos seleccionar las siguientes opciones:

APIs / Add API

Runtime API Downstream Upstream Review

Runtime

Choose the runtime where your API instance will run on.

\* Required field

Select runtime

☐ Flex Gateway **NEW**  
 Ultrafast API gateway designed to manage and secure APIs running anywhere.

☒ Mule Gateway  
 API gateway embedded in Mule runtime. Connect directly to an existing Mule app or deploy a new proxy app.

☐ Service Mesh  
 Manage Kubernetes-based non-Mule microservices with Anypoint Service Mesh.

Proxy type \*

☒ Connect to existing application (basic endpoint)  
 Connect your API to a Mule application using Autodiscovery.

☐ Deploy a proxy application  
 Select a deployment target and deploy a new Mule application to serve as a proxy.

Mule version \*

☒ Mule 4 (recommended)

☐ Mule 3 or below

Una vez seleccionadas, damos clic a “Next”

Después se nos solicitara la API a utilizar, podemos usar una ya existente o crear una nueva, en mi caso utilizare una nueva:

APIs / Add API

Runtime API Downstream Upstream Review

API

Select the API you want to manage.

☐ Select API from Exchange

☒ Create new API

Once the API is created it will be published in Exchange in stable state.

Name

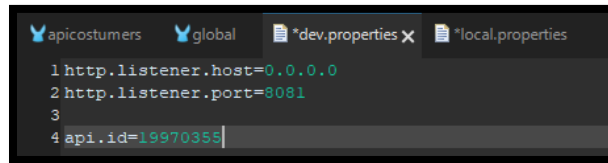
Asset types

Advanced >

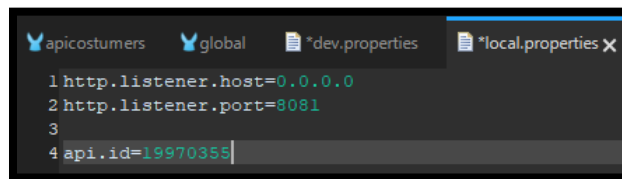
Dejamos todo por defecto, hasta que la plataforma nos de la opción “Save”.

Una vez terminado, observaremos información sobre la API recién creada, es importante guardar el “API Instance ID”, pues este nos servirá para vincular la app de Mule.

Volvamos ahora a nuestra app en Anypoint Studio, vamos a agregar como propiedad a los archivos de configuración dev y local de nuestro proyecto el ID obtenido de nuestra API, de la siguiente manera:

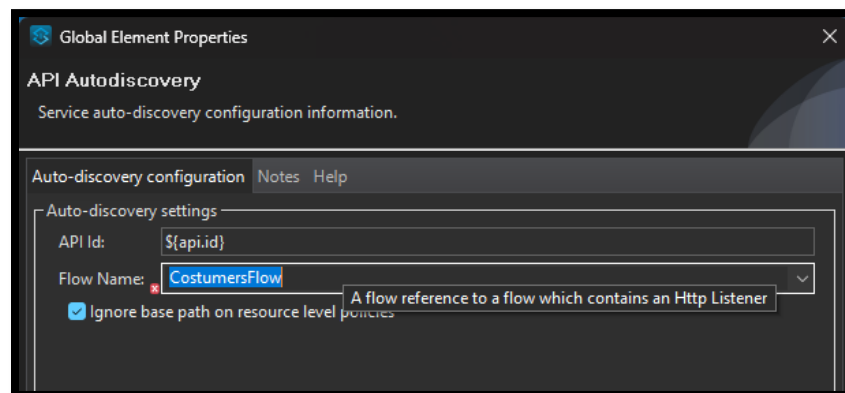


```
apicostumers  global  *dev.properties x  *local.properties
1 http.listener.host=0.0.0.0
2 http.listener.port=8081
3
4 api.id=19970355
```



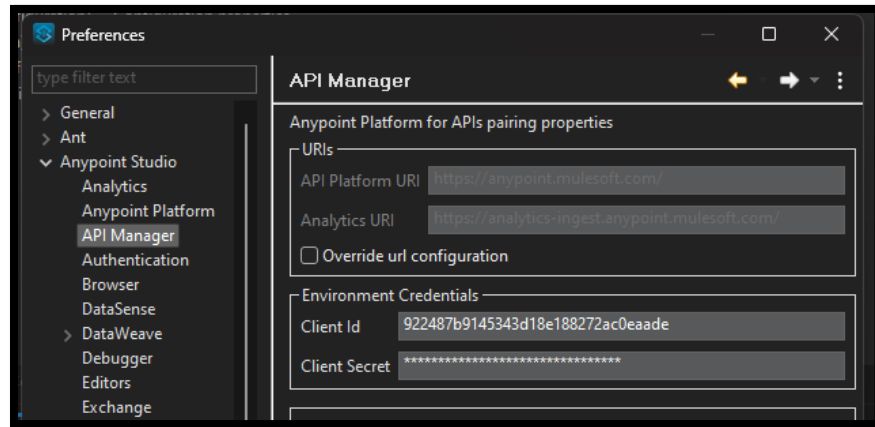
```
apicostumers  global  *dev.properties  *local.properties x
1 http.listener.host=0.0.0.0
2 http.listener.port=8081
3
4 api.id=19970355
```

Hora iremos a la vista de elementos globales del archivo global.xml y crearemos un “API Autodiscovery” con las siguientes propiedades (El id añadido mediante variable):



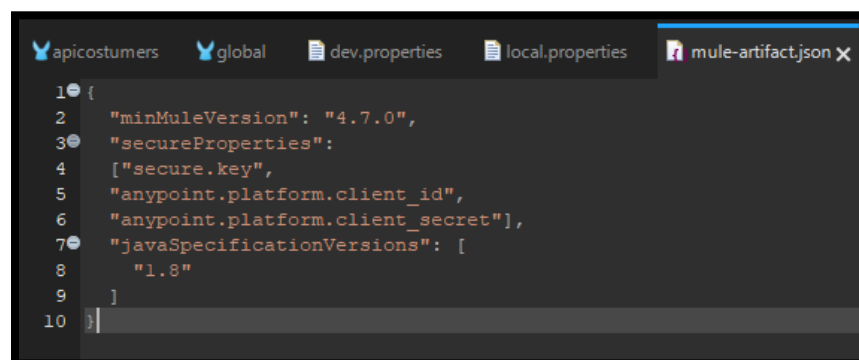
Ahora, debemos regresar a la plataforma de Anypoint y dirigirnos a la sección de API Manager y luego dar clic al botón “*Enviroment*”, allí se nos proporcionara un Client ID y un Client Secret, debemos añadir ambos como propiedades tanto de la implementación como de Anypoint Studio, empecemos con este primero.

Debemos ir a las preferencias del IDE, pueden buscarse como “*Preferences*” en la barra de búsqueda del IDE, luego irnos a “*Anypoint Studio*” y después a “*API Manager*”, allí debemos colocar las credenciales, de esta forma:

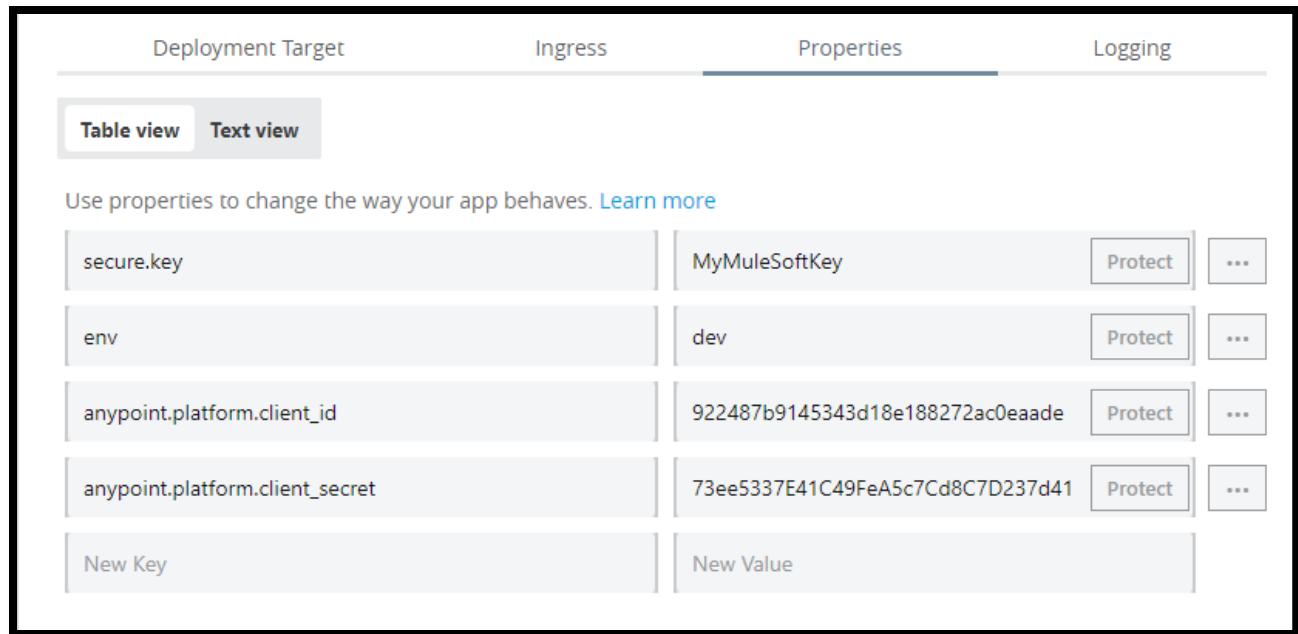


Aplicamos y guardamos cambios, después probamos nuestra aplicación localmente con Postman y si el estado es DEPLOYED, podemos continuar.

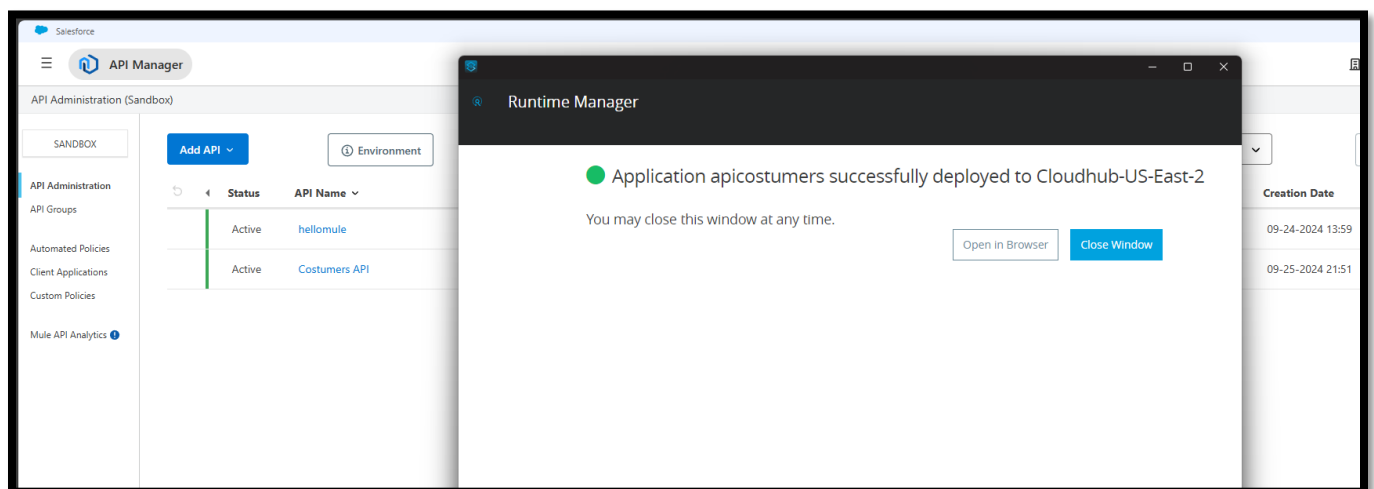
Vamos a añadir estas nuevas credenciales a las propiedades ocultas por seguridad, debemos hacerlo como antes con `secure.key`, abrimos el archivo `mule-artifact.json` y las añadimos, de la siguiente forma:



Guardamos cambios y procedemos a hacer Deploy, vamos a sobrescribir, pero antes, es importante que no se nos olvide añadir a las propiedades el Client ID y Secret Client, de la siguiente forma:



Y ahora sí, procedemos con el Deploy, si sale correctamente y el API aparece como activa en el API Manager de Anypoint Platform, entonces hemos tenido éxito.



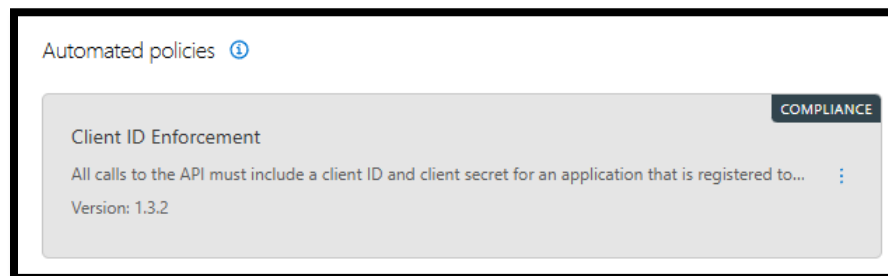
### 4.3-Protección de la API

Ahora veremos como aplicar una política de cumplimiento de ID de cliente para proteger nuestra aplicación con un proceso de autenticación básico.

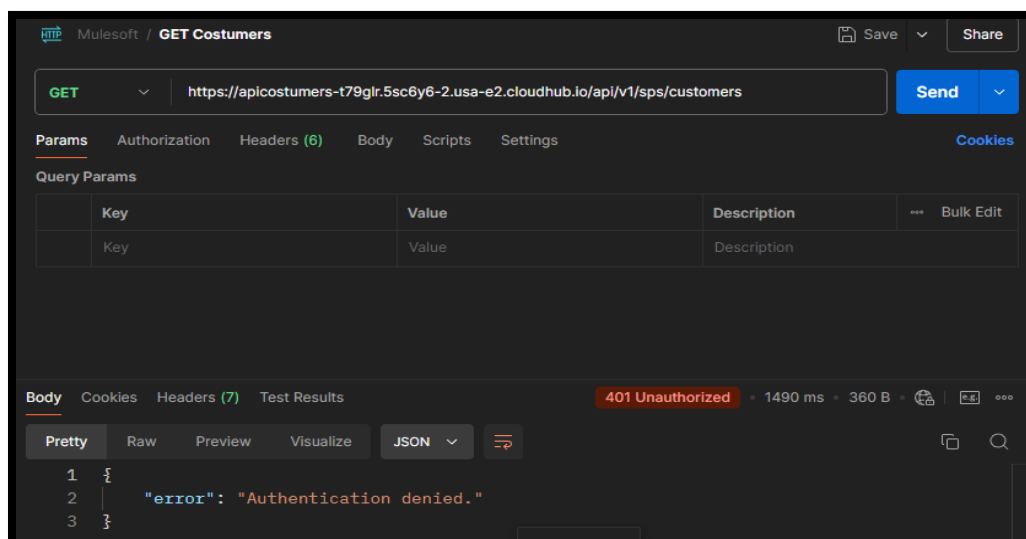
Lo primero que debemos hacer es ir a Anypoint Platform y ubicarnos en la sección API Manager, aquí debemos seleccionar la API a la que se le desea aplicar la política, vamos a utilizar la que esta vinculada a nuestra app de Mule, la que llamamos Costumers API.

Una vez allí, debemos ir a la sección “*Policies*”, ya dentro de esa sección debemos hacer clic en el botón “*Add policy*”.

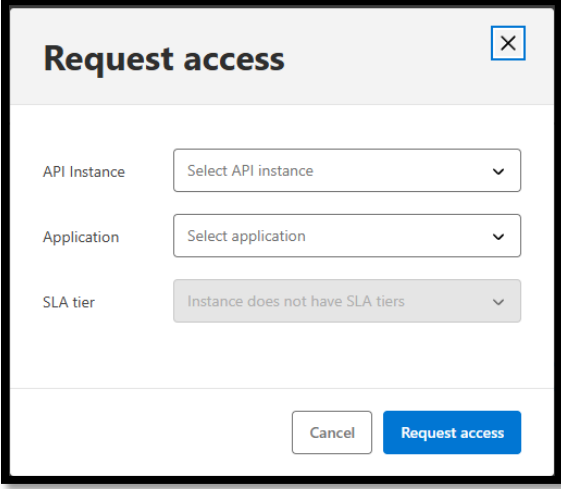
Debemos añadir la política “*Client ID Enforcement*”, en mi caso se añadió automáticamente, posiblemente debido a que la añadí en otra API de practica anteriormente:



Utiliza un encabezado de autenticación básica HTTP y se aplica a todos los métodos y recursos de la API, vamos a verificarlo con Postman, si hacemos una solicitud al método GET que creamos, debería rechazarnos.

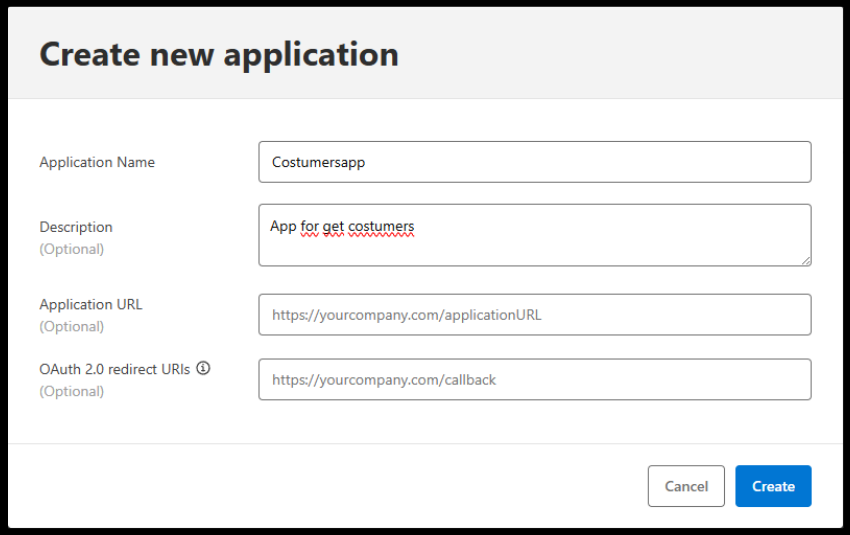


Ahora, para poder autorizar una solicitud, necesitamos generar unas credenciales de acceso de Client ID y Client Secret, para ello debemos ahora movernos a la sección Exchange de la plataforma y allí, seleccionar nuestra API, una vez allí debemos dar clic al botón “*Request Access*”, debe salir una ventana como esta:



The image shows a modal dialog box titled "Request access" with a close button (X) in the top right corner. Inside the dialog, there are three dropdown menus: "API Instance" with the placeholder text "Select API instance", "Application" with the placeholder text "Select application", and "SLA tier" with the placeholder text "Instance does not have SLA tiers". At the bottom right of the dialog, there are two buttons: a "Cancel" button and a blue "Request access" button.

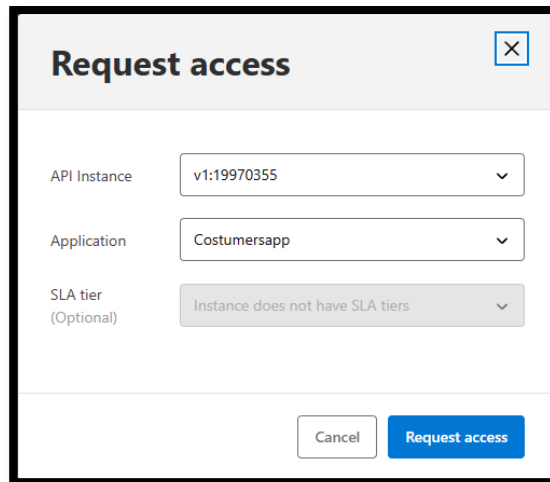
En API Instance debemos seleccionar la instancia de nuestra API (Única opción disponible) y en Application, debemos seleccionar “*Create a new application*”, en seguida, llenamos los campos de la siguiente manera y damos clic a “*Create*”:



The image shows a form titled "Create new application". It contains four input fields: "Application Name" with the value "Costumersapp", "Description (Optional)" with the value "App for get costumers" (which has a red squiggly underline), "Application URL (Optional)" with the value "https://yourcompany.com/applicationURL", and "OAuth 2.0 redirect URIs (Optional)" with the value "https://yourcompany.com/callback". At the bottom right, there are two buttons: a "Cancel" button and a blue "Create" button.



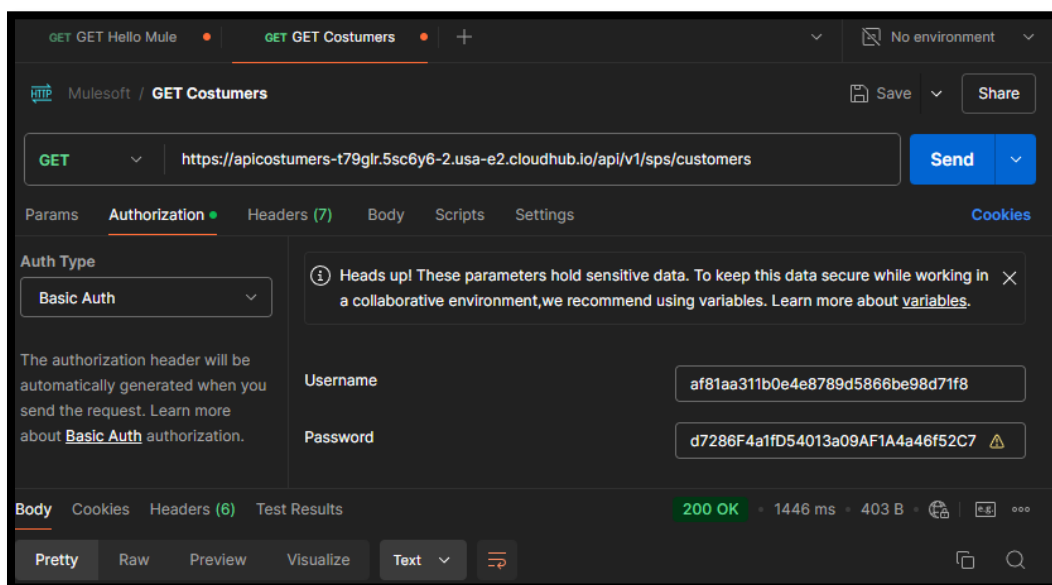
Seleccionamos la aplicación recién creada y damos clic a “Request Access”:



A dialog box titled "Request access" with a close button (X) in the top right corner. It contains three dropdown menus: "API Instance" with the value "v1:19970355", "Application" with the value "Costumersapp", and "SLA tier (Optional)" with the value "Instance does not have SLA tiers". At the bottom, there are two buttons: "Cancel" and "Request access".

Una vez hecho nos otorgara las credenciales de acceso, las guardamos (aunque podemos consultarlas en esta misma página si lo requerimos).

Ahora, ya solo nos queda volver a realizar una solicitud con Postman, esta vez, utilizando las credenciales de acceso que solicitamos, para pasar las credenciales a Postman, en el apartado de params, vamos a la pestaña “Authorization”, en Auth Type seleccionamos “Basic Auth” y colocamos nuestro Client ID en el campo Username y el Client Secret en Password, luego mandamos la solicitud, si lo hicimos correctamente la solicitud debería ser aprobada:



Con ello hemos terminado.

## 5.-Retos y problemas presentados

### 5.1-Documentación de MuleSoft

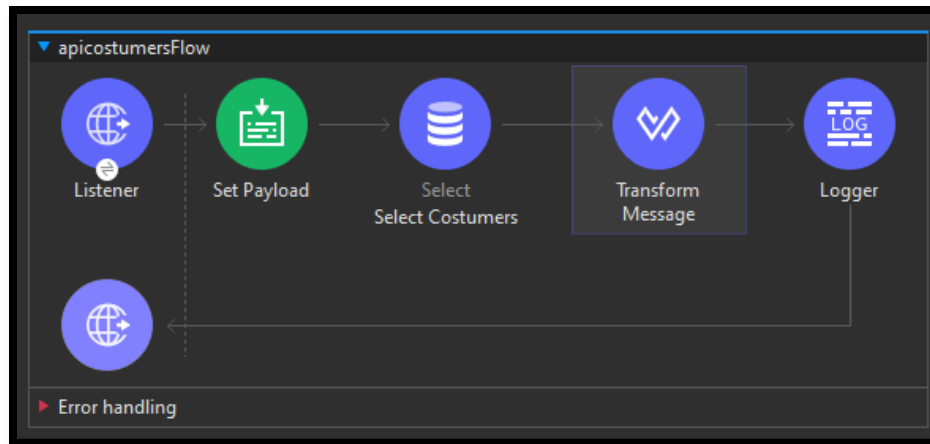
El primer problema que se me presentó fue con la documentación de MuleSoft, si bien esta está bien explicada, en algunos puntos se encuentra desactualizada, mencionare de forma muy resumida algunos de estos puntos:

- El comando para encriptar valores usando el archivo JAR Secure Properties Tool no está actualizado, la sintaxis correcta no lleva “/” para separar los parámetros, la sintaxis correcta afortunadamente la ofrece la propia terminal, aunque también pude corroborar esto en un comentario del videotutorial que se encuentra en la documentación.
- En los tutoriales de los puntos extra como la protección de la API, las interfaces, nombres y ubicaciones de algunos componentes se encuentran distintas, algunas más que otras, por ejemplo, en el caso de protección de la API, el componente que se menciona para añadir la protección tiene un nombre diferente y está ubicado en un sitio diferente al mostrado en la documentación.
- Por último, aunque la documentación fue muy útil se quedó corta para realizar algunas tareas como la especificación de la API, ya que la información que viene en la documentación es teórica y no práctica, por lo que encontré algo de información para desarrollar este punto por mi cuenta

### 5.2-Consulta a la Base de datos

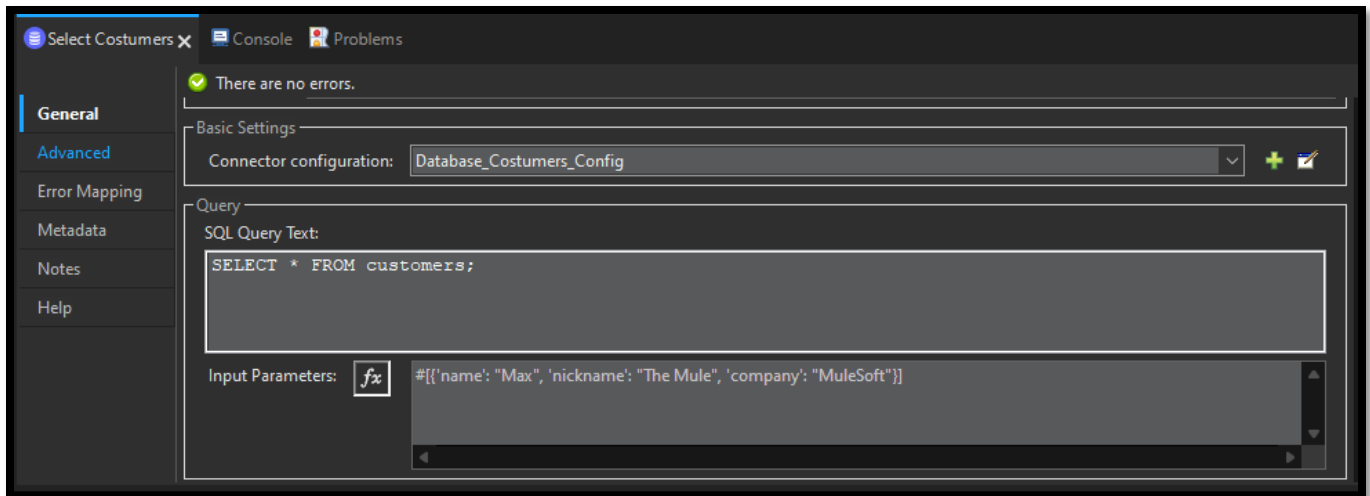
Este es otro tema que no se toca en la documentación enviada y del que encontré algo de información, pero no la suficiente, antes de explicar mi problema, diré de forma breve la información que encontré.

Para conectar a una base de datos existente con nuestra API desde Anypoint Studio, es necesario añadir dos módulos a nuestro flujo, “*Database Select*” y “*Transform Message*”, el primer módulo permite definir los parámetros de la conexión y el segundo, transformar una respuesta en otra antes de que esta llegue al cliente, aquí dejo el flujo creado:



En cuanto a los parámetros de conexión, como mencione estos se definen en las propiedades del módulo Database Select, aquí están los valores utilizados (sacados del documento proporcionado por SPS):

The screenshot shows the 'Global Element Properties' dialog box for a 'Database Config' element. The 'General' tab is selected. The 'Name' field is 'Database\_Costumers\_Config'. The 'Connection' dropdown is set to 'MySQL Connection'. Below this, there is a section for 'Required Libraries' with a checked box for 'MySQL JDBC Driver (mysql:mysql-connector-java:8.0.30)'. The 'Connection' section contains fields for 'Host' (mudb.learn.mulesoft.com), 'Port' (3306), 'User' (mule), 'Password' (masked with dots), and 'Database' (training). A 'Show password' checkbox is next to the password field. At the bottom, there are buttons for 'Test Connection...', 'OK', and 'Cancel'.



El problema comenzó con la transformación de los datos, si bien averigüé que era posible hacerlo mediante el módulo Transform Message, no encontré la información suficiente para poder realizar el proceso, por lo que este se quedó desarrollado a medias.

### 5.3-Bloopers

Ahora hablare un poco de las “metidas de pata” que tuve durante el desarrollo de la aplicación.

En general, hubo muy pocos errores pequeños o que pudiera considerar “un clásico” como los errores de sintaxis, referencias incorrectas o procesos cuyo resultado no era el esperado, los únicos que tuve fue al manipular los archivos XML, de al intentar mover módulos de archivos de configuración, no colocarlos de forma adecuada y generar un error por la apertura y cierre de etiquetas.

Sin embargo, si tuve un problema más serio al cual no le encontré una solución por completo, este se derivó a raíz de intentar implementar la comunicación con la base de datos, en algún punto uno de los movimientos que hice corrompió el archivo XML principal del proyecto, afortunadamente luego de darme cuenta del origen del problema, pude solucionarlo quitando un espacio que había en el archivo, sin embargo, aunque el archivo ya se visualizaba correctamente por el IDE y la aplicación se podía desplegar, **esta no respondía a mis solicitudes, al menos no de forma local, curiosamente desplegada funciona perfectamente, es posible que el problema se encuentre en mi sistema operativo, (considere dicho punto al ejecutar el proyecto en su equipo).**

## 6.-Conclusiones

Mulesoft con su plataforma Anypoint Platform y su IDE Anypoint Studio ofrece todo un sistema para el desarrollo de API's bastante completo, considero que es competente para ofrecer soluciones que cumplan con necesidades no demasiado particulares y que puede ser una buena opción para iniciarse en el desarrollo de API's, diría que su punto fuerte es la accesibilidad que ofrece y la forma en que integra lo necesario para desarrollar una API e implementarla en una sola plataforma.

Como punto negativo, diría que cuenta con poca información comparado a otras soluciones o frameworks, la documentación se encuentra desactualizada, en poca variedad de idiomas y el contenido generado por la comunidad se encuentra en la misma situación, además, si bien la simplificación de procesos puede ser positiva sobre todo en proyectos no muy ambiciosos, en proyectos donde se requiere tener mayor control de cada aspecto herramientas como estas pueden resultar una mala opción.

Al final y como es común en el mundo de la tecnología y el software, considero que no hay herramientas superiores o inferiores, simplemente hay herramientas que se adaptan mejor o peor a unas necesidades en particular y creo que esta es una de esas tantas herramientas que en ciertos casos, puede ser una gran opción para desarrollar API's.

Considero que la realización de la practica estuvo mas cerca de ser un éxito que un fracaso, casi todos los puntos fueron desarrollados de forma exitosa, conseguí familiarizarme ligeramente con la plataforma y su IDE, entendí el porque de algunos procedimientos y en otros me quede con duda, pero considerando que intente abarcar todos los puntos incluidos los opcionales, considero que fue tiempo bien invertido.