

PRÁCTICA TÉCNICA

Para la posición de Desarrollador
MuleSoft

ISRAEL JOSAFAT PEREZ CAZARES

soy.josa.perez@gmail.com

Contenido

Introducción.	1
Solución.	1
Diseño de Especificación de una API.	1
Desarrollo.	6
Pruebas.	13

Introducción.

Este documento ha sido redactado para explicar el paso a paso de la solución propuesta por el aspirante a la problemática planteada, utilizando la tecnología MuleSoft y todas sus herramientas. Tomando en cuenta la problemática, se decidió explicar la solución en dos partes, el diseño de la especificación de la API y el desarrollo.

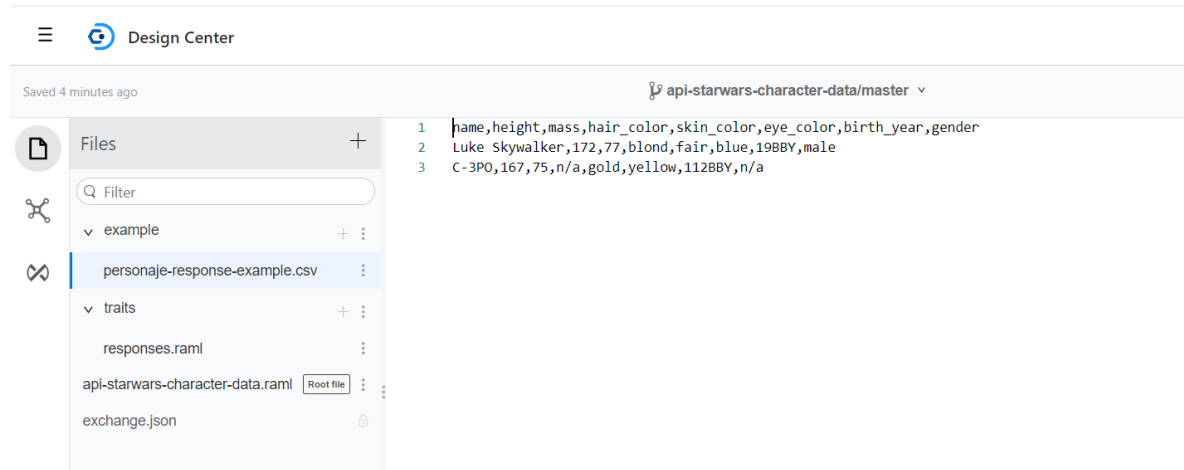
Solución.

Diseño de Especificación de una API.

La API diseñada tiene como objetivo un solo propósito que es obtener de SWAPI información sobre los personajes del universo cinematográfico de Starwars, en este caso para el diseño de sus especificaciones se utilizó la herramienta Design Center de Anypoint Plataform, donde se trabajó con RAML para la definición de los endpoints, de las diferentes respuestas que estos generarían, y además de agregar una capa de seguridad, como se muestra en la siguiente imagen.

```
1  #%RAML 1.0
2  title: api-starwars-character-data
3  description: API Starwars gets data of the charateres of Starwars from SWAPI CSV format
4  version: 1.0
5
6  traits:
7    client-id-required:
8      headers:
9        client_id:
10         type: string
11        client_secret:
12         type: string
13      responses: !include traits/responses.raml
14
15  types:
16    personajeRespons:
17      type: string
18      example: !include example/personaje-response-example.csv
19
20  /personajes:
21    is: [client-id-required]
22    get:
23      queryParameters:
24        genero:
25          type: string
26          required: false
27          displayName: Genero del personaje
28          description: Genero por el cual se desea filtrar los datos
29      responses:
30        200:
31          body:
32            application/csv:
33              type: personajeRespons
34      is: [responses]
```

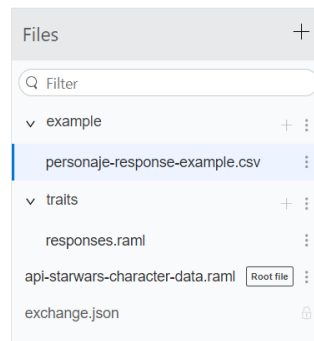
Para esta API se creó solo un endpoint /personajes con el método GET, el cual regresa todos los personajes que contiene SWAPI, y se agregó un query parameter opcional, "genero" que mostrará los personajes del género seleccionado. La respuesta se definió como tipo string, y el ejemplo se definió de la siguiente forma.



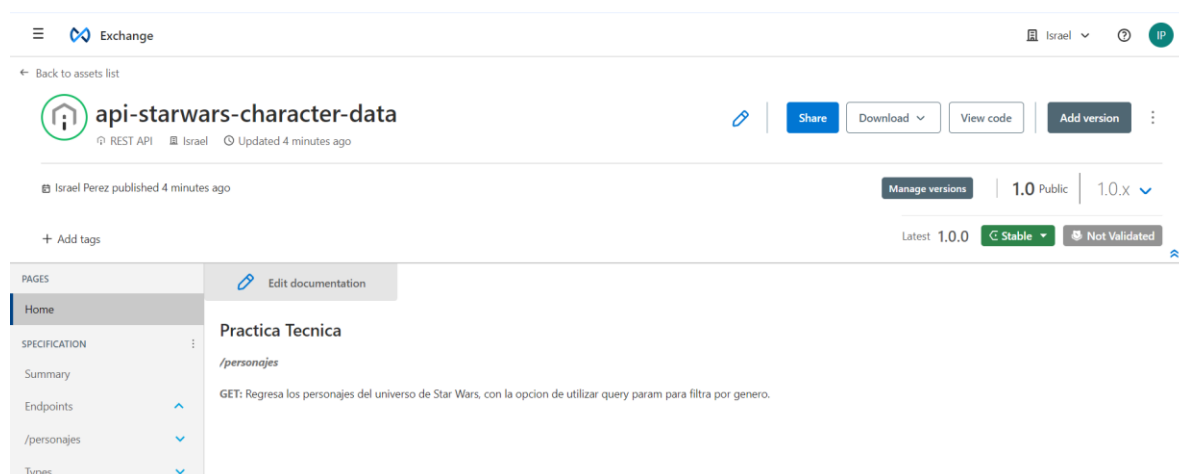
También se declararon respuestas de error por si la API llega a tener un erro, estas se definieron como traits y se importaron, los mensajes de error que se utilizaron fueron los aquellos que tiene una mayor posibilidad de que sucedan.



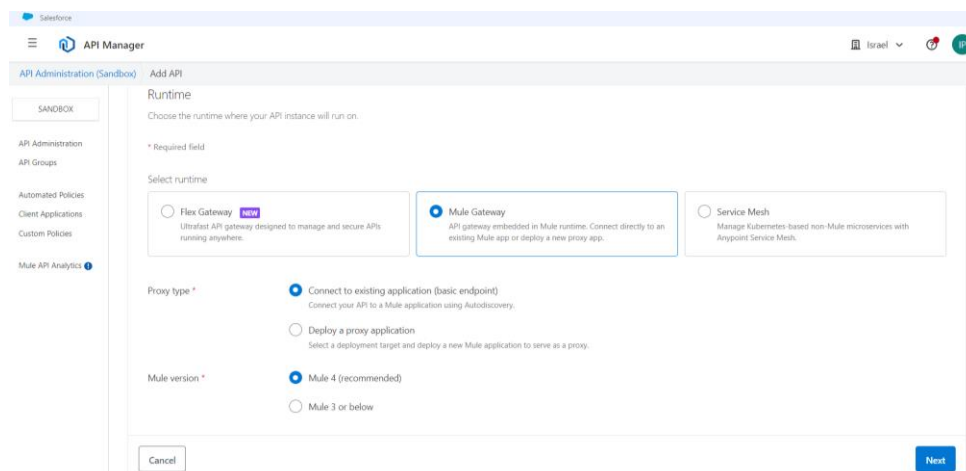
También cabe resaltar que los ejemplos y respuestas se importan desde archivos externos, como se puede ver en la imagen, estas rutas forman parte del mismo proyecto.



Después de terminar el diseño se procedió a publicar la API en Exchange, siendo la versión 1.0.0 estable. En Exchange, por el momento solo se buscará que la API sea publica y opcionalmente se puede agregar un poco de información acerca de la API, como se muestra en la imagen



Después de esto nos dirigimos a API Manager, donde en este caso agregaremos la nueva API, seleccionando Mule Gateway como runtime.



Y a continuación seleccionamos la API desde Exchange, al igual que la versión y el tipo de especificación, como se muestra en las siguientes imágenes.

The first screenshot shows the 'Add API' page in Salesforce API Manager. It has a sidebar with navigation links: SANDBOX, API Administration, API Groups, Automated Policies, Client Applications, Custom Policies, and Mule API Analytics. The main area has a 'Select the API you want to manage' section with a 'Select API from Exchange' button and a 'Create new API' radio button. Below is a 'Select API' section with a search bar and a list of APIs. The second screenshot shows the same page after selecting the 'api-starwars-character-data' API. The 'Selected API' section shows the API name and a 'View in Exchange' link. Below are dropdown menus for 'Asset type' (RAML/OAS), 'API version' (1.0 (Latest)), and 'Asset version' (1.0.0 (Latest)). There is also a 'Conformance Status' section showing 'Not Validated' and a 'Cancel' button at the bottom left, and 'Previous' and 'Next' buttons at the bottom right.

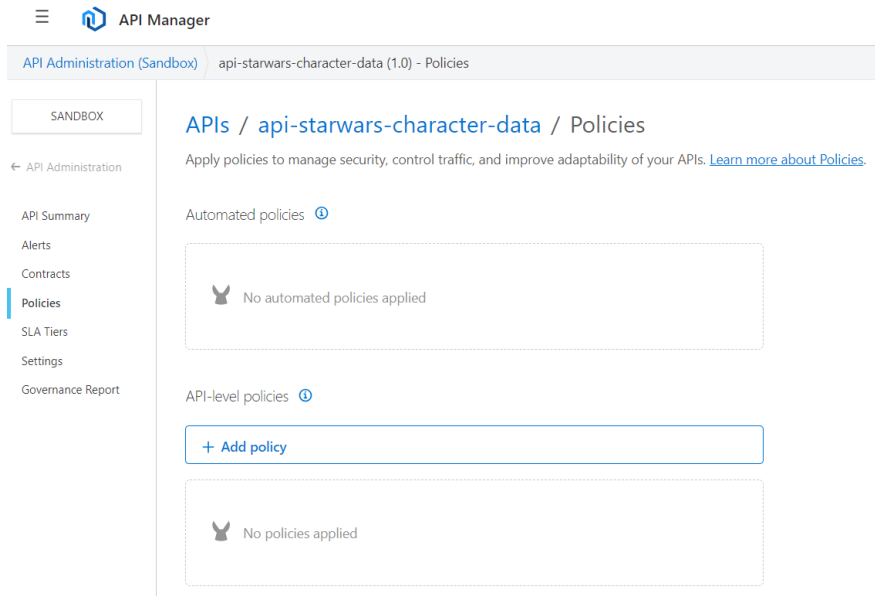
Para después concluir con el registro de esta API en API Manager y obtener datos importantes como API ID, que nos servirá más adelante.

The screenshot shows the 'API Summary' page for the 'api-starwars-character-data (1.0)' API. The page has a sidebar with navigation links: SANDBOX, API Administration, Alerts, Contracts, Policies, SLA Tiers, Settings, and Governance Report. The main area has a header 'APIs / api-starwars-character-data / API Summary' and an 'Actions' dropdown. Below the header is a blue box with a message: 'To complete the registration process, you need to connect this API to your Mule application using Autodiscovery. Learn more'. Below this is a table with the following data:

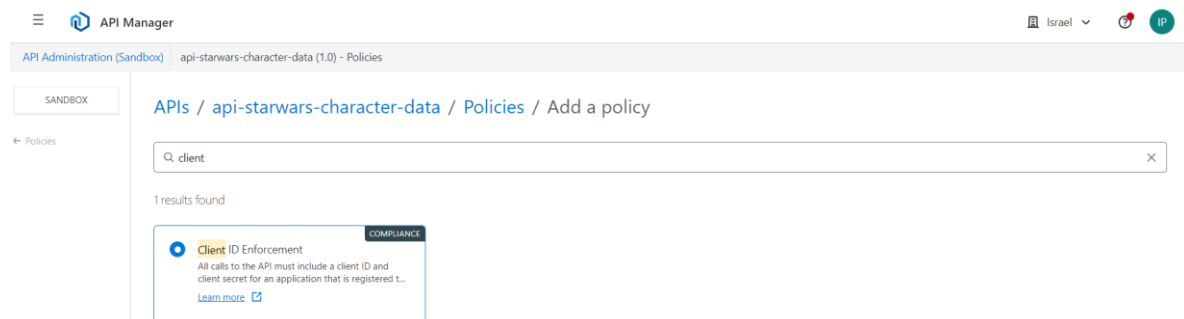
Type	Asset Version	Implementation URI ⓘ	API Label ⓘ
RAML/OAS	1.0.0 (Latest)	N/A	-
API Version	API Status	Consumer Endpoint	API Instance ID ⓘ
1.0	Unregistered	N/A	19898520

Below the table is a section for 'Instance Conformance ⓘ' showing 'Not Validated'. At the bottom is a 'Tags' section with an 'Add New Tag +' button.

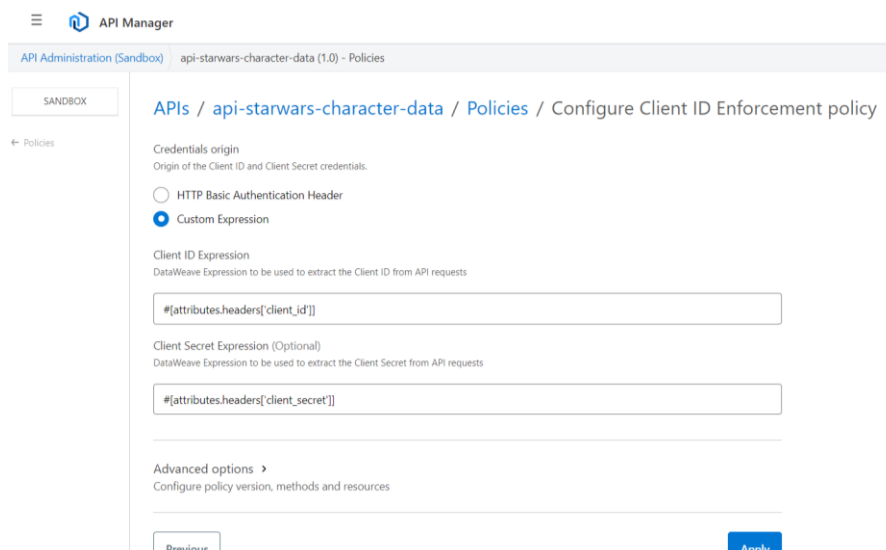
En esta parte es donde podemos agregar la capa de seguridad que anteriormente se menciona, en la pestaña de "Policies", ahí seleccionamos "Add Policy".



Y buscamos la policy "Client ID Enforcement".



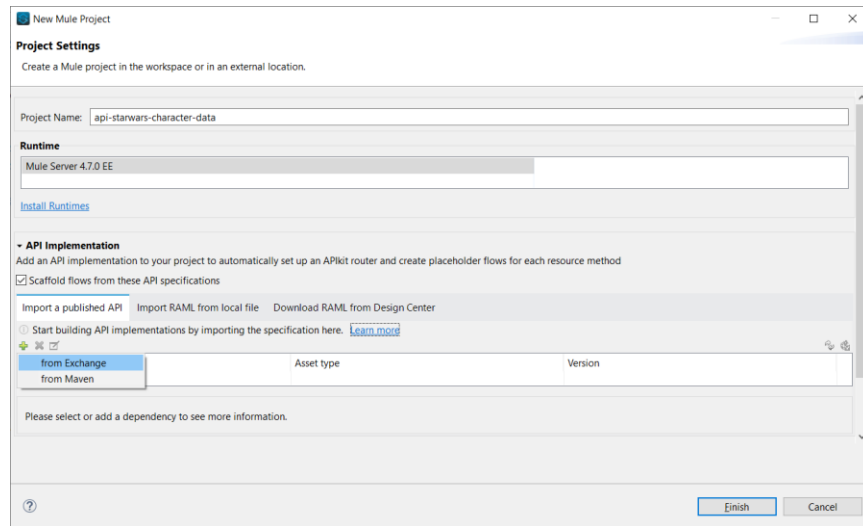
Al dar siguiente nos mostrara los headers que serán requeridos cada vez que hagamos un request a nuestra API, y esto es por lo que al momento de diseñar la API se definen esos headers.



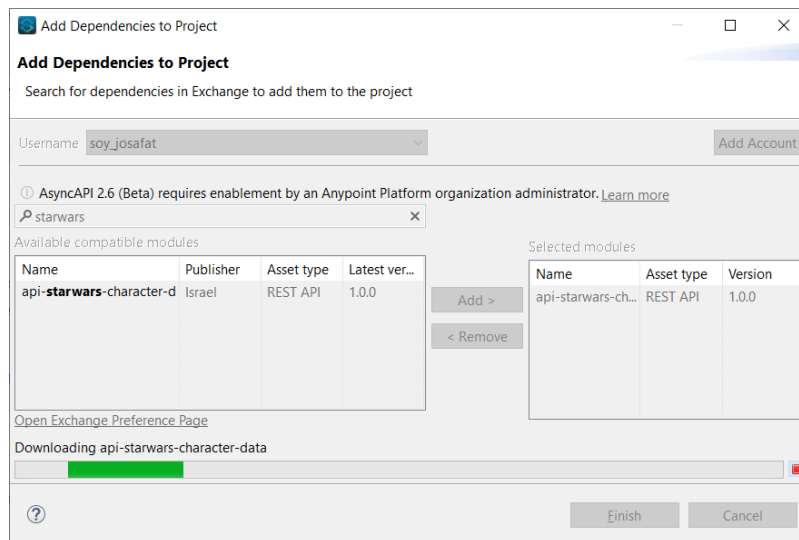
Desarrollo.

Para la implementación de la API se utilizó la herramienta Anypoint Studio y sus diferentes componentes, y para el despliegue las herramientas proporcionadas por Anypoint Plataforma, como CloudHub. Se realizó los últimos pasos de configuración para que la API pueda recibir request.

Previamente se prepara el entorno necesario para poder trabajar con Anypoint Studio. Al tener esto listo, se importa el proyecto de las especificaciones desde Exchange como se muestra en la imagen.

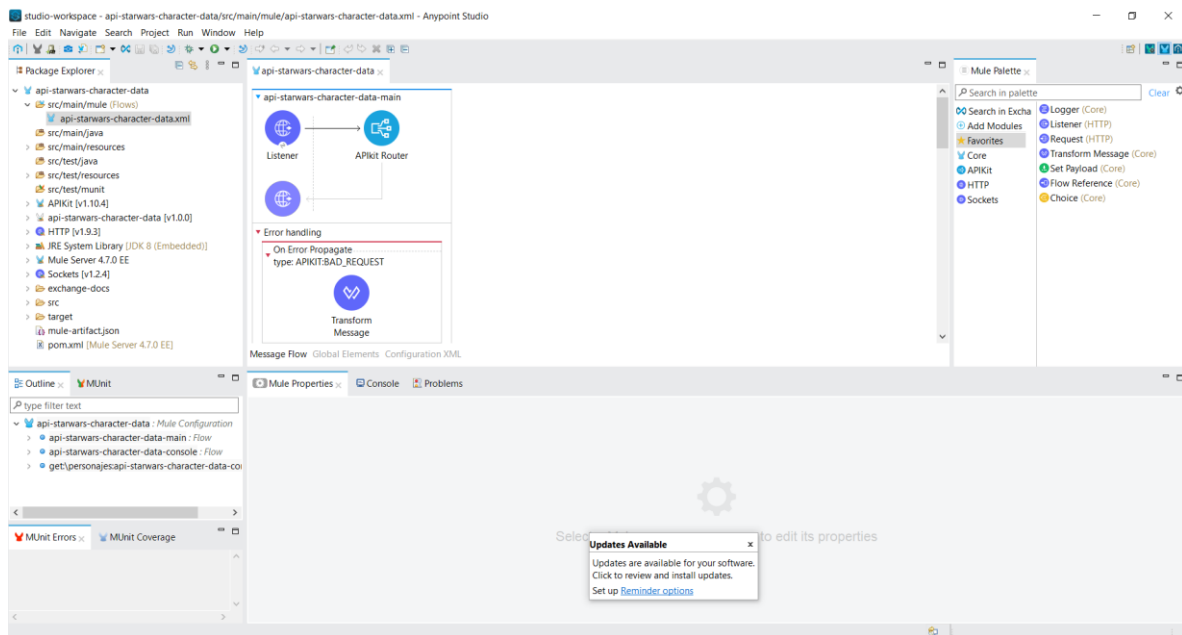


Se busca y selecciona el proyecto deseado, en este caso “api-starwars-character-data”

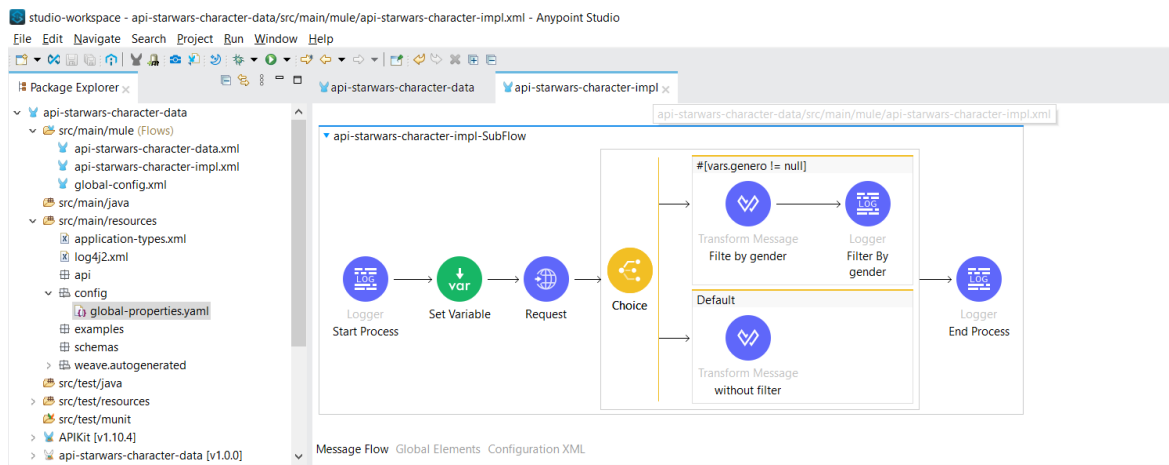


Después de que todo cargue se termina el proceso y se puede empezar a trabajar en el proyecto.

Después de importar el proyecto de Exchange, se genera un proyecto con APIKit Router que nos ayudara a enrutar todas las request que la API reciba al endpoint que se definió en el diseño.



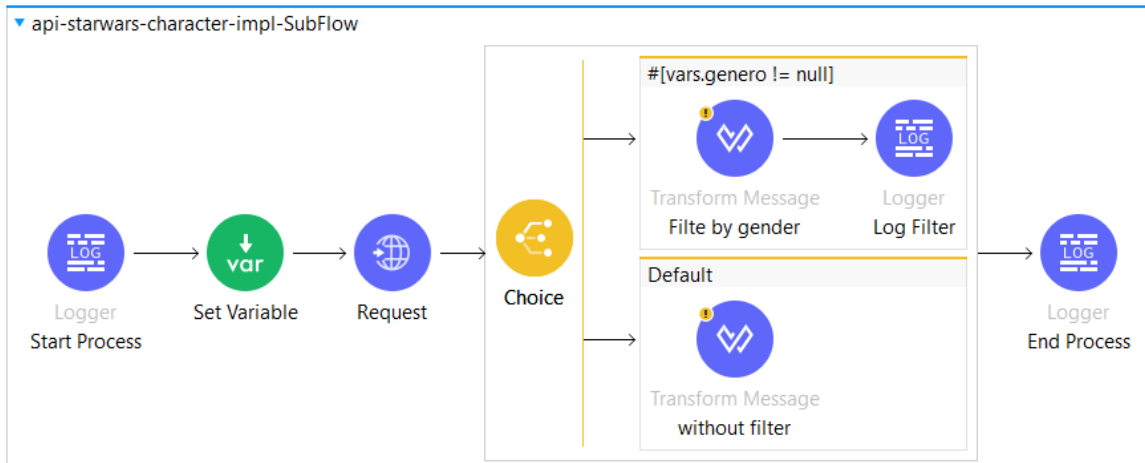
Para la creación de la implementación de la API se creo un nuevo archivo, donde se crearon todos los flujos necesarios para que la API haga lo deseado.



Todas las configuraciones de request y listeners o alguna otra configuración global, se realizan en otro archivo llamado global-config.xml.

La solución que se implemento fue simple obtener la información y procesarla, dependiendo si se necesitaba filtrar por genero o no, donde también se se convertía a formato CSV.

Para esto se creó el siguiente flujo:



Donde con ayuda del componente "Set Variable" se extrae el query param de la llamada que recibe la API guardando la en la variable "genero", para poder utilizarlo más adelante, después se hace un Request a SWAPI con el siguiente url: <https://swapi.dev/api/people/> para obtener la información de los personajes.

Después con ayuda de Choice y la variable "genero", se verifica si el request que recibió algo en la query, si esto es verdad se procede a un Transform Message donde se utilizó Data Weave para mapear los datos filtrando solo aquellos datos que se solicitaron y además utilizando la función "filter" para realizar el filtro por género, como se muestra en la imagen.

```

Output Payload  ▾  [Icons]  Preview
1 %dw 2.0
2
3 var list = payload.results map ((item, index) -> {
4   name: item.name,
5   height: item.heught,
6   mass: item.mass,
7   hair_color: item.hair_color,
8   skin_color: item.skin_color,
9   eye_color: item.eye_color,
10  birth_year: item.birth_year,
11  gender: item.gender
12 })
13
14 output application/csv quoteValues=true, header=true
15
16 ---
17
18 list filter ((item, index) -> item.gender == vars.genero )

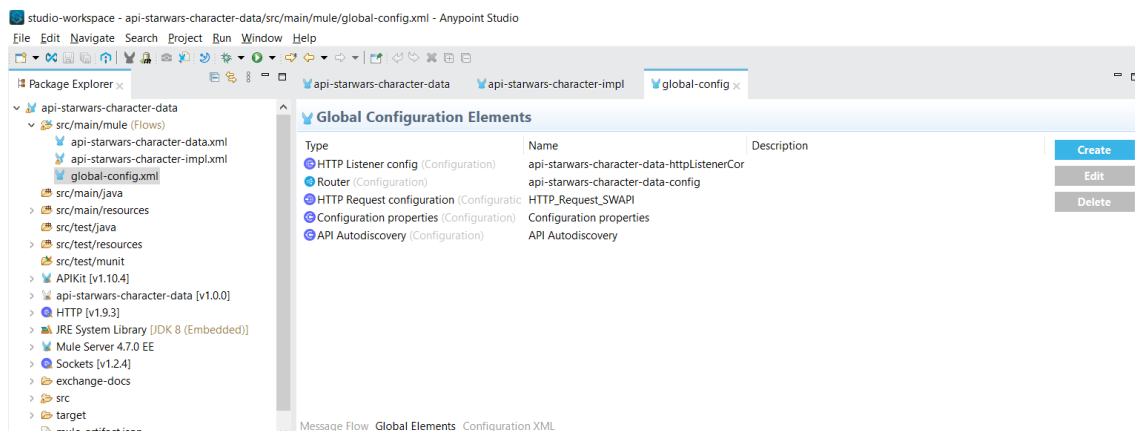
```

En caso contrario en que el query no contuviera nada, este se dirigiría a la opción por defecto donde con ayuda de un Transform Message la información solo se mapearía a un formato CSV y solo se obtendrían los datos solicitados.

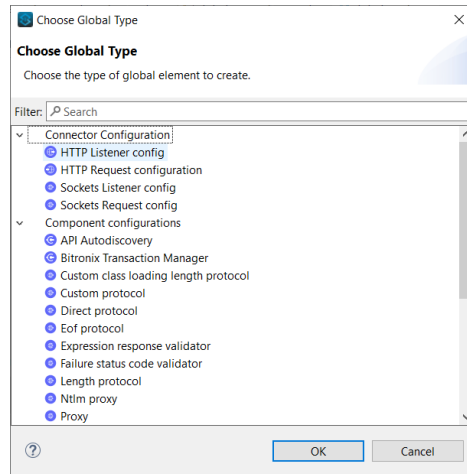
```
Output Payload ▾ ↗ ✎ 🗑 Preview
1 10%dw 2.0
2
3 output application/csv quoteValues=true, header=true
4 ---
5 payload.results map ((item, index) -> {
6   name: item.name,
7   height: item.heught,
8   mass: item.mass,
9   hair_color: item.hair_color,
10  skin_color: item.skin_color,
11  eye_color: item.eye_color,
12  birth_year: item.birth_year,
13  gender: item.gender
14 })
```

Los componentes Logger nos ayudan a registra mediante logs cómo funciona la ejecución de esta API, y estos son solo informativos.

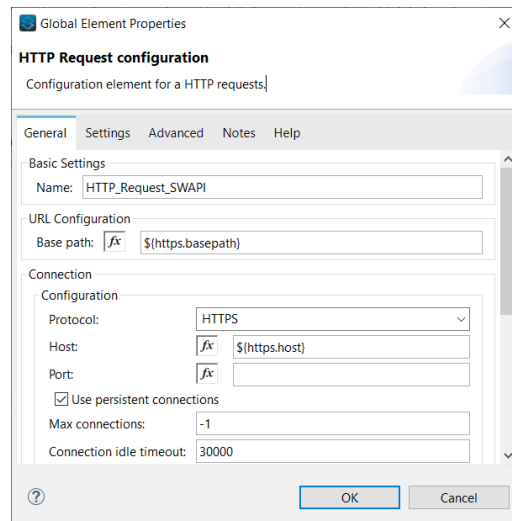
El archivo global-config.xml antes mencionado se muestra en siguiente la imagen, y como se mencionó en este se configuraron todas las propiedades de configuración de ciertos elementos.



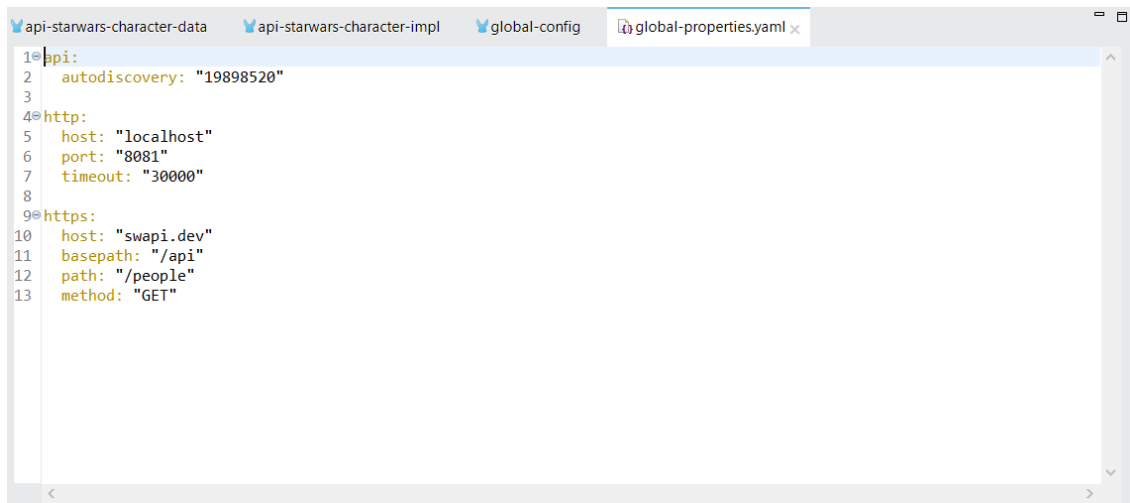
Iniciamos con la creación del HTTP Request, para esto se dio click en el boton crear el cual nos arroja una ventana en la cual se pueden buscar todas las configuraciones que se pueden crear en Anypoint Studio.



En este caso seleccionamos HTTP Request Configuration, la cual nos arrojará una ventana donde agregaremos las configuraciones necesarias para poder llamar a SWAPI.

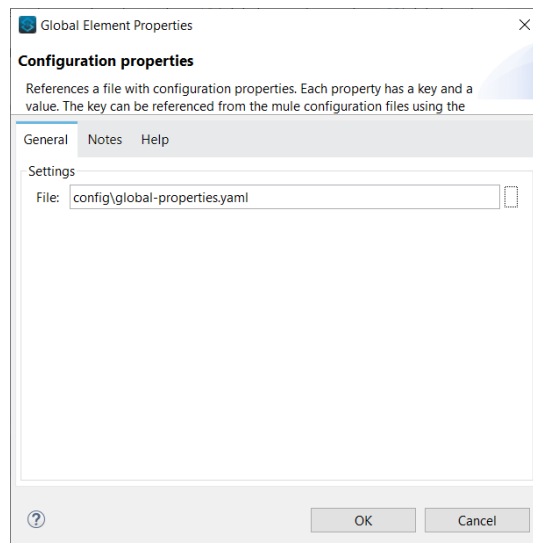


Para evitar la codificación rígida se utilizó la anotación que se ve en la imagen la cual llama los datos desde un archivo de configuración. Este archivo contiene todas las propiedades de configuración de cada uno de los elementos globales como se puede apreciar en la imagen.

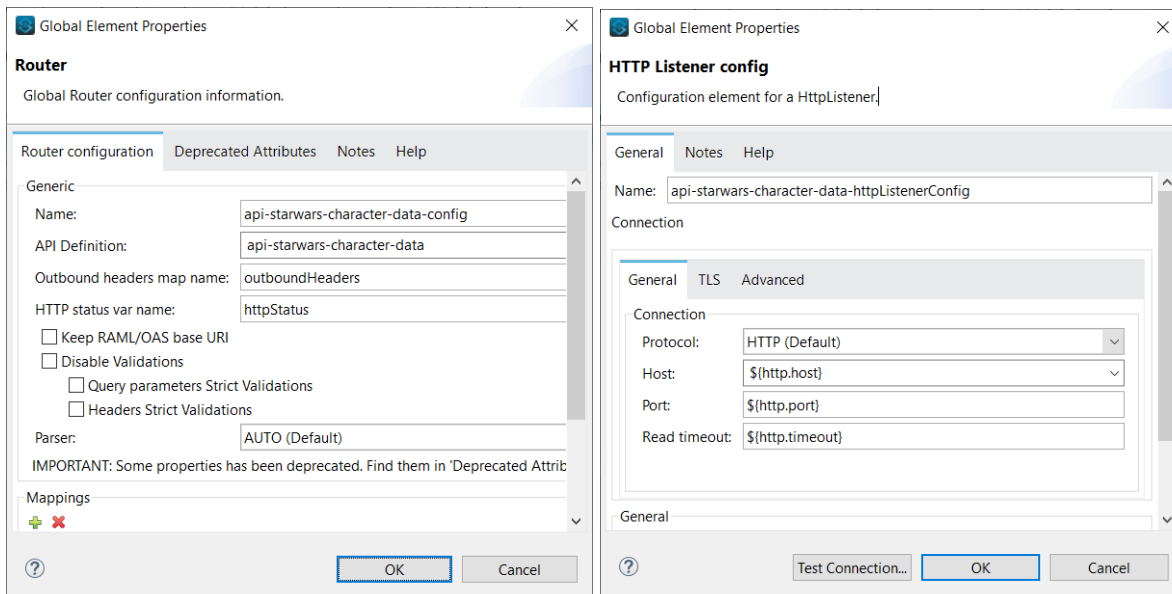


```
1 api:
2   autodiscovery: "19898520"
3
4 http:
5   host: "localhost"
6   port: "8081"
7   timeout: "30000"
8
9 https:
10  host: "swapi.dev"
11  basepath: "/api"
12  path: "/people"
13  method: "GET"
```

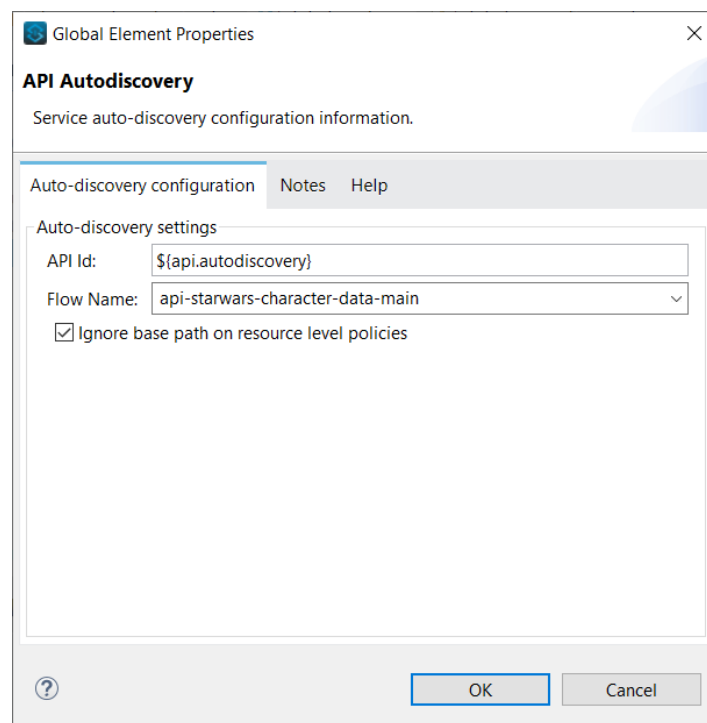
Para que mule sepa que esas propiedades se tienen que tomar de ahí, se tiene que configurar, medite una configuración global llamada "Configuration properties", normalmente también se evita tener un path rígido de este archivo, ya que en diferentes ambientes las propiedades podrían cambiar, pero por la simpleza de este proyecto se decidió dejarlo así.



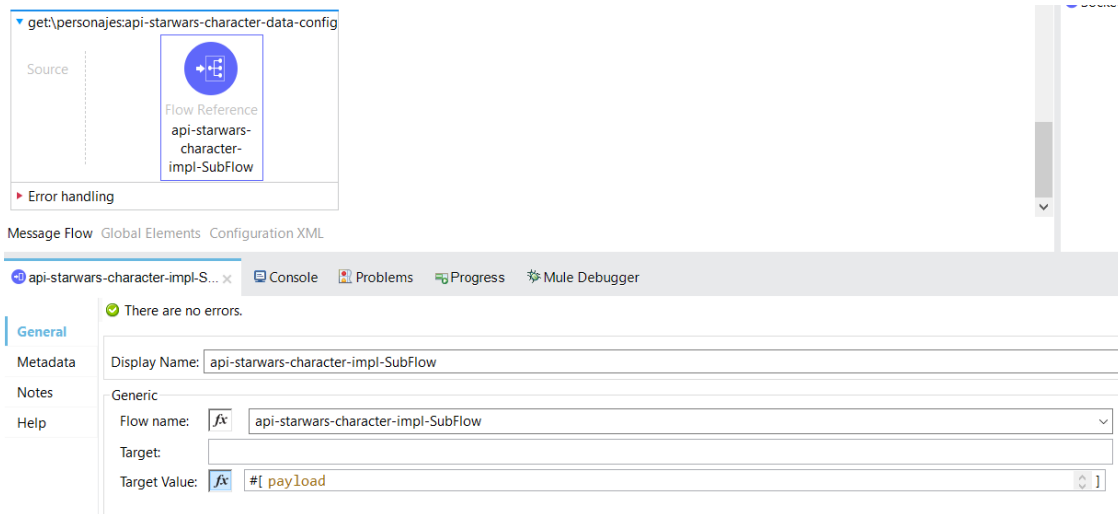
La configuración del APIkit Router, se duplica en este archivo, así como la del HTTP Listener que se generó automáticamente en la creación del proyecto, y siguiendo la misma línea se evita una codificación rígida en todo elemento que se pueda, como se puede ver en las imágenes siguientes.



También se configura el API Autodescovery, que este nos ayudara mas a delante cuando la aplicación sea lanzada a la nube. Cabe que aclar que esto se realizo antes de las pruebas locales.

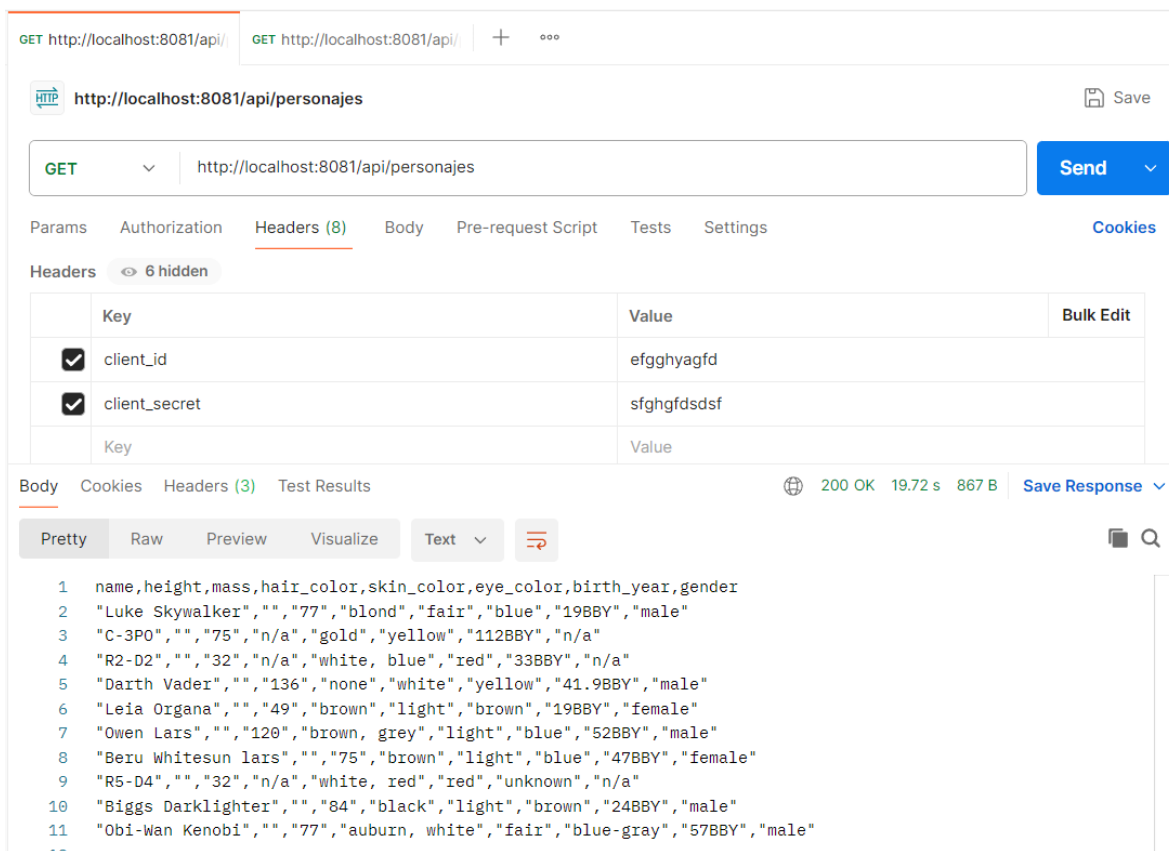


Para que todo este funcione tenemos asegurarnos de que el código que se genero al crear el proyecto del path /personajes este referenciado a la implementación, y esto se logra agregando un componente llamado Flow Reference que lo único que hará será enrutar o llamar al Flow de implementación creado. Esto se vería como se muestra en la imagen.



Pruebas.

Para probar el funcionamiento de esta API de manera local utilizaremos Postman, en el cual crearemos un request con la URL <http://localhost:8081/api/personajes>, agregaremos los headers `client_id` y `client_secret`, en este caso su valor no importa ya que la implementación de la policy no se completó.



En la imagen se puede ver la respuesta que se obtuvo del request, y es la esperada, los datos en formato CSV.

La siguiente prueba sería verificar el funcionamiento del query parameter, en este caso solo se tendría que agregar la URL, quedando así, <http://localhost:8081/api/personajes?genero=male>, en la respuesta podemos ver que realmente funciona y retorna solo de los datos de aquellos que tienen el género deseado.

The screenshot displays a REST client interface with two main sections. The top section shows a GET request to `http://localhost:8081/api/personajes?genero=male`. The 'Headers' tab is active, showing a table with headers `client_id` and `client_secret`. The bottom section shows the response body in 'Pretty' format, displaying a CSV of Star Wars characters filtered by gender.

Request Headers:

Key	Value
<input checked="" type="checkbox"/> client_id	efgghyagfd
<input checked="" type="checkbox"/> client_secret	sfghgfdsd

Response Body (Pretty):

```
1 name,height,mass,hair_color,skin_color,eye_color,birth_year,gender
2 "Luke Skywalker","",77,"blond","fair","blue","19BBY","male"
3 "Darth Vader","",136,"none","white","yellow","41.9BBY","male"
4 "Owen Lars","",120,"brown, grey","light","blue","52BBY","male"
5 "Biggs Darklighter","",84,"black","light","brown","24BBY","male"
6 "Obi-Wan Kenobi","",77,"auburn, white","fair","blue-gray","57BBY","male"
7
```

Y es así como se realiza esta práctica, espero haya sido claro y conciso en la explicación del paso a paso.