Mulesoft Jr / Mid Developer

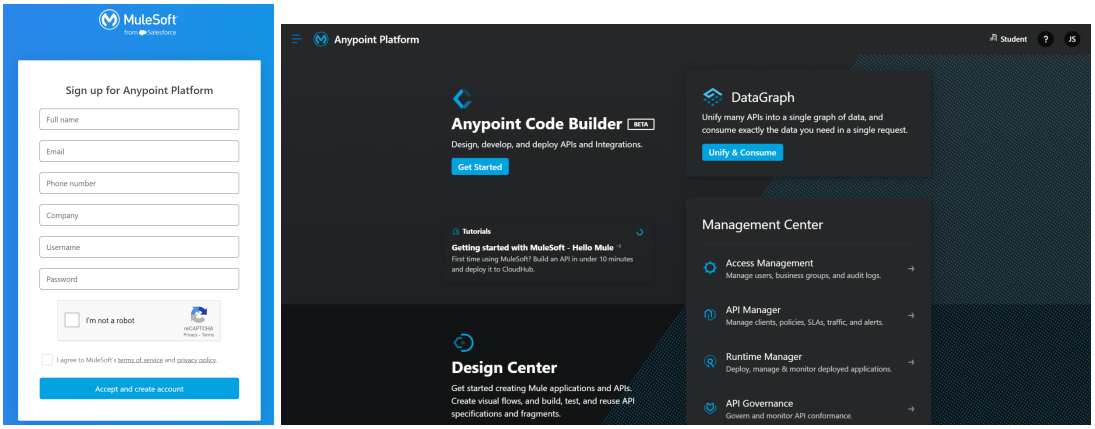
Technical practice

Jesús Sánchez Nájera

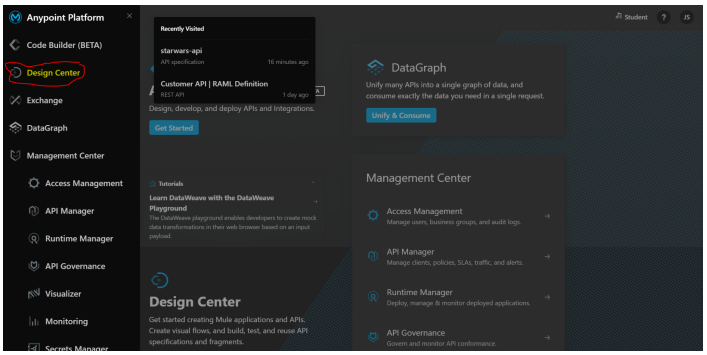
Content

API specification design

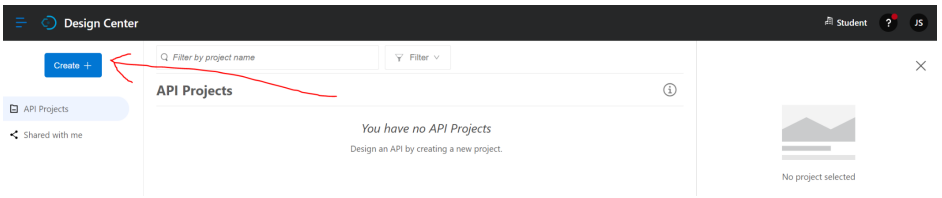
The first step is to create an account in Anypoint Platform.



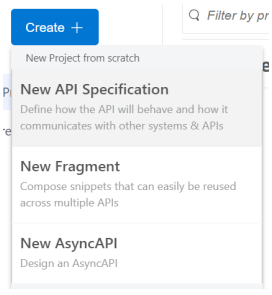
Once the account is created and we are on the main screen, we will look for the option "DESIGN CENTER" in the left side menu.



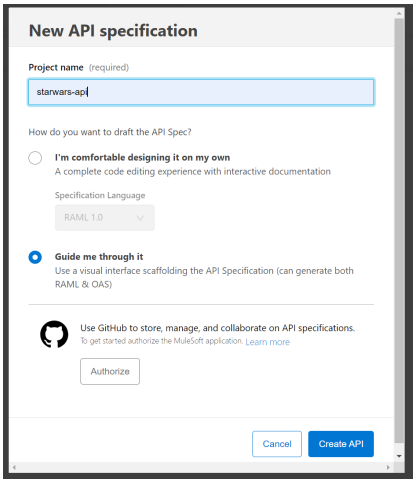
On the next page go to “Create +”



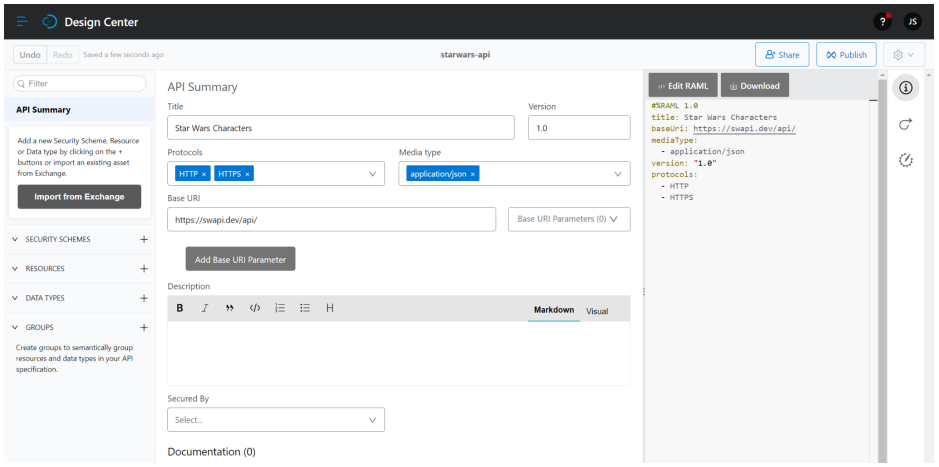
Click on “New API Specification”



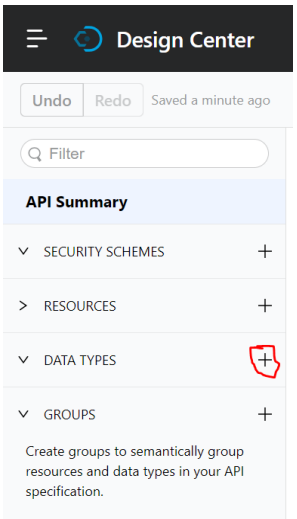
Create API with the next values



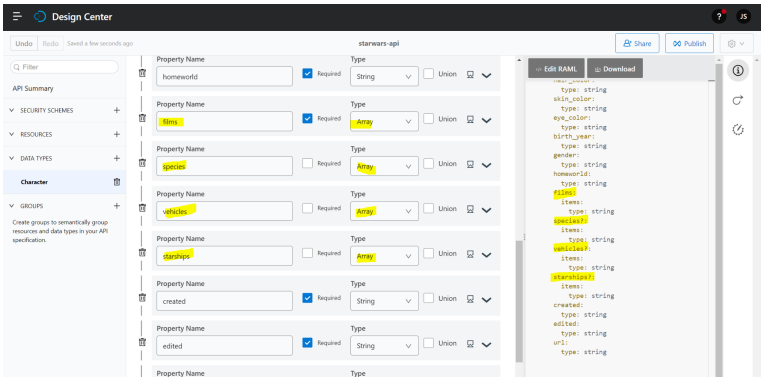
The next step is start adding the properties of our api

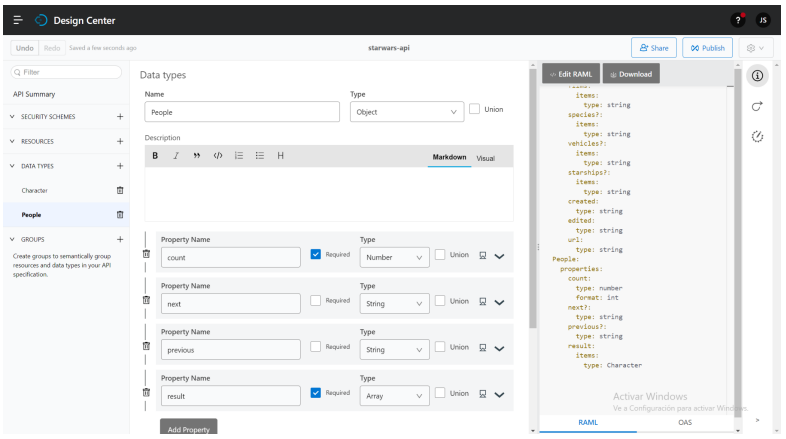


Then we will aggregate a Data Types

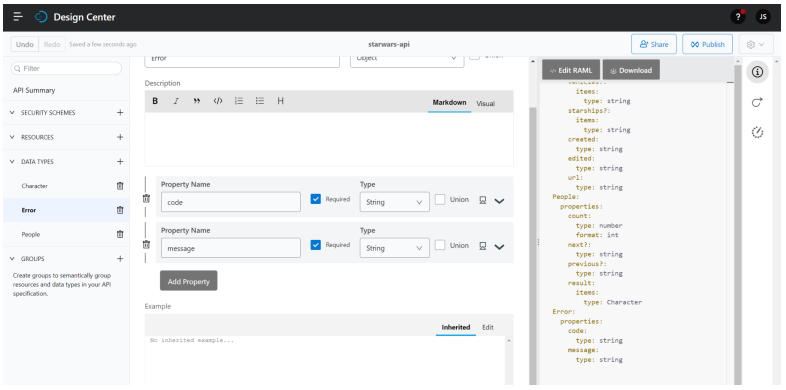


In this case we will add the values of the response that we expect in the api based on the response that the SWAPI offers us

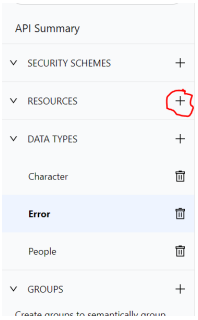




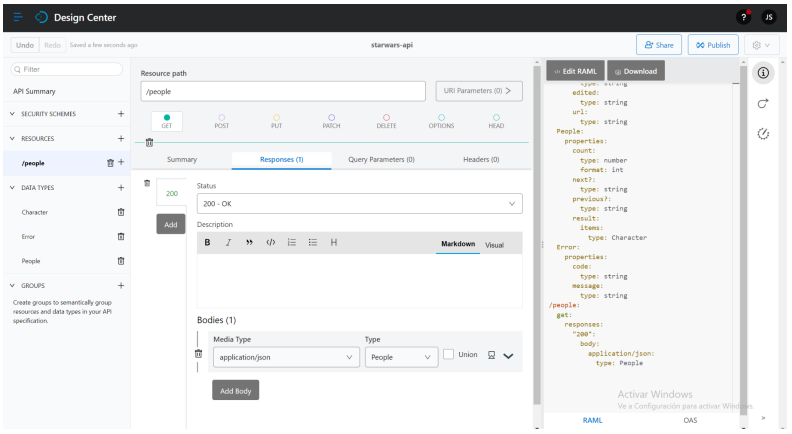
And aggregate a datatype from the possible error responses



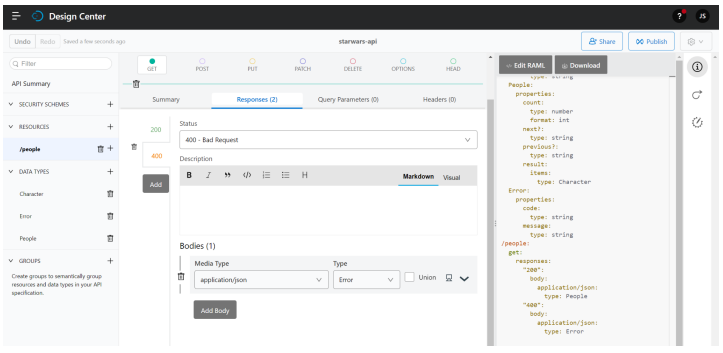
Once we finish adding the datatypes and properties we will start with the resources

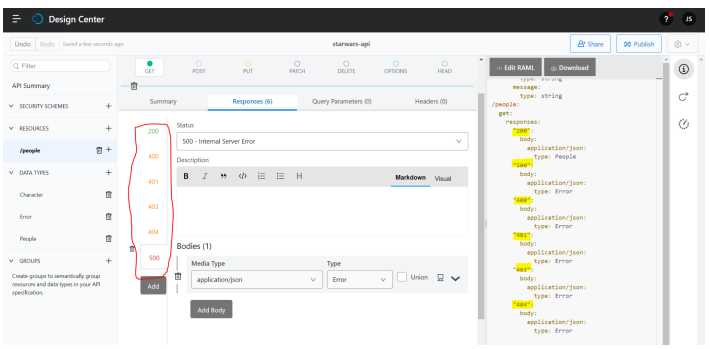


In this case, we will add the code OK-200 and assign the datatype "people" as the expected response.



We continue adding the rest of the possible error response codes





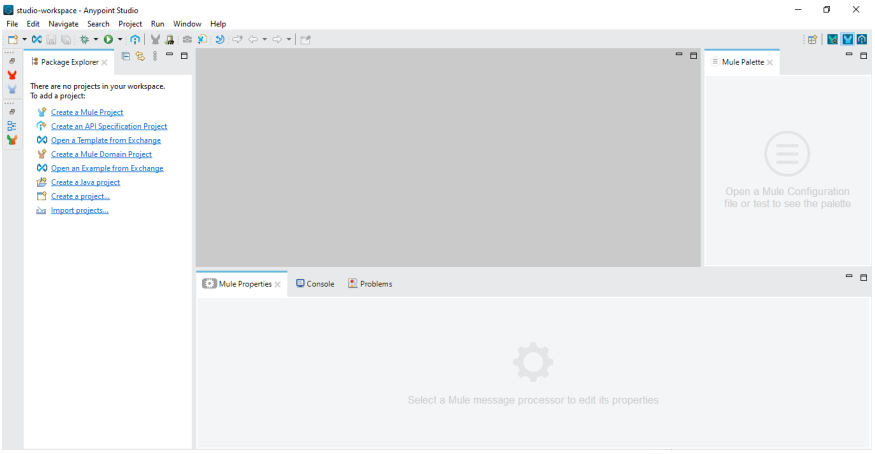
Done!, we already have our api specification in RAML format.

API specification in RAML format

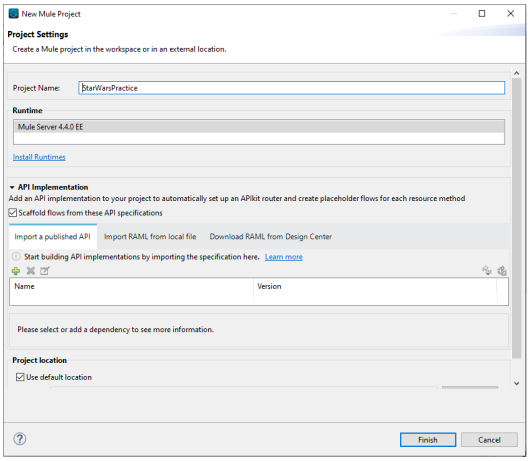
```
#%RAML 1.0
title: Star Wars Characters
baseUri: https://swapi.dev/api/
mediaType:
  - application/json
version: "1.0"
protocols:
  - HTTP
  - HTTPS
types:
  Character:
    properties:
      name:
        type: string
      height:
        type: string
      mass:
        type: string
      hair_color:
        type: string
      skin_color:
        type: string
      eye_color:
        type: string
      birth_year:
        type: string
      gender:
        type: string
      homeworld:
        type: string
      films:
        items:
          type: string
      species?:
        items:
          type: string
      vehicles?:
        items:
          type: string
      starships?:
        items:
          type: string
      created:
        type: string
      edited:
        type: string
      url:
        type: string
  People:
    properties:
      count:
        type: number
        format: int
      next?:
        type: string
      previous?:
        type: string
      result:
        items:
          type: Character
  Error:
    properties:
      code:
        type: string
      message:
        type: string
/people:
  get:
    responses:
      "200":
        body:
          application/json:
            type: People
      "500":
        body:
          application/json:
            type: Error
      "400":
        body:
          application/json:
            type: Error
      "401":
        body:
          application/json:
            type: Error
      "403":
        body:
          application/json:
            type: Error
      "404":
        body:
          application/json:
            type: Error
```

Create a Mule Project

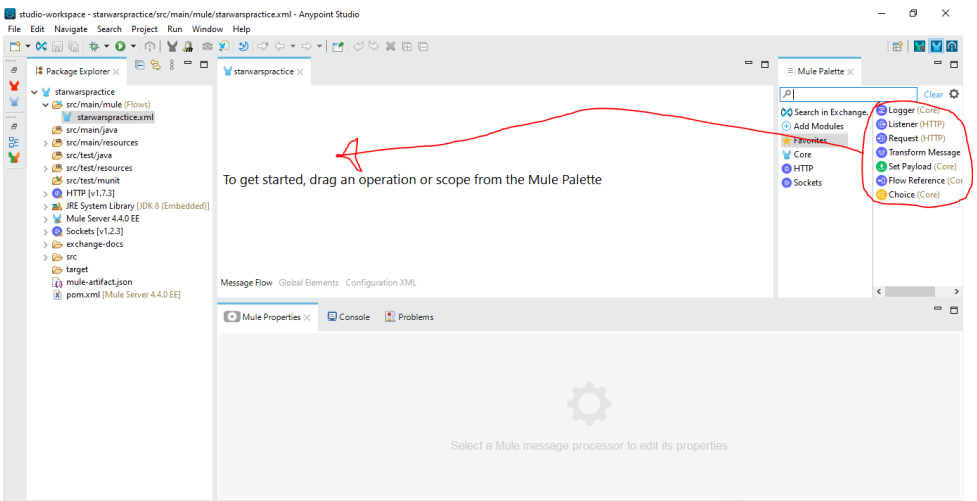
After download and install anypoint studio we are going to create a new mule project



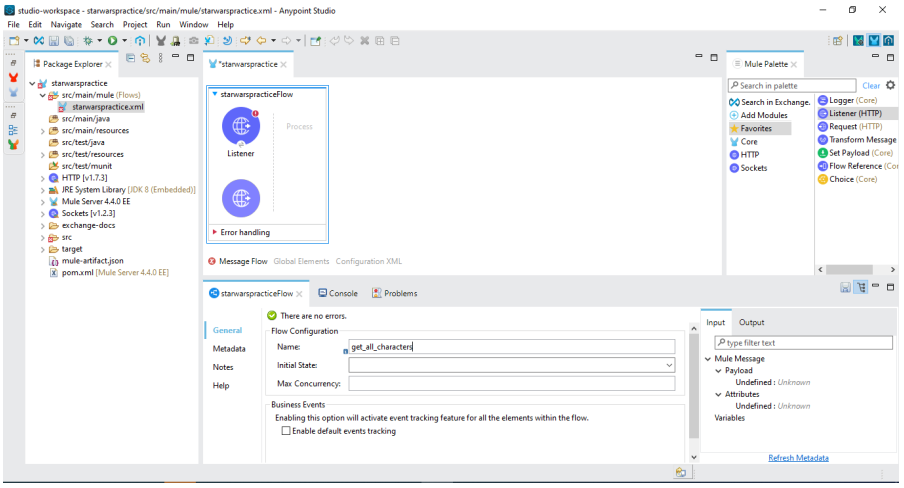
We can name the project whatever we want



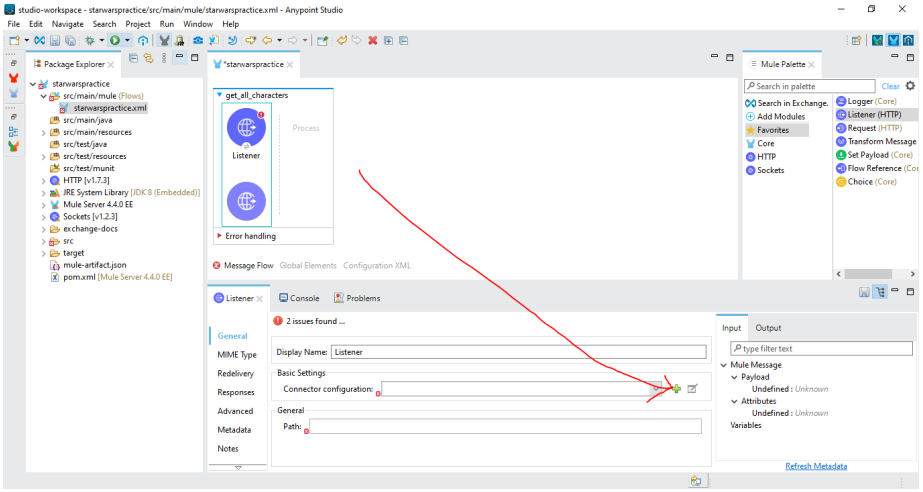
The next step is to add the components by dragging and dropping in the workspace according to what we need



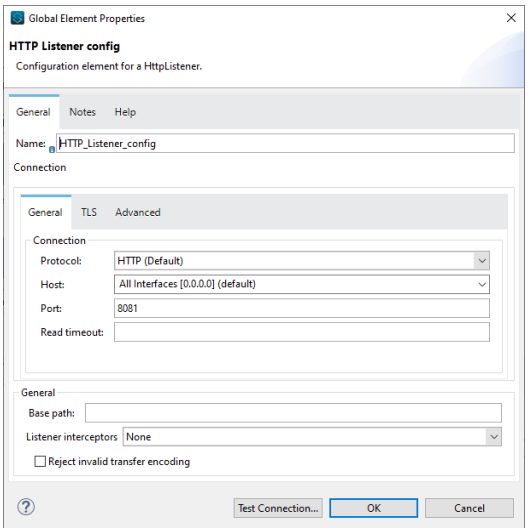
The first component we will use is a *"listener"*.



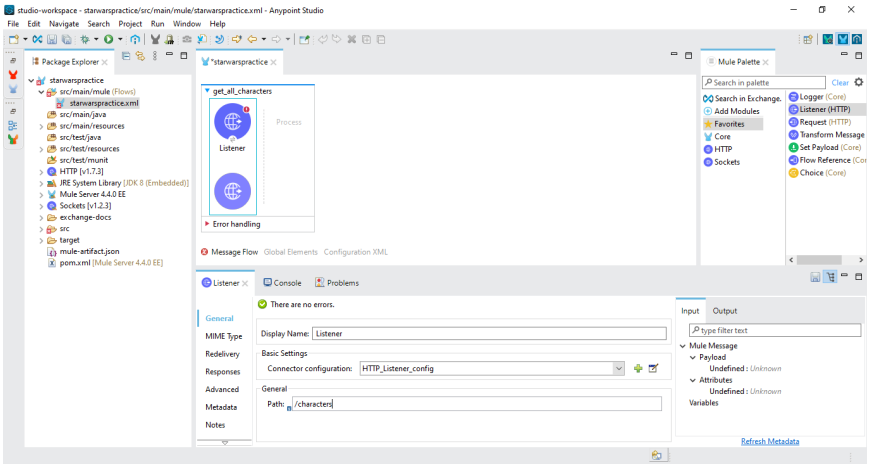
Click on “+” button.



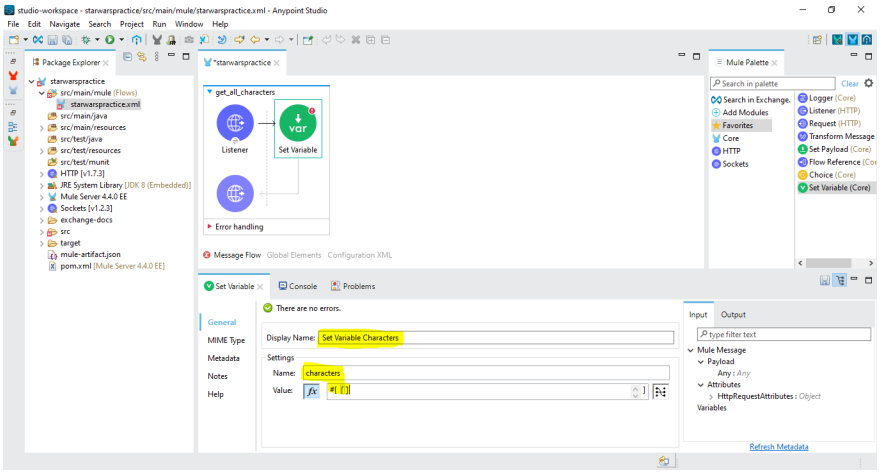
Click the ok button with the default properties.



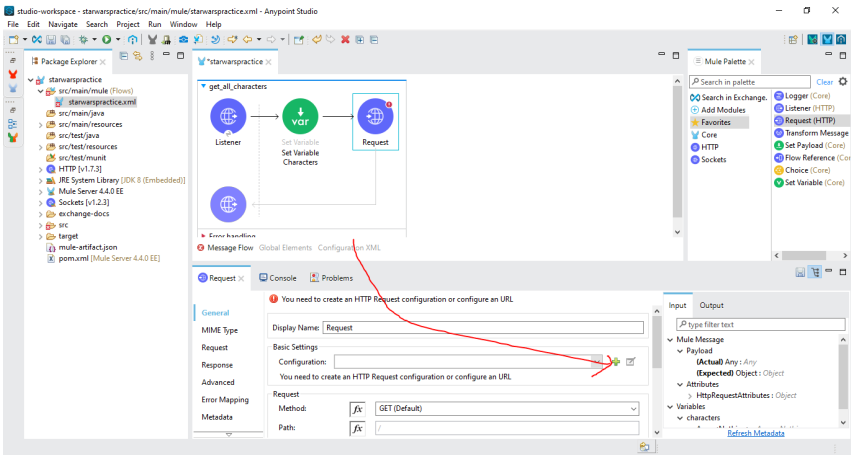
Agregamos el Path *"/characters"*.



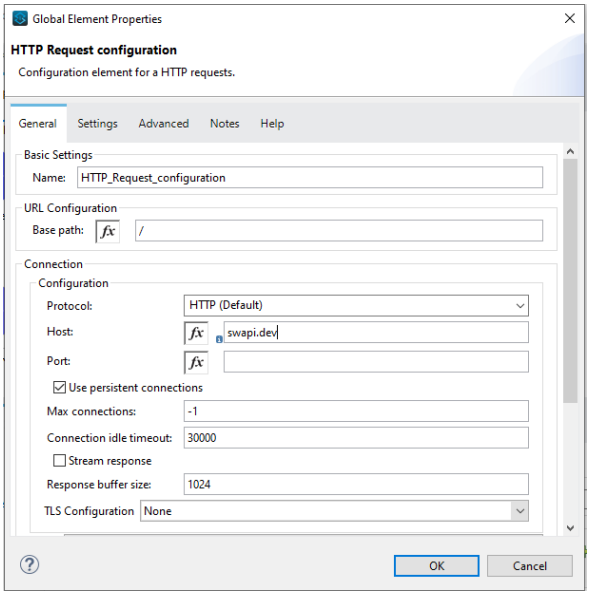
The next component we will use is *"set variable"* and we will initialize a variable of type array called characters.



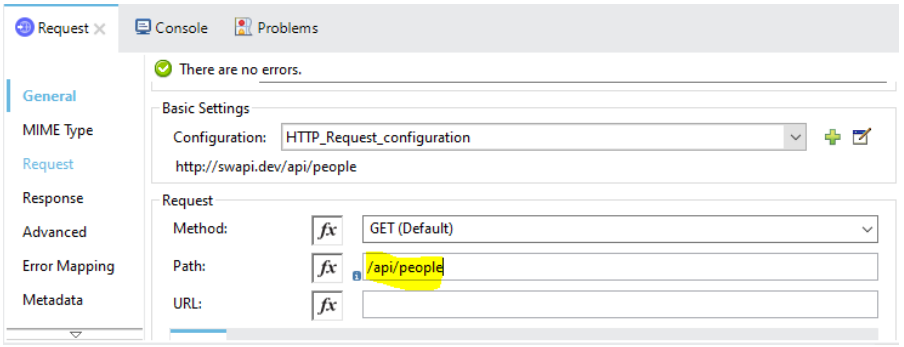
The next component we will use is "request". We will click on the "+" button to add the properties.



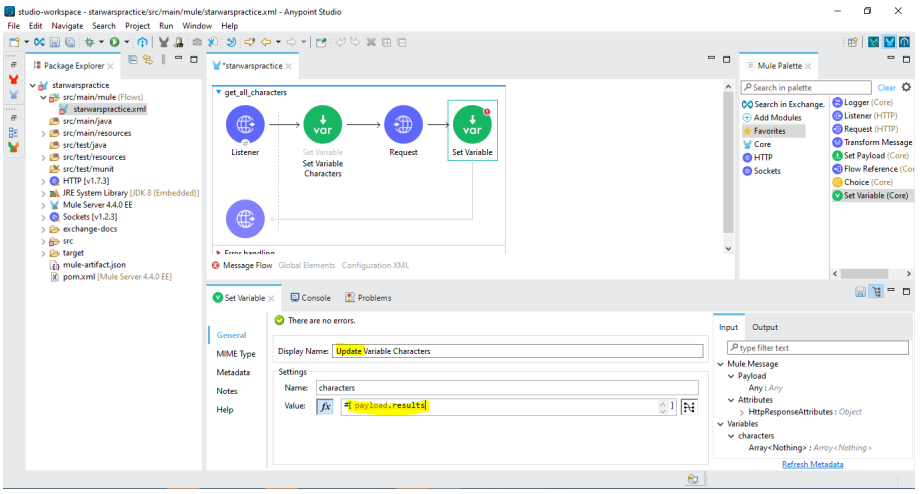
We will add *"swapi.dev"* in the *"Host"* property.



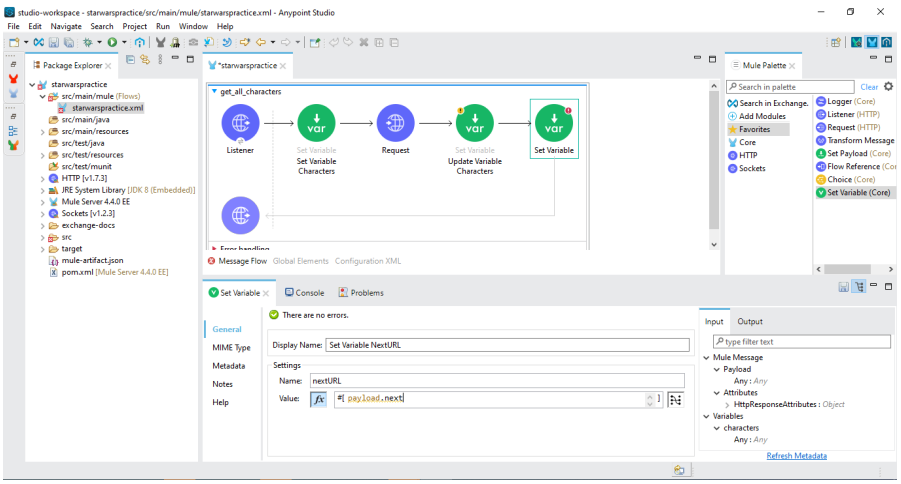
In the request properties we will add *"/api/people/"* in the *"Path"* property.



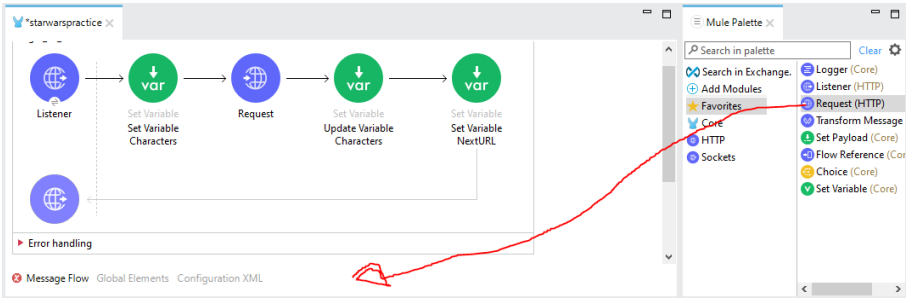
The next component is a *"set variable"* to assign the characters obtained from the previous request in the *"characters"* variable.



The next component is a *"set variable"* to assign the nextUrl obtained from the previous request, this url will give us the information of the characters of the next page of the SWAPI.

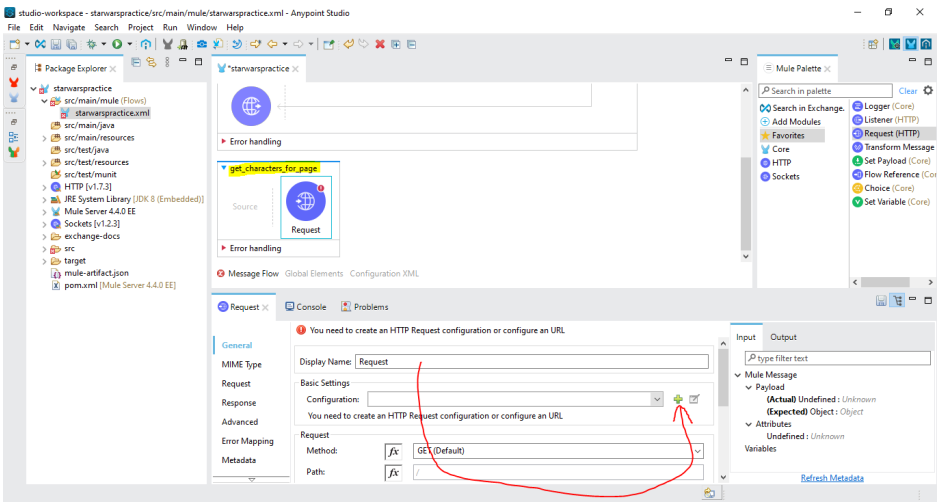


The next step is to drag and drop a *"request"* component outside of the flow we're working on to create another flow that will serve as a recursive function.

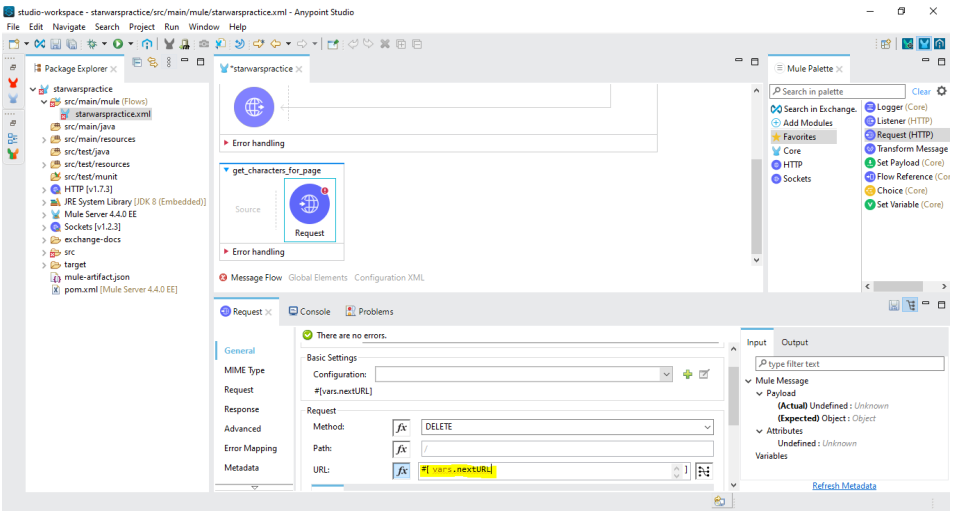


This recursive function will help us to navigate through the SWAPI character pages.

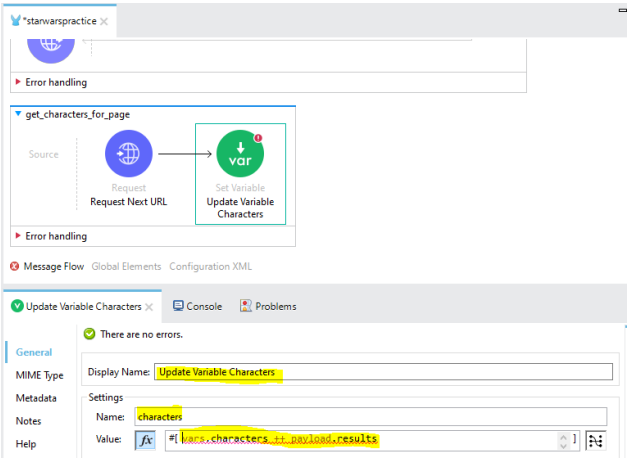
In the properties of the *"request"* component we will click on the *"+"* button.



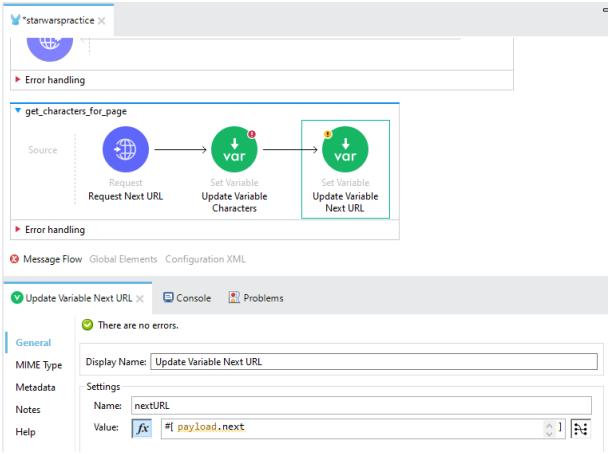
In the URL property we will assign the value of the *"nextURL"* variable.



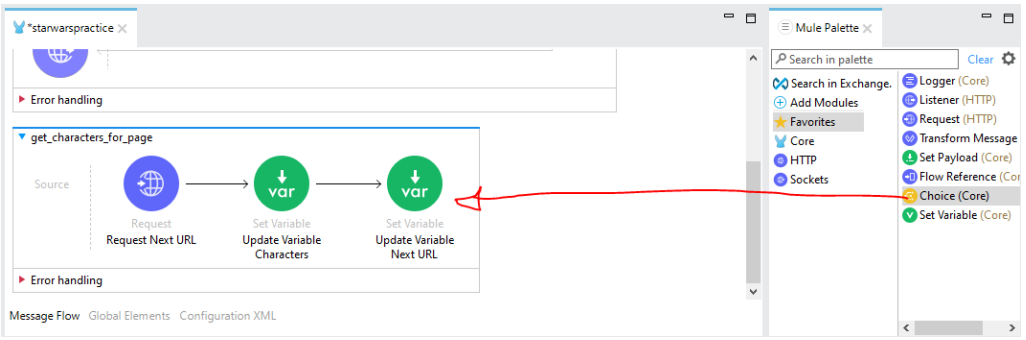
We add a *"Set Variable"* component to add the information obtained from this new request to the *"characters"* variable.



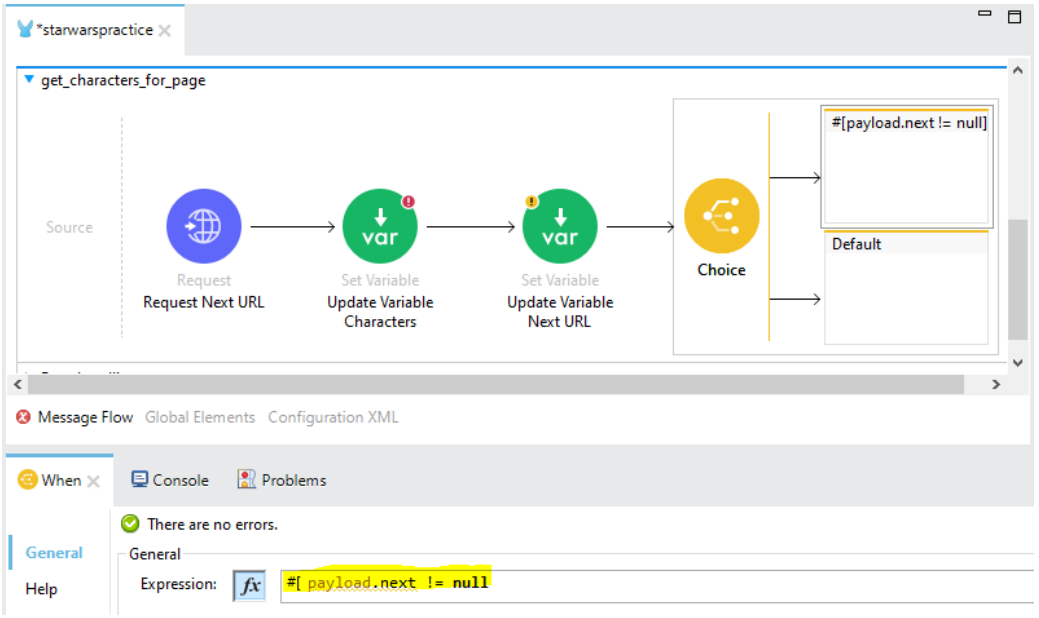
We add another *"Set Variable"* component to add to the *"nextURL"* variable the url of the next page of the SWAPI characters obtained from the new request.



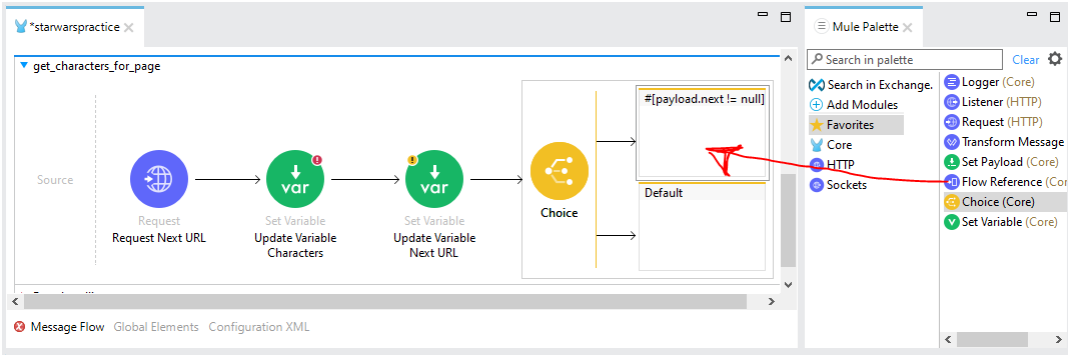
The next component we will add is *"Choice"*.



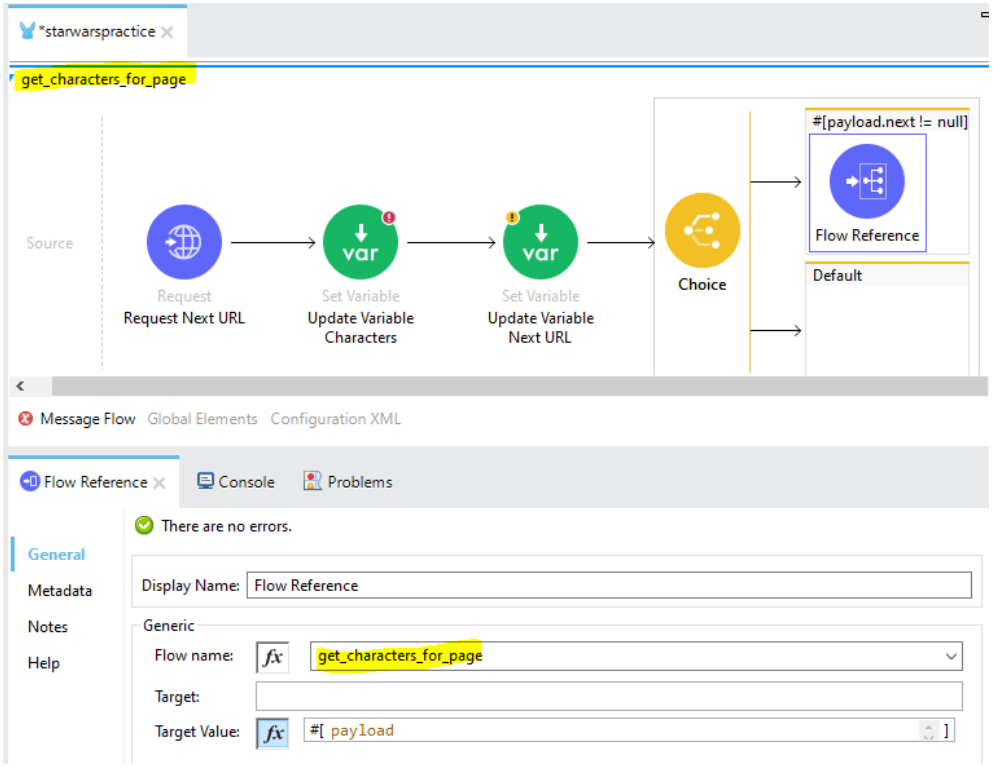
This component will make a validation. If the *"nextURL"* value of the previous request exists, the flow will continue.



In the case of continuing we will add the *"Flow Reference"* component.

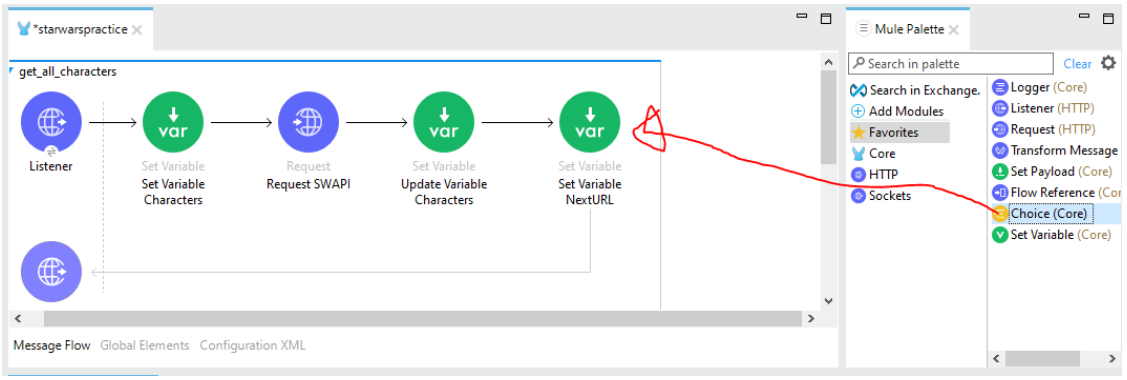


In the properties of this component we are going to assign the same flow as the objective to apply the recursion.

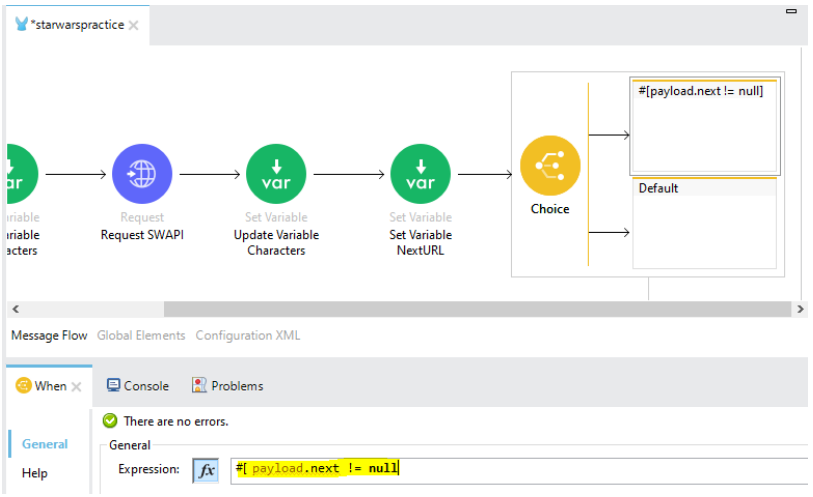


At this point it's important to understand the recursion of the flow, it will consume itself by changing the url of the request to the next one until the nextURL no longer exists.

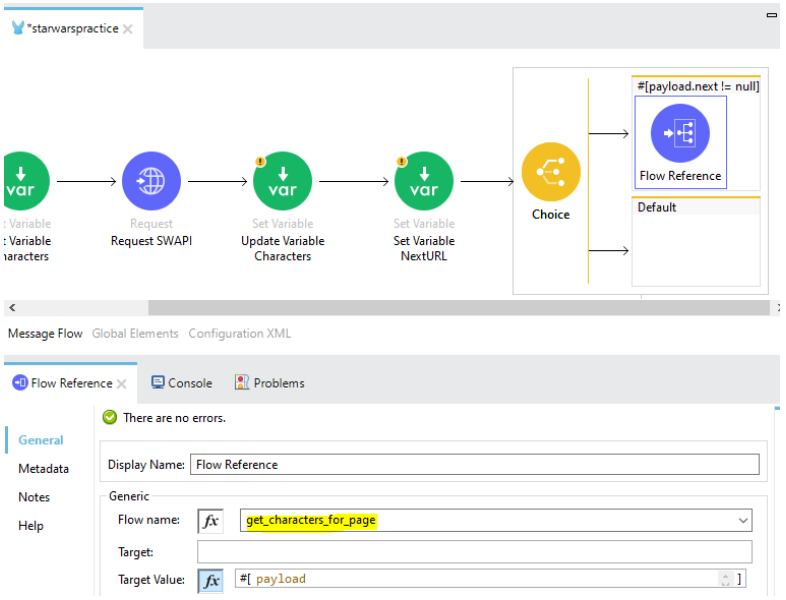
Going back to the first flow we are going to add a *"choice"* component.



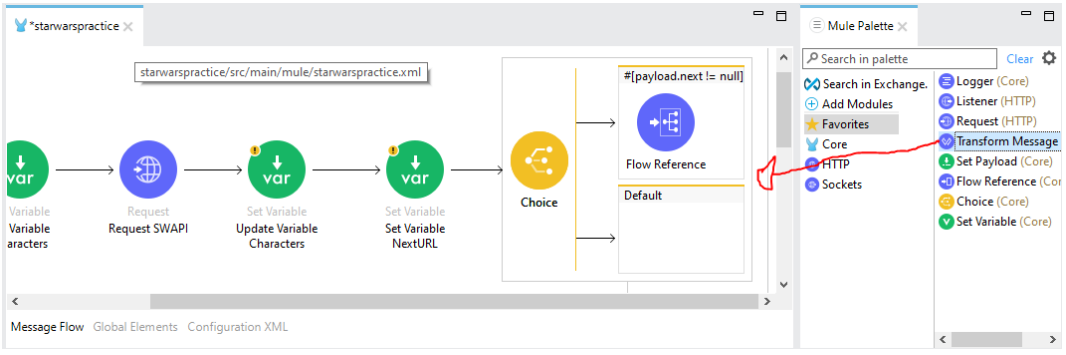
Once again, the choice component will perform a validation in which it will continue with the flow as long as the *"nextURL"* variable of the previous request is not null.



Here we will use a *"Flow Reference"* component to invoke our second created flow which we assimilate as a recursive function.

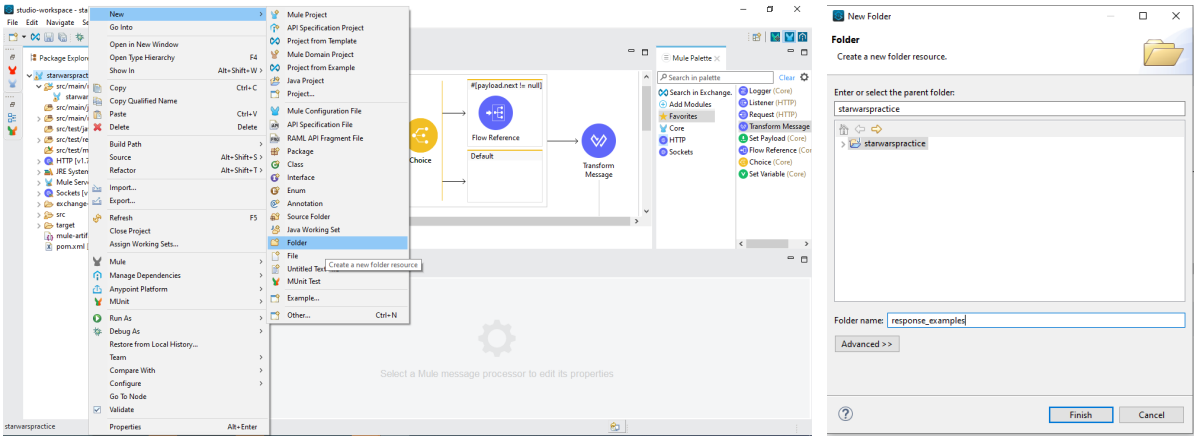


Finally we will use a *"Transform Message"* component.

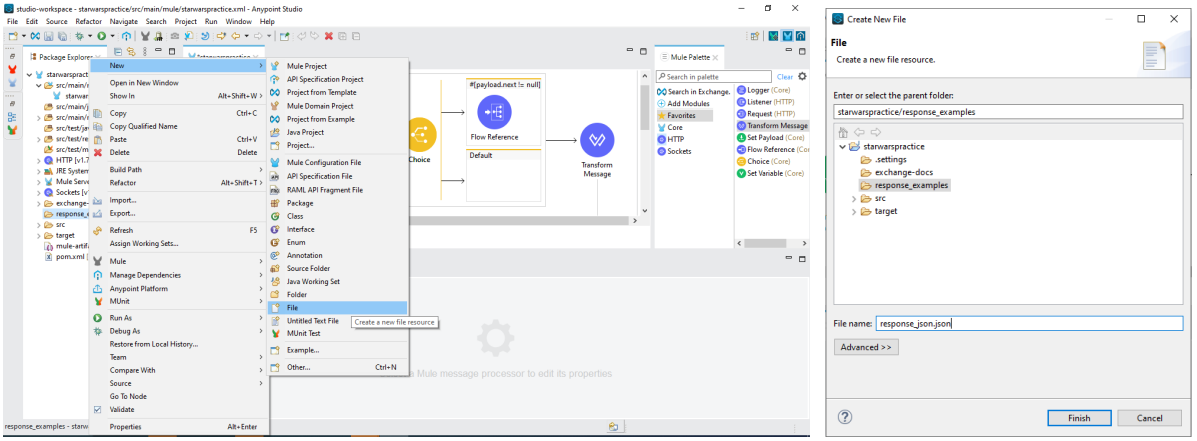


In the *"Transform message"* component we can format the response that our project will give. To give the response in the requested format it is necessary to import two files as an example. The first is a .json file which is the format of the response provided by SWAPI. The second is a .csv file that is the format of the response that our project will give.

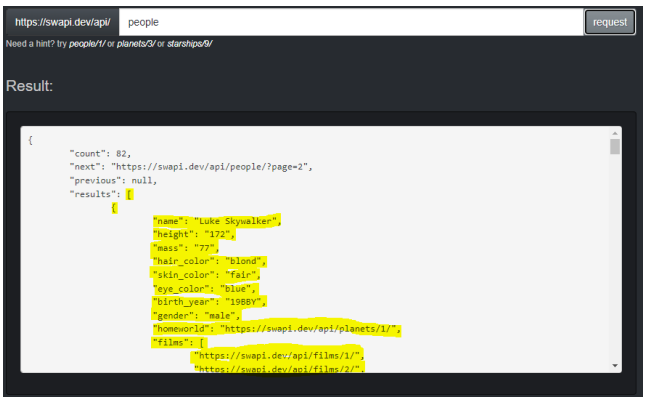
We will create a new folder called *“response_examples”*.



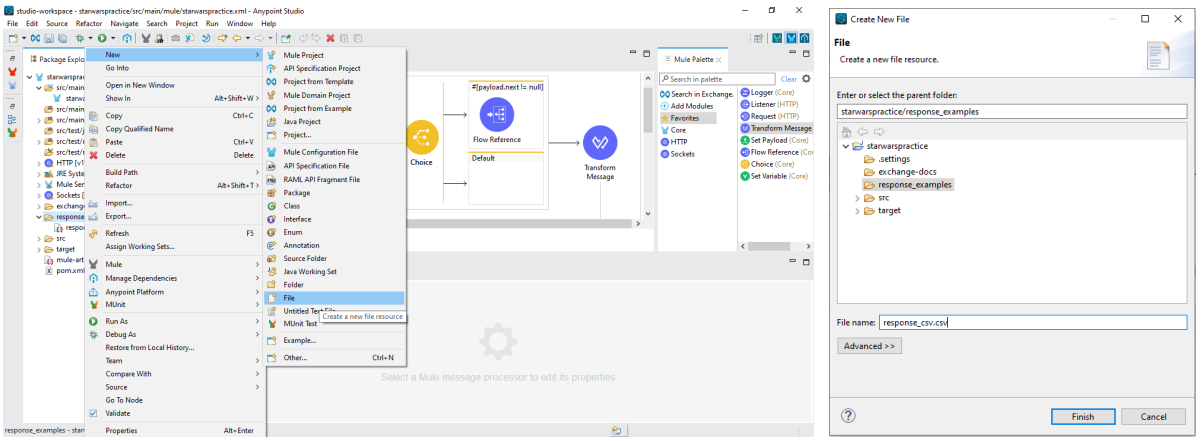
Inside the folder we will create a file called *“response_json.json”*.



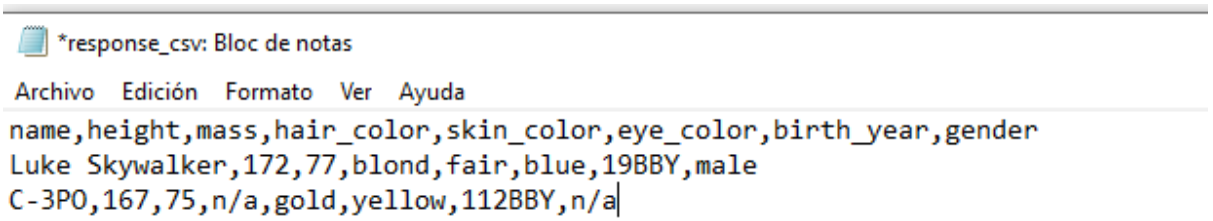
This file will contain the json that SWAPI gives us.



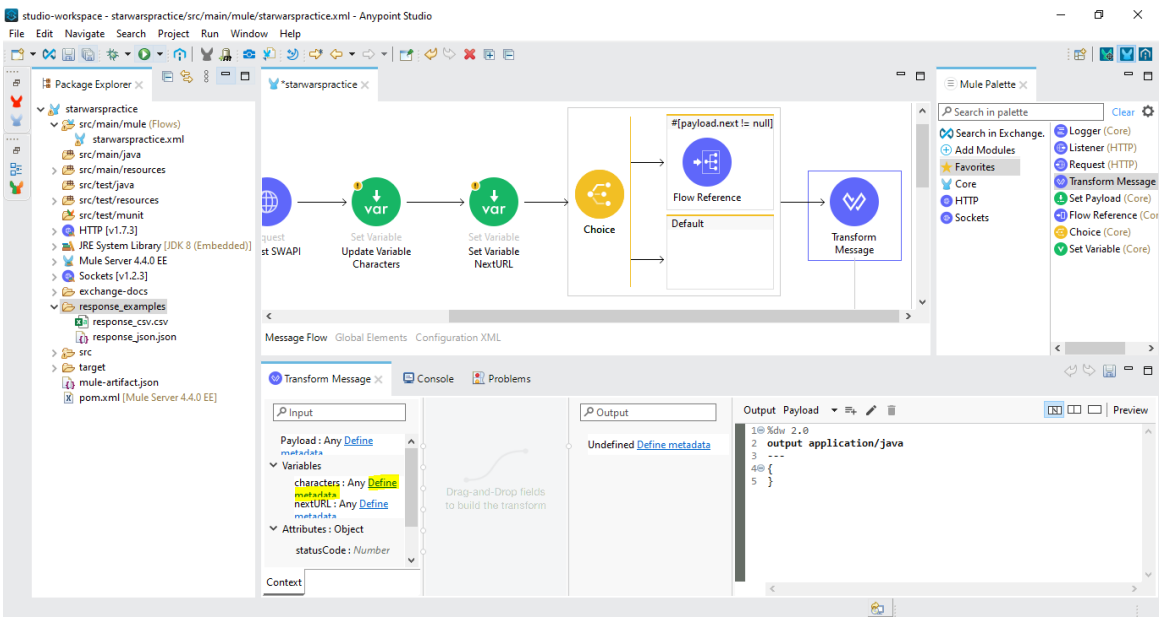
In the same folder we will create a file called *“response_csv.csv”*.



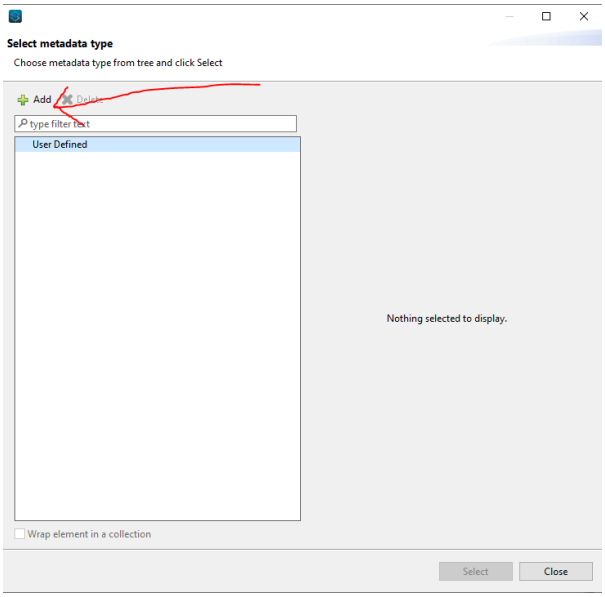
This file will contain the required format.

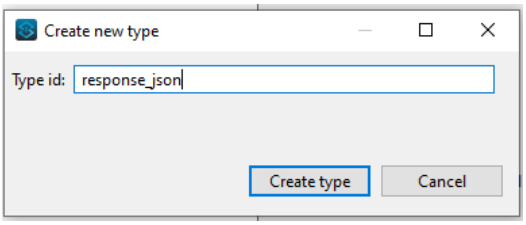


The next step is to map the response from the *"Transform Message"* component.
We define our *"Characters"* variable with the json format that we added as an example.
Click on *“Define metadata”* on variable *“Characters”*

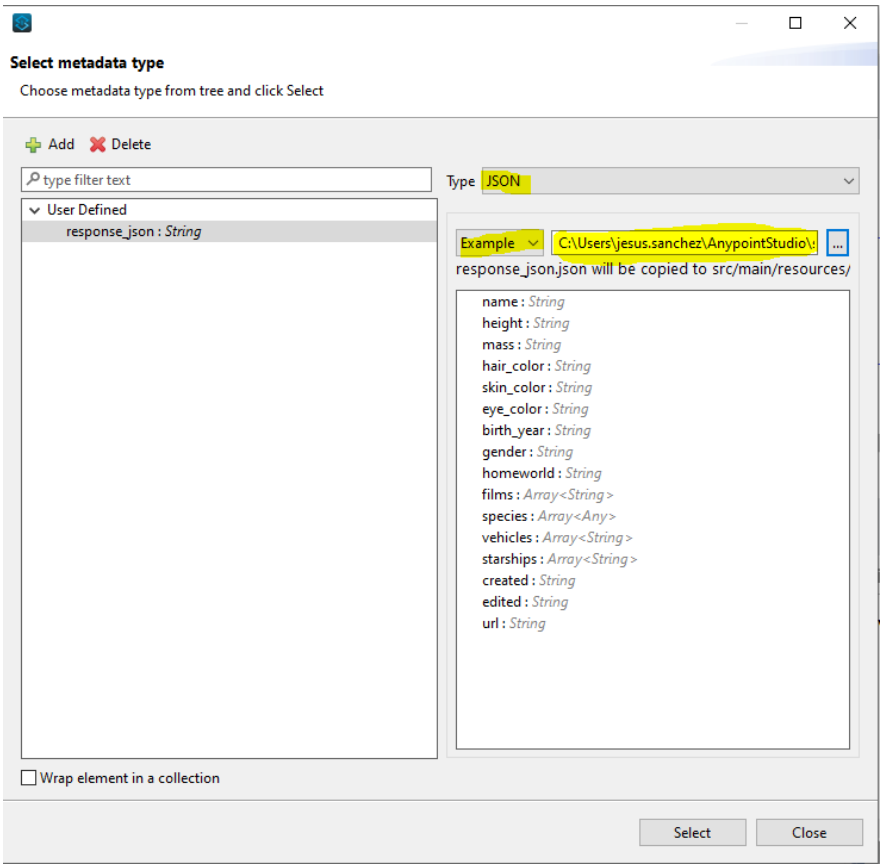


Follow the next steps.

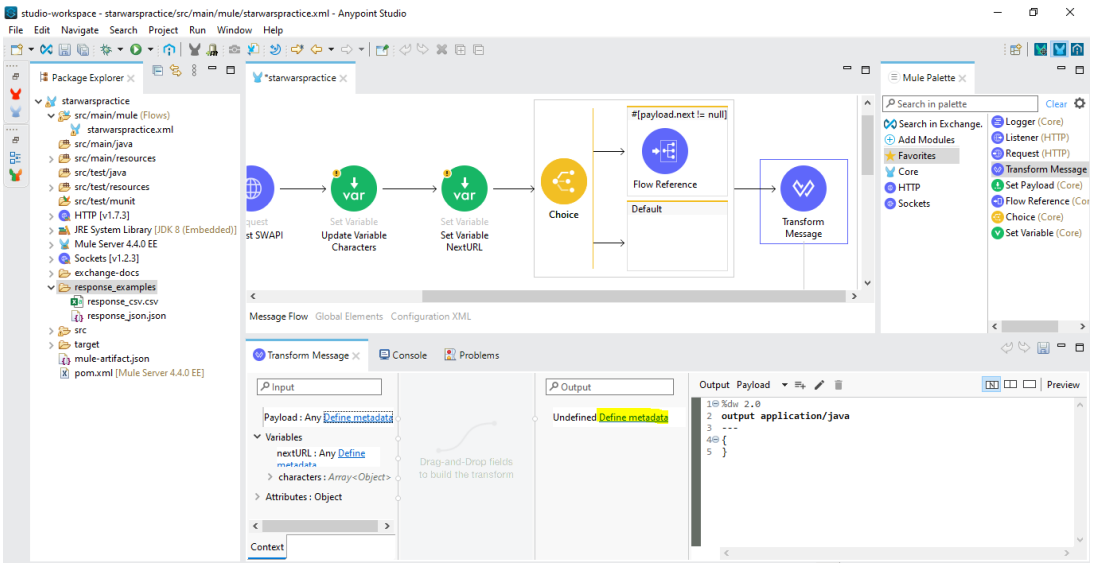


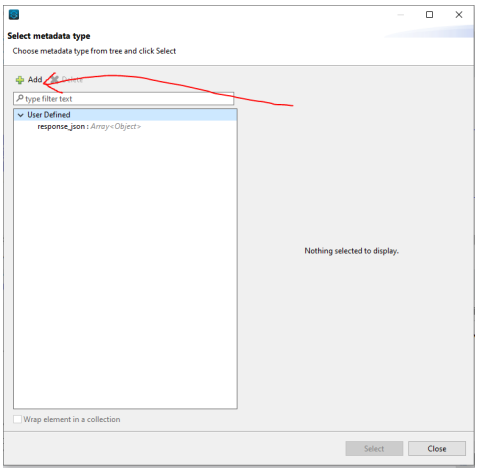


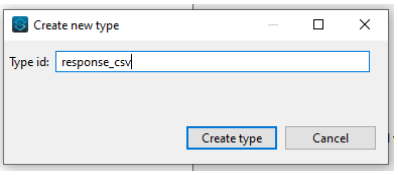
We choose the following options.



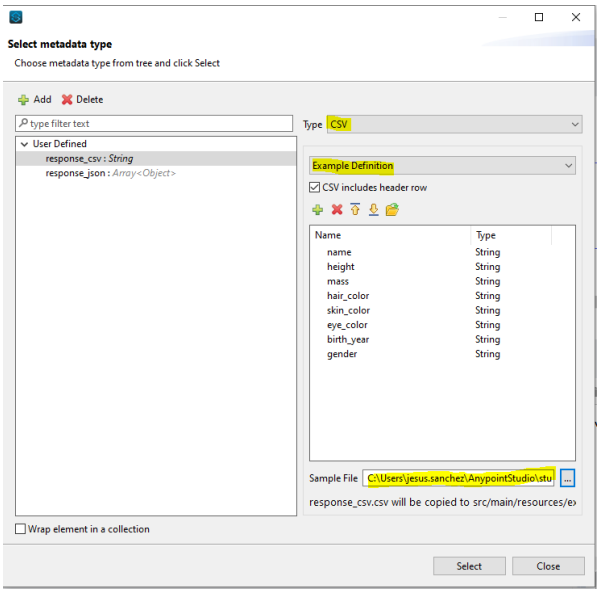
We apply the same steps for output but with the csv format



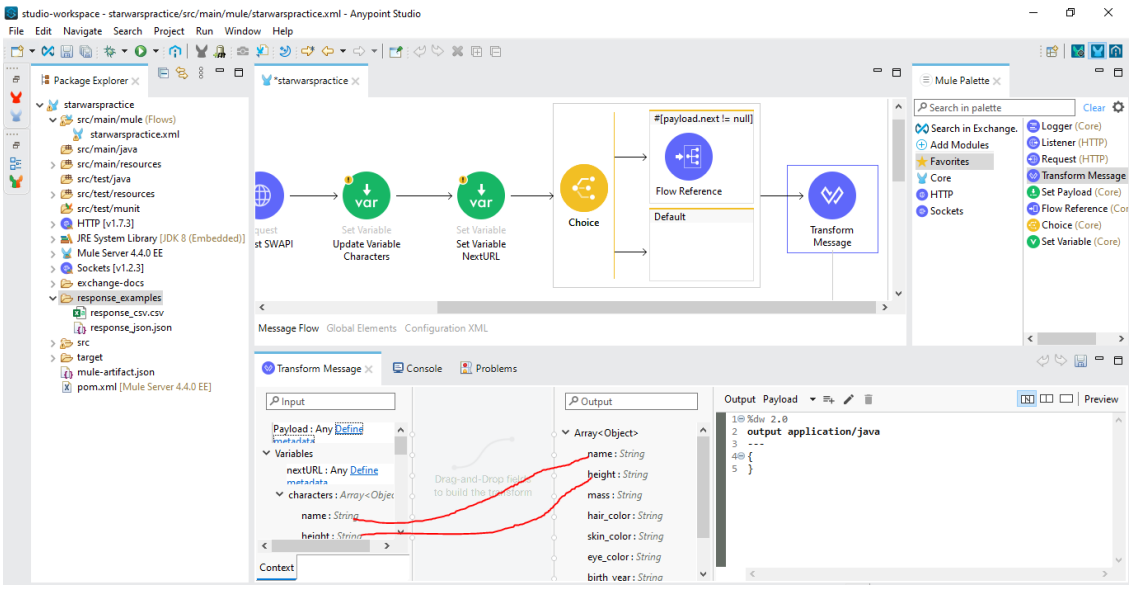




We choose the following options.

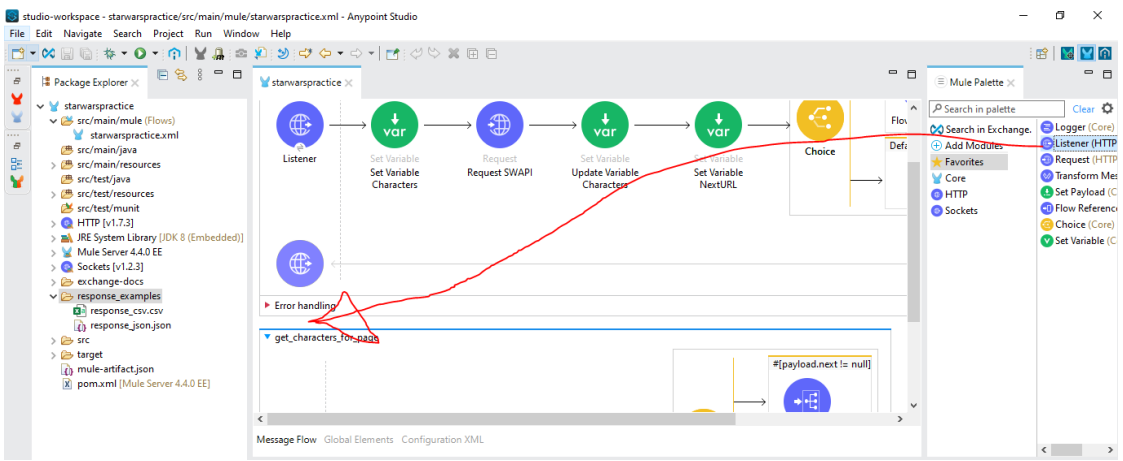


The next step is to map the variables, from the *"Characters"* variable to the output object.

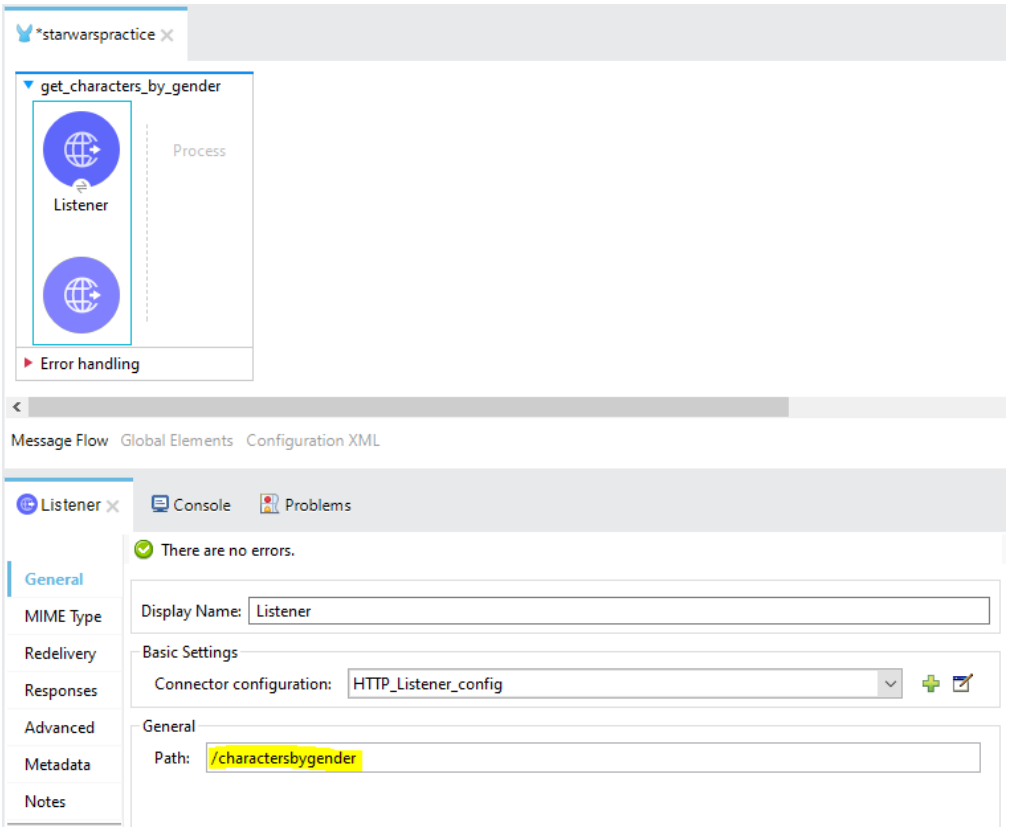


At the end of the mapping we will obtain the following code.

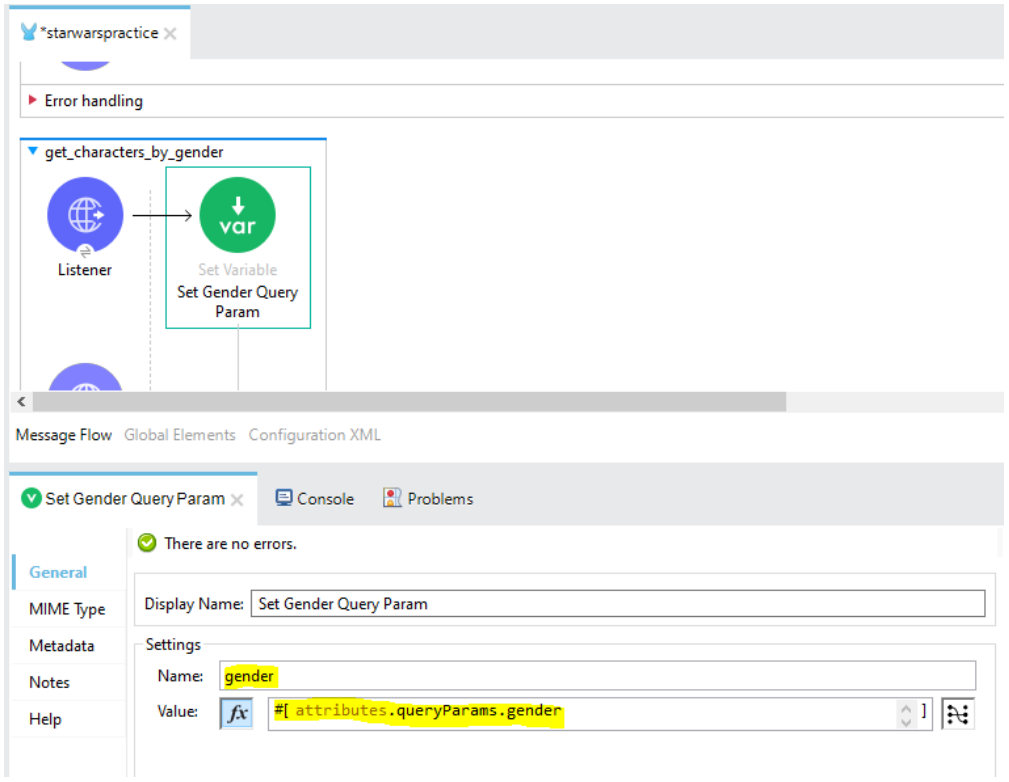
To add the functionality of filtering the response through a QueryParameter we need to add another flow starting with a *"Listener"* component.



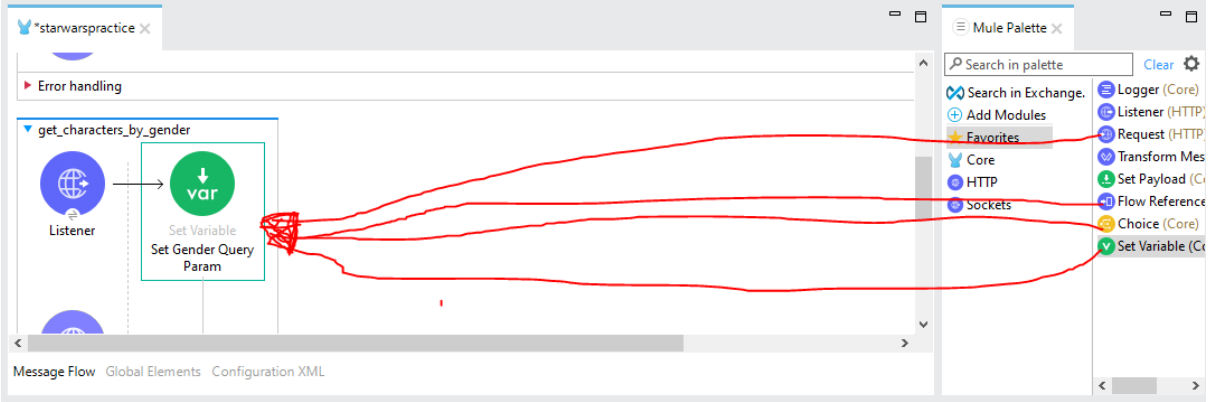
In the Path property we add *"/charactersbygender"*.



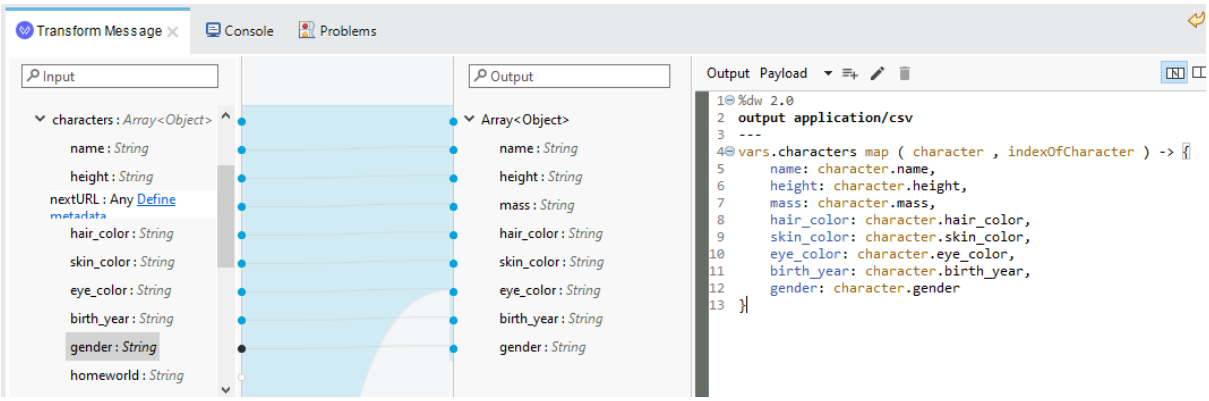
The next component is a *"Set Variable"* in which we are going to assign the value of the Query Parameter we receive.



The following components that are added are the same as the first flow with the same properties.



The only thing that will change is the mapping in the *"Transform message"* component.

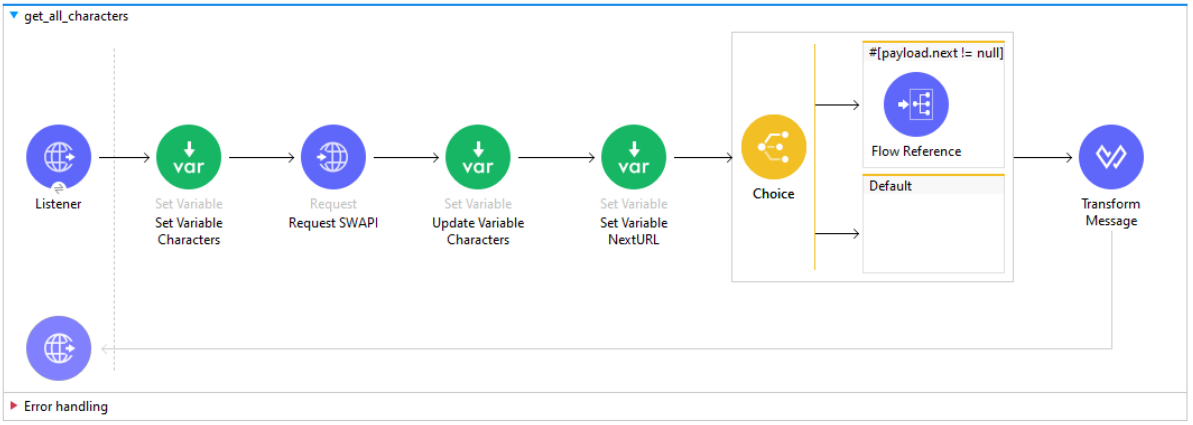


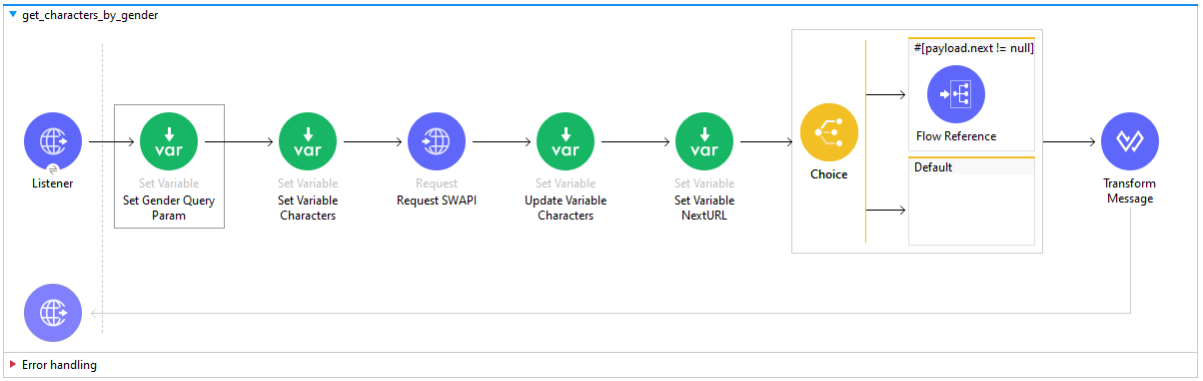
In the code obtained after mapping we are going to add the following filter.

```
%dw 2.0
output application/csv
---
vars.characters filter ($.gender == vars.gender) map ( character , indexOfCharacter ) -> {
    name: character.name,
    height: character.height,
    mass: character.mass,
    hair_color: character.hair_color,
    skin_color: character.skin_color,
    eye_color: character.eye_color,
    birth_year: character.birth_year,
    gender: character.gender
}
```

This filter is used to only give us as a response the elements of the csv that are equal to the Query Parameter that they sent us from the beginning.

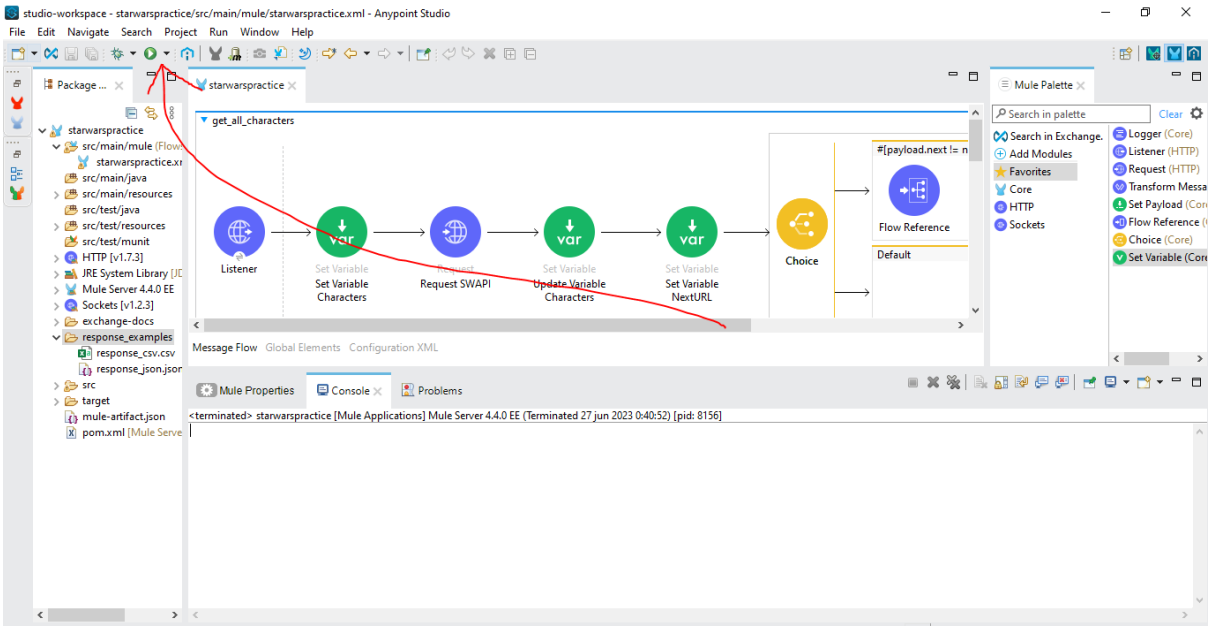
Next we have a comparison of the two main flows.



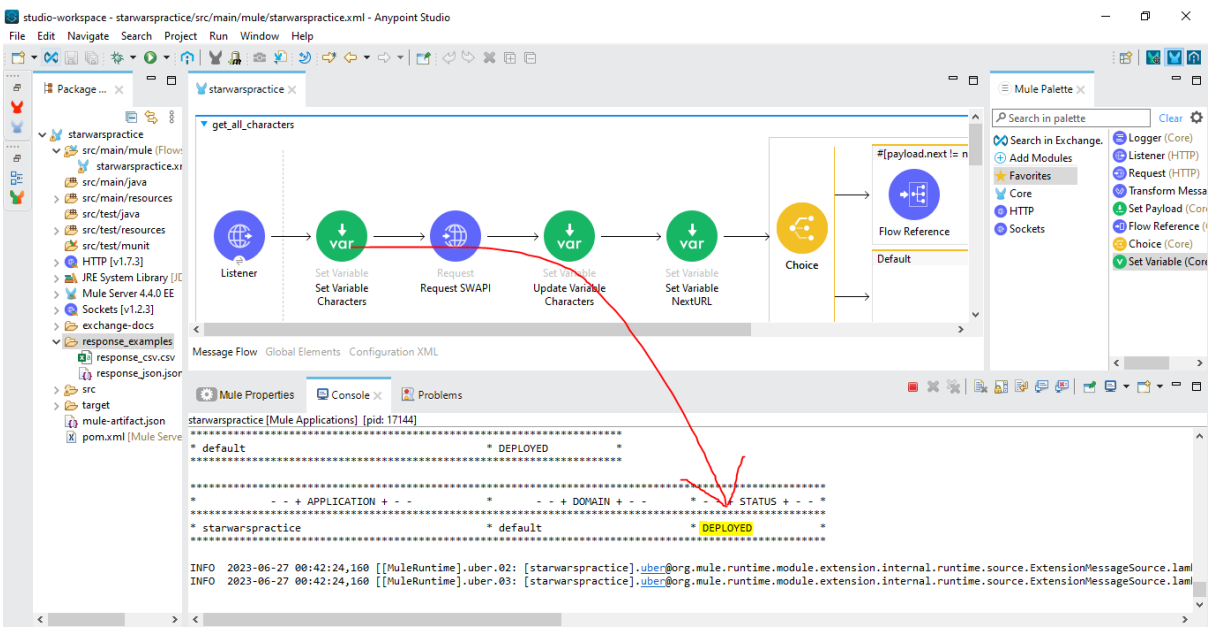


Test API on localhost

When we finish all the development of our project we will click on the *"Run"* button.

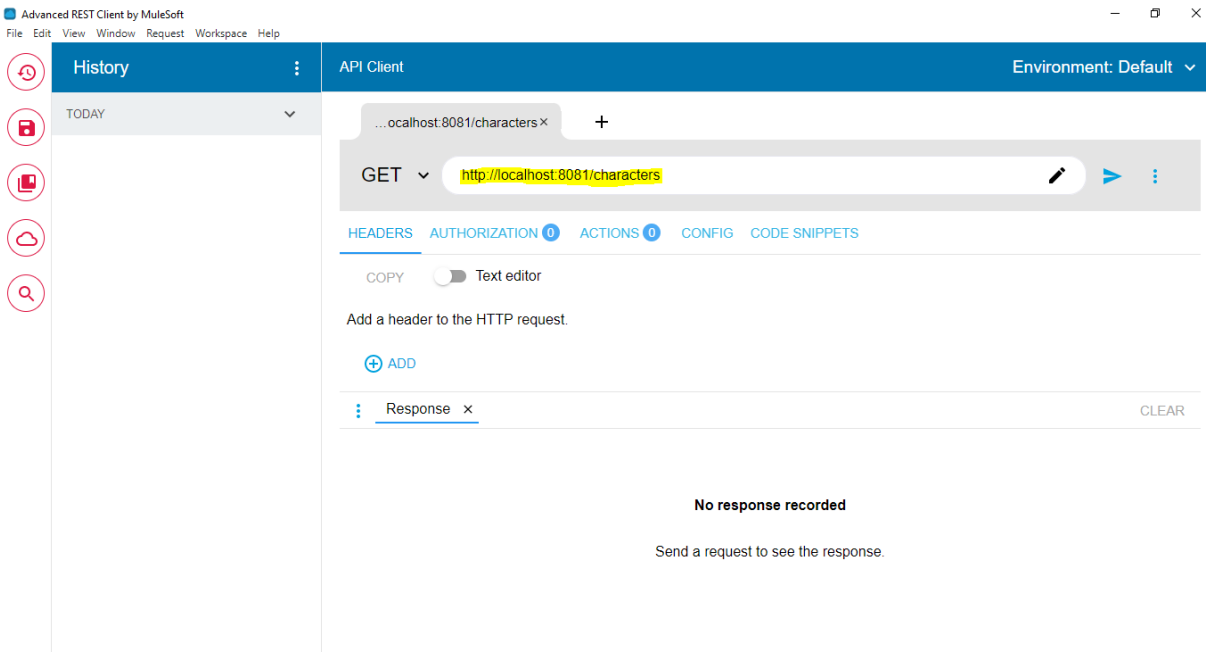


When we see the message *"DEPLOYED"* in the console, we will know that we can now make a request to our API

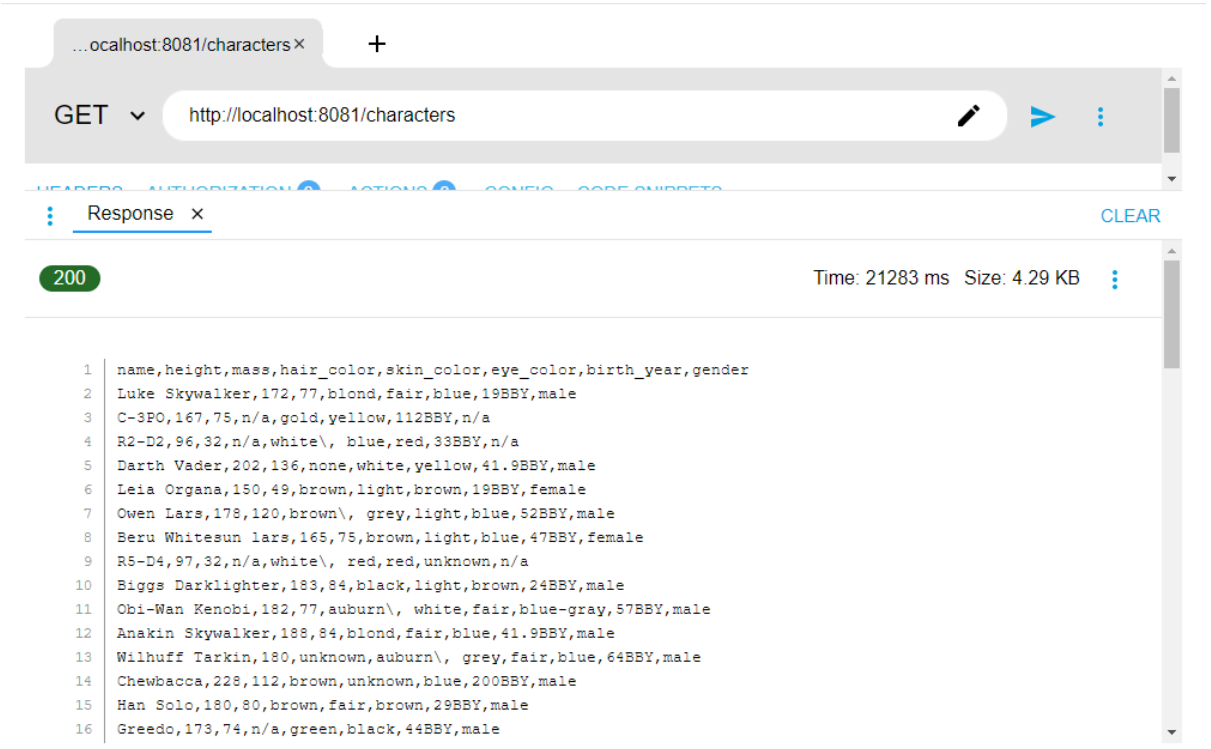


We can do the test in any REST Client, in this case we will use ARC (Advanced REST Client).

The endpoint that we must use to obtain all the characters is http://localhost:8081/characters

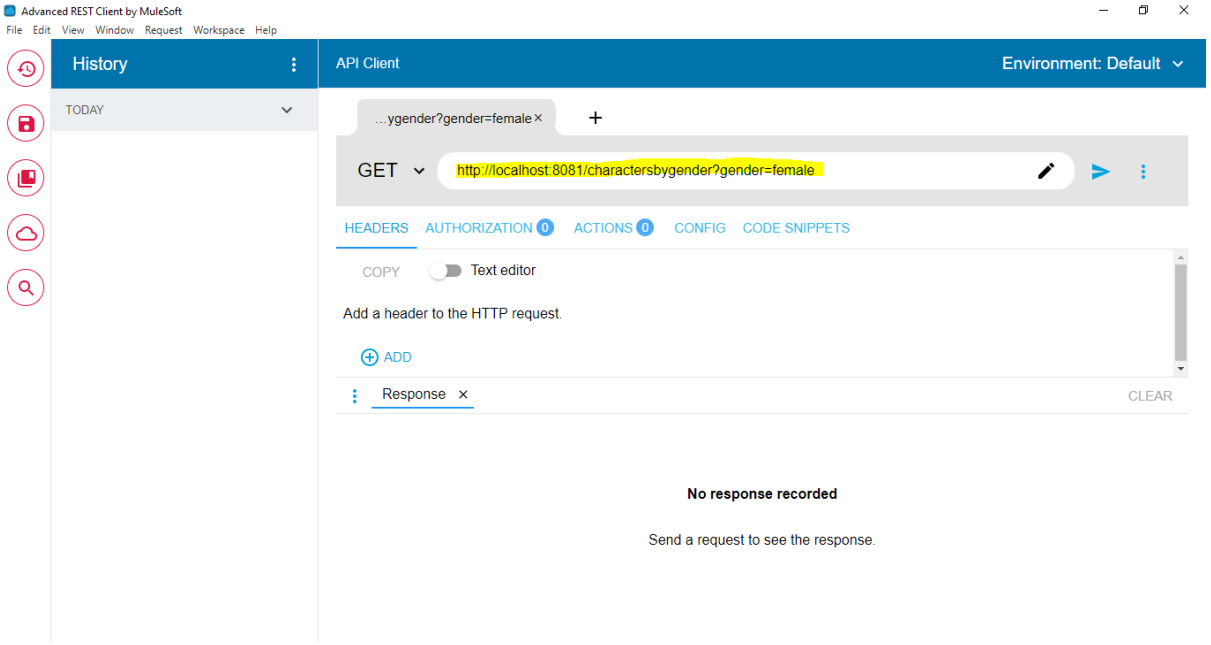


Response.

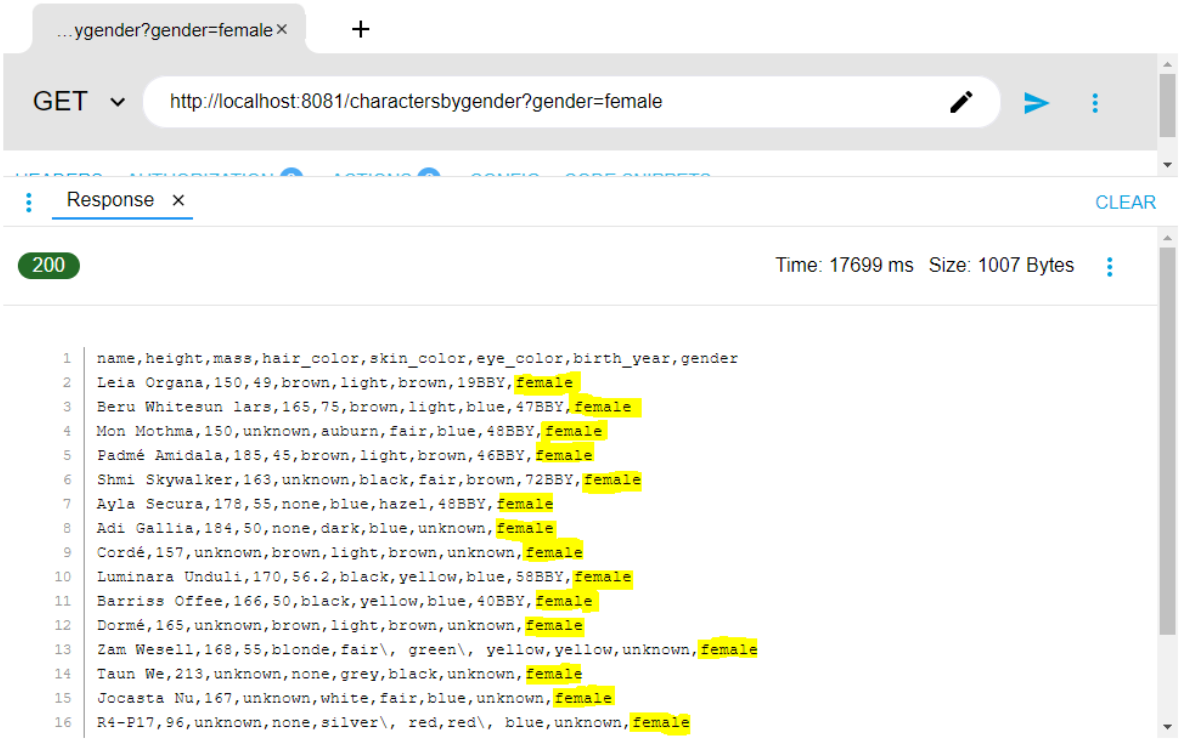


The endpoint to get the characters filtered by gender is:
http://localhost:8081/charactersbygender?gender={gender}

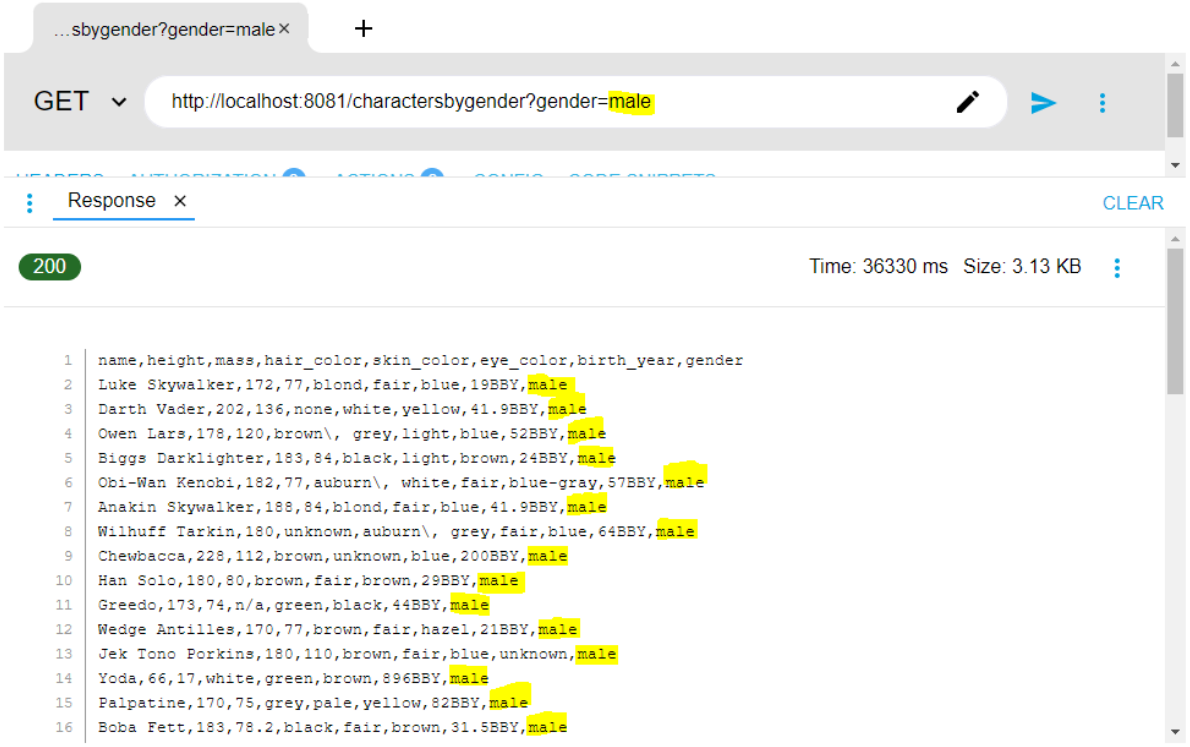
gender = female



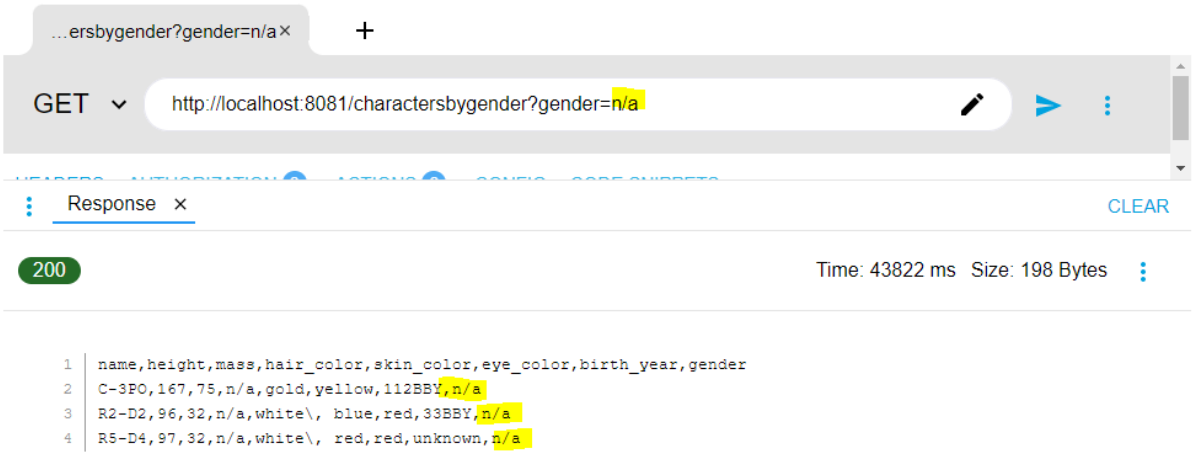
Response:



gender = male



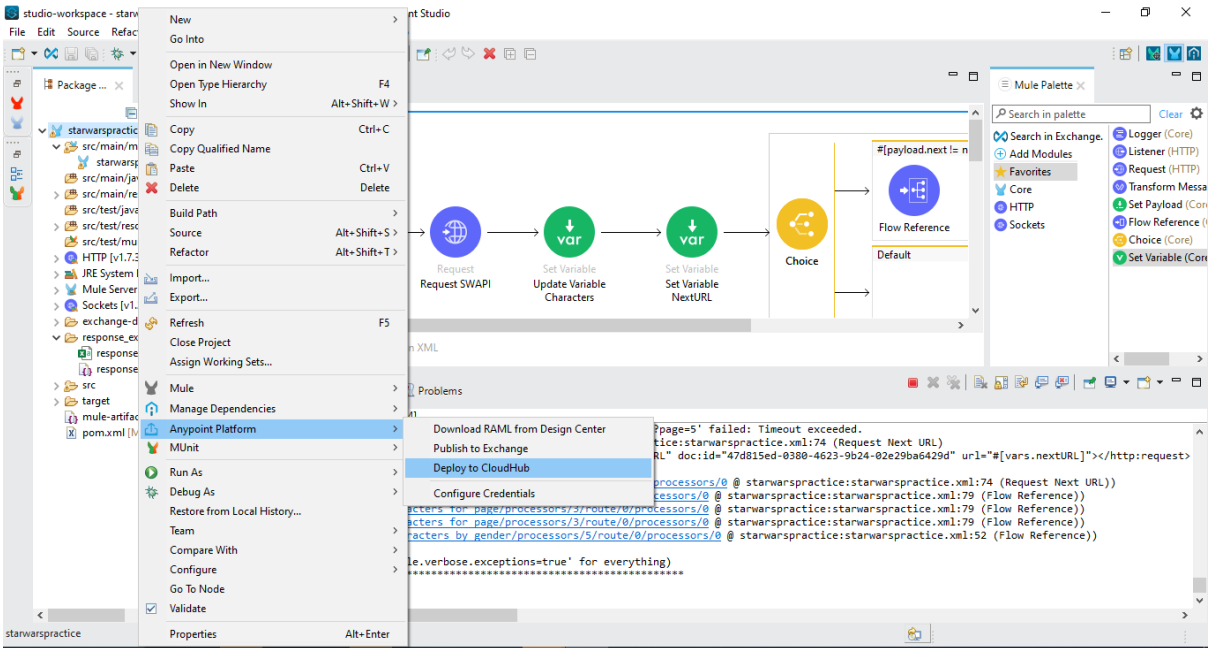
gender = n/a



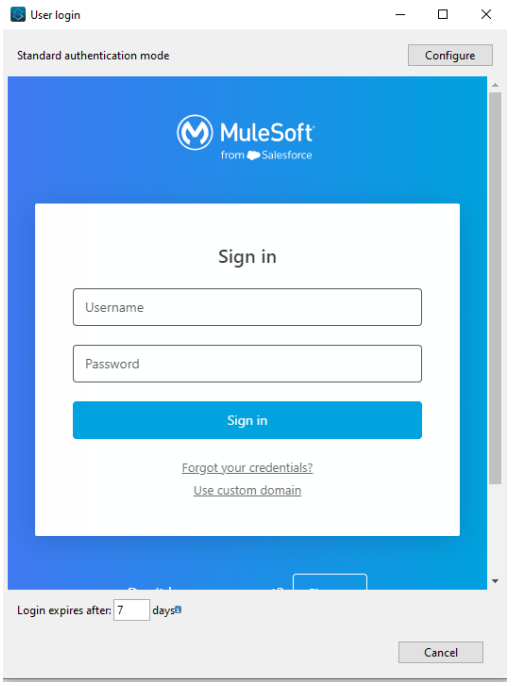
Deploy to CloudHub (Anypoint platform)

To deploy the project in CloudHub we follow the steps:

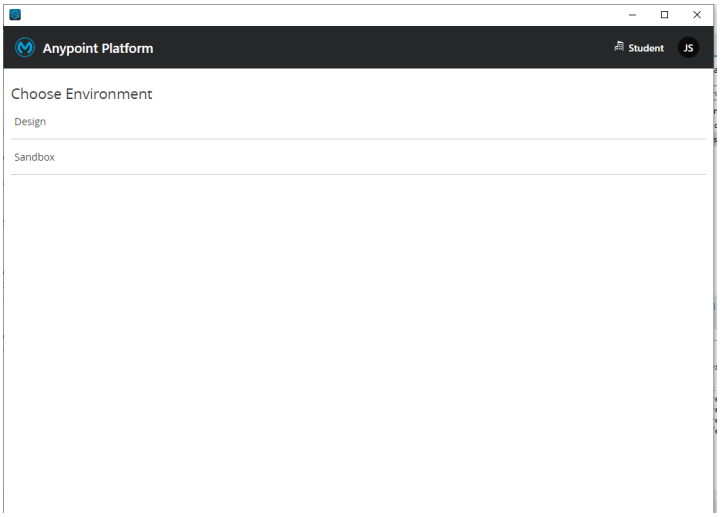
Right click on the project -> Anypoint Platform -> Deploy to CloudHub.



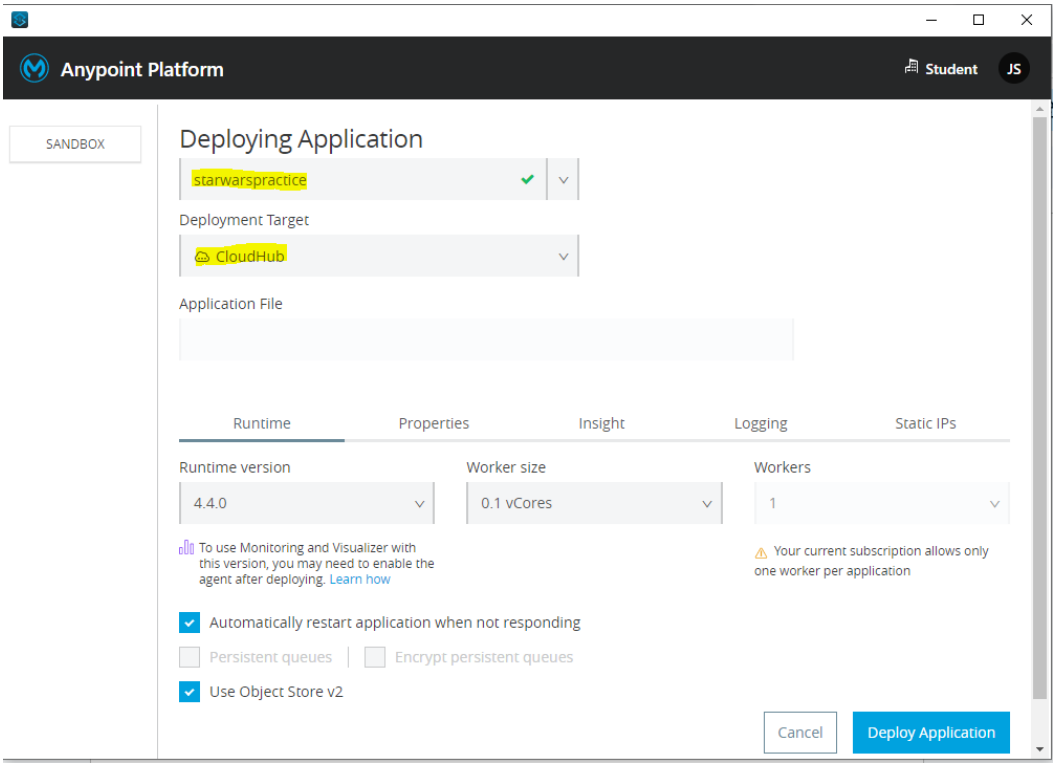
Log in with our Anypoint Platform account in the window that opens.



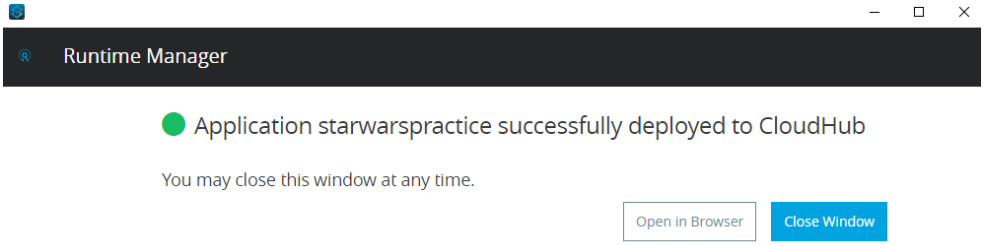
Choose *"Sandbox"* option



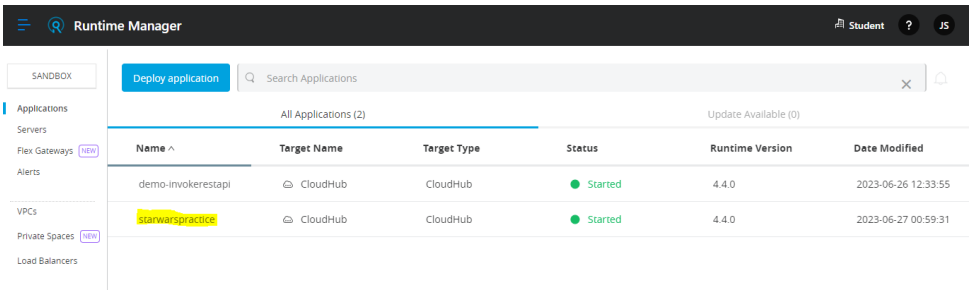
Assign the next properties



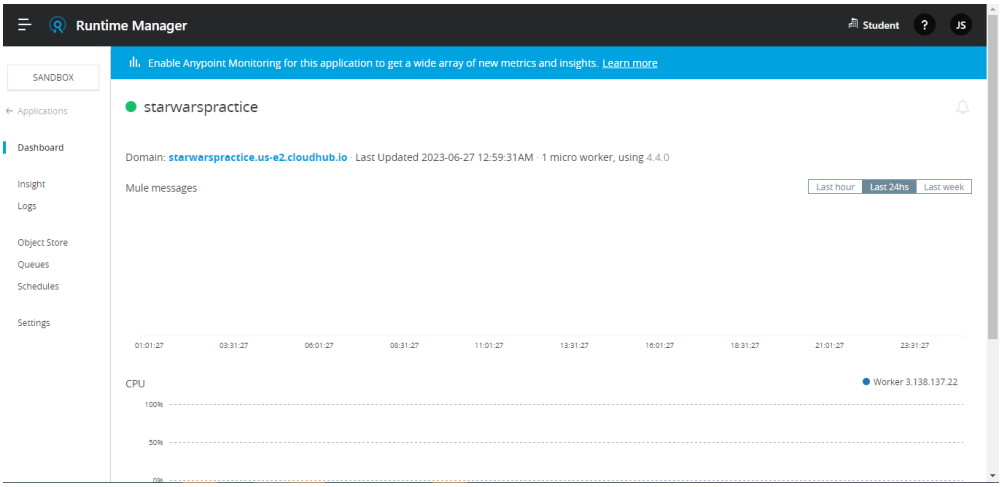
Wait to see the next message



In the "Runtime Manager" section of our Anypoint Platform session we can see our project deployed.

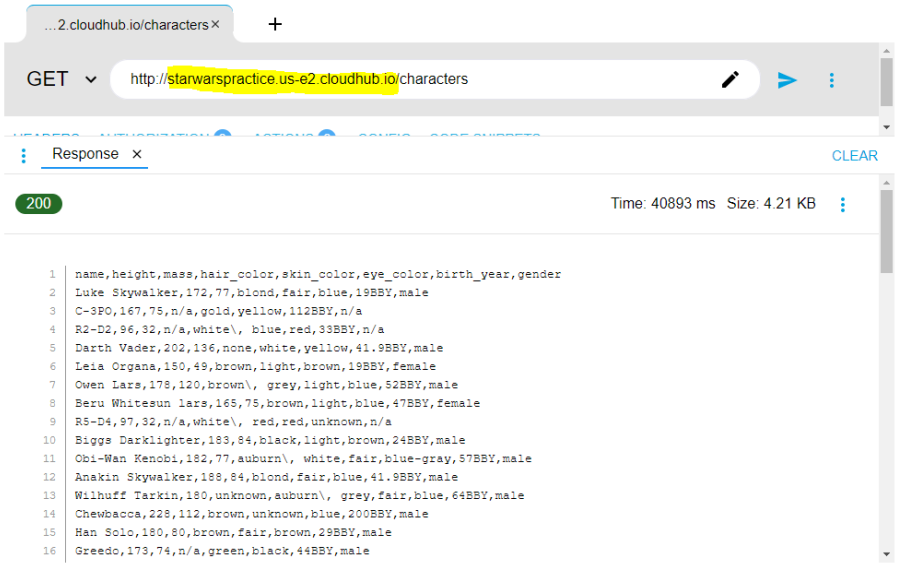


From here we can obtain our public endpoint:
http://starwarspractice.us-e2.cloudhub.io/characters

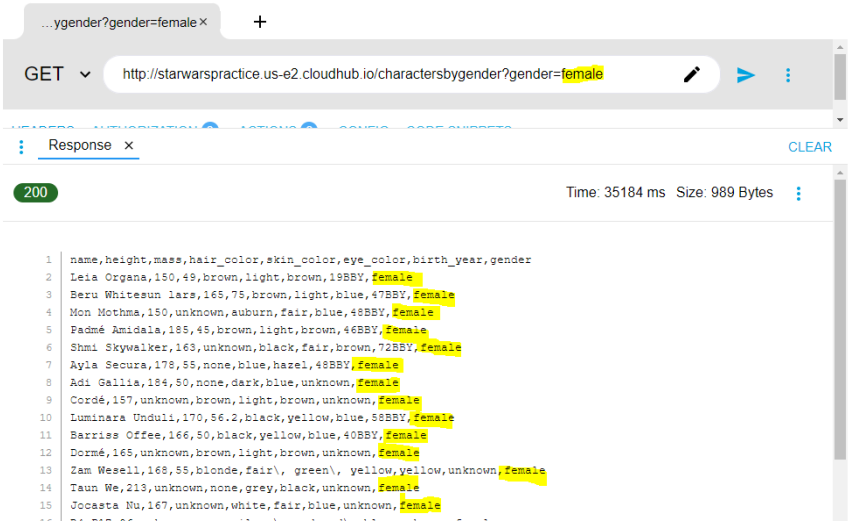


We can now make requests with the following URLs

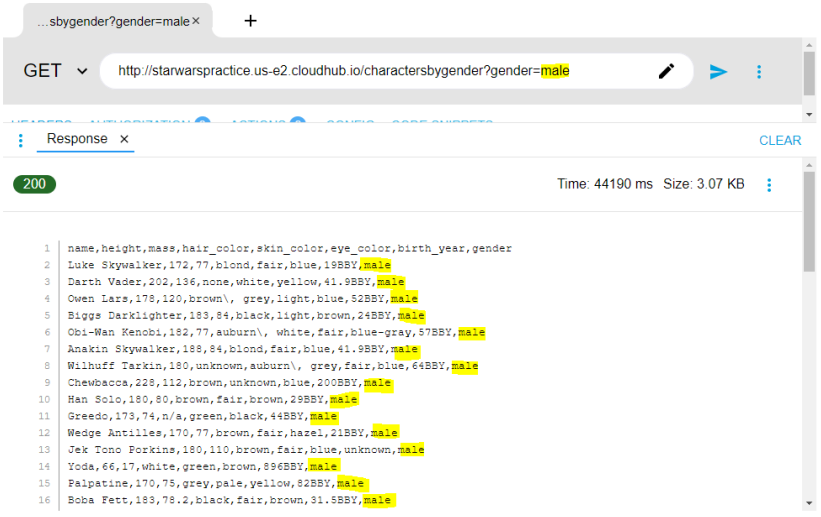
ALL CHARACTERS:



CHARACTERS BY GENDER (gender = female)



CHARACTERS BY GENDER (gender = female)



CHARACTERS BY GENDER (gender = n/a)