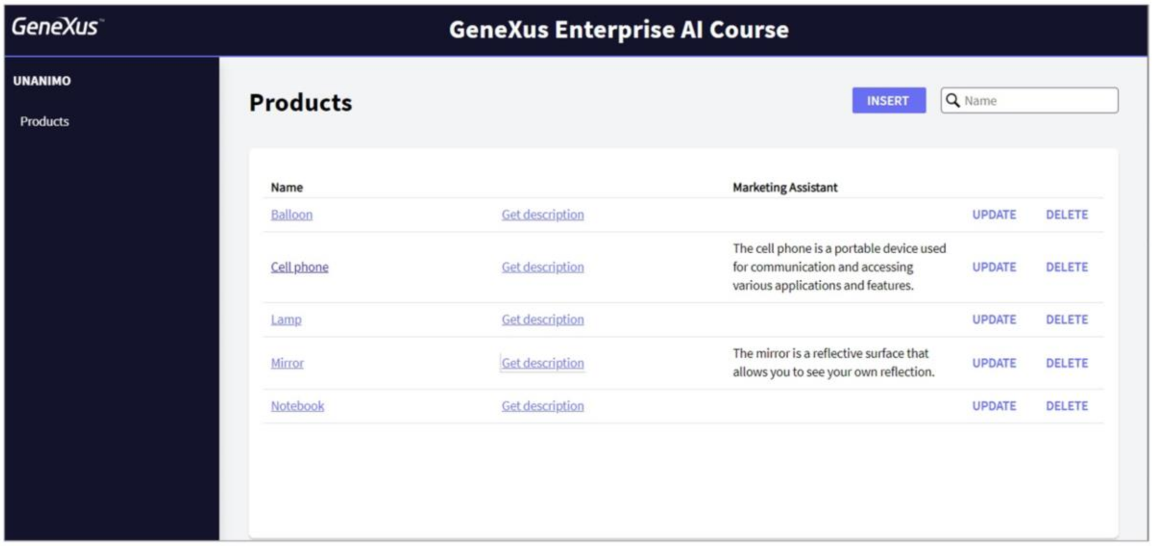
GeneXus by Globant

GeneXus application that interacts with a Chat Assistant

Alejandra Caggiano

At this point, we already know how to interact with GeneXus Enterprise AI, we know its Backoffice and Frontend, and we know how to create assistants and test them, either by editing the prompt, through the Playground or via API.

GeneXus application that interacts with a Chat Assistant

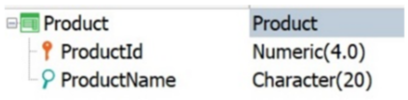


Products

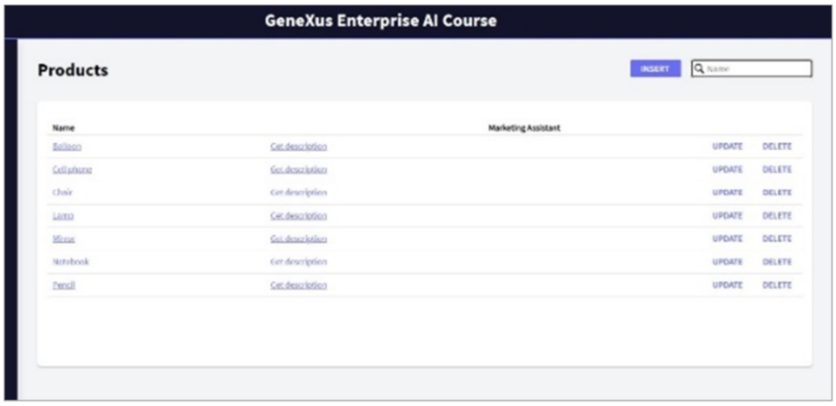
Name		Marketing Assistant		
Balloon	Get description		UPDATE	DELETE
Cell phone	Get description	The cell phone is a portable device used for communication and accessing various applications and features.	UPDATE	DELETE
Lamp	Get description		UPDATE	DELETE
Mirror	Get description	The mirror is a reflective surface that allows you to see your own reflection.	UPDATE	DELETE
Notebook	Get description		UPDATE	DELETE

Now let's see an example of how to interact with our assistants from a GeneXus knowledge base. The objective is to interact with our MarketingAssistant so that it returns the formal description of the product indicated by the user.

GeneXus application that interacts with a Chat Assistant



Product	Product
ProductId	Numeric(4.0)
ProductName	Character(20)



GeneXus Enterprise AI Course

Products

Name	Marketing Assistant		
Balloon	Get description	UPDATE	DELETE
Cellphone	Get description	UPDATE	DELETE
Chair	Get description	UPDATE	DELETE
Lamp	Get description	UPDATE	DELETE
Mirror	Get description	UPDATE	DELETE
Notebook	Get description	UPDATE	DELETE
Pencil	Get description	UPDATE	DELETE

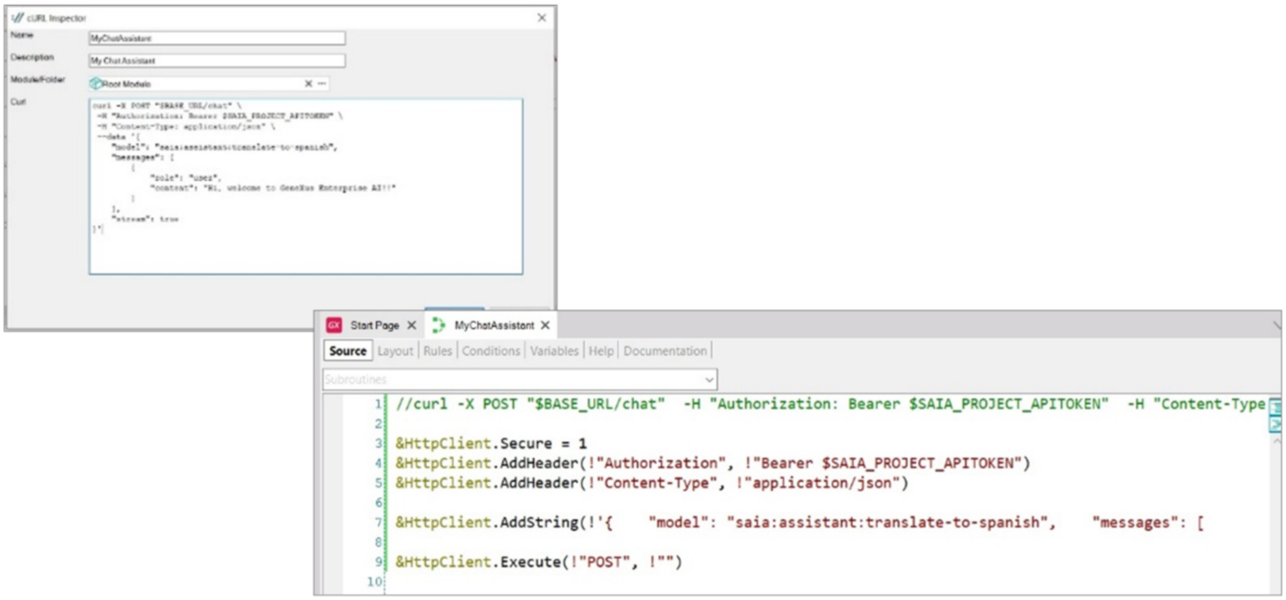
Good. In our KB we have a simple Product transaction, with an ID and Name to which we have applied the Work With for Web pattern.

Our objective is to request the description of the product from the main screen generated by applying this pattern.

For that, in the instance of the applied pattern, we have defined two variables:

- The &Description variable that the user will click on, and has the text “Get description”.
- And the &AssistantDescription variable that will receive the description returned by the assistant.

GeneXus application that interacts with a Chat Assistant


cURL Inspector
Start Page
MyChatAssistant
Source | Layout | Rules | Conditions | Variables | Help | Documentation
Subroutines

Name: MyChatAssistant
Description: My Chat Assistant
Module/Folder: Root Module
Curl:

```
curl -X POST "$BASE_URL/chat" \
 -H "Authorization: Bearer $SAIA_PROJECT_APITOKEN" \
 -H "Content-Type: application/json" \
 --data '{
    "model": "saia:assistant:translate-to-spanish",
    "messages": [
        {
            "role": "user",
            "content": "Hi, welcome to GeneXus Enterprise AI!!"
        }
    ],
    "stream": true
}'
```

```
//curl -X POST "$BASE_URL/chat"  -H "Authorization: Bearer $SAIA_PROJECT_APITOKEN"  -H "Content-Type

&HttpClient.Secure = 1
&HttpClient.AddHeader(!"Authorization", !"Bearer $SAIA_PROJECT_APITOKEN")
&HttpClient.AddHeader(!"Content-Type", !"application/json")

&HttpClient.AddString(!'{    "model": "saia:assistant:translate-to-spanish",    "messages": [

&HttpClient.Execute(!"POST", !"")

```

Good. The goal now is to define the procedure that will establish the interaction with the MarketingAssistant and will return the desired description.

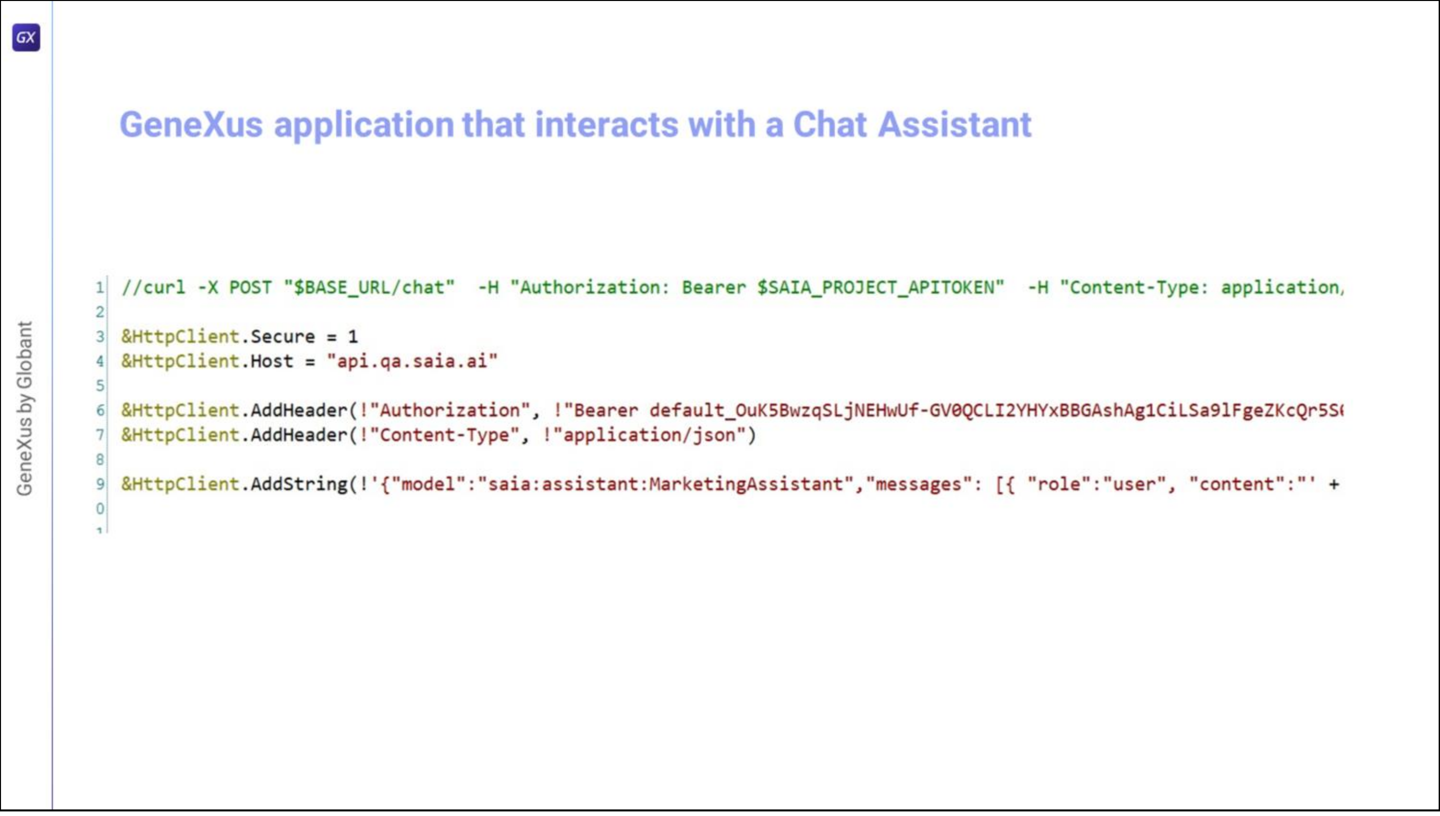
Although we could create a blank procedure and start defining the connection data through a variable of HttpClient type, what we will do is to give GeneXus the cURL sample that we need to make the POST and GeneXus will return the base structure for this definition.

We go to the Tool / Application Integration menu option, and choose cURL Inspector. We name it MyChatAssistant, and paste the sample, which is available, as we already know, in the GeneXus Enterprise AI technical documentation.

https://wiki.genexus.com/enterprise-ai/wiki?8,Table+of+contents%3AEnterprise+AI,

We click on OK, and in the KB Explorer window we can see that the procedure named MyChatAssistant has been created, and that it provides the basics of what we need:

A variable called &HttpClient has been defined based on the data type of the same name. This data type allows you to create a request, send it to a URL and read the results.



GeneXus application that interacts with a Chat Assistant

```
//curl -X POST "$BASE_URL/chat"  -H "Authorization: Bearer $SAIA_PROJECT_APITOKEN"  -H "Content-Type: application,

&HttpClient.Secure = 1
&HttpClient.Host = "api.qa.saia.ai"

&HttpClient.AddHeader(!"Authorization", !"Bearer default_OuK5BwzqSLjNEHwUf-GV0QCLI2YHYxBBGAshAg1CiLSa9lFgeZKcQr5S
&HttpClient.AddHeader(!"Content-Type", !"application/json")

&HttpClient.AddString(!'{"model":"saia:assistant:MarketingAssistant","messages": [{ "role":"user", "content":"' +
```

What should we do now?

Replace the parameters with the information of our context.

The first thing to specify is whether the call will be made using a secure protocol or not.

For this, we must declare the Secure method. The value 0, which is the default value, indicates that the HTTP protocol will be used, and the value 1 indicates that the HTTPS protocol will be used. We indicate the value 1.

Good. Next, we declare the Project API Token that we copied from the platform beforehand.

Now we'll focus on the Body of the request.

What we will do is similar to what we did from Postman when we tested the assistant via API.

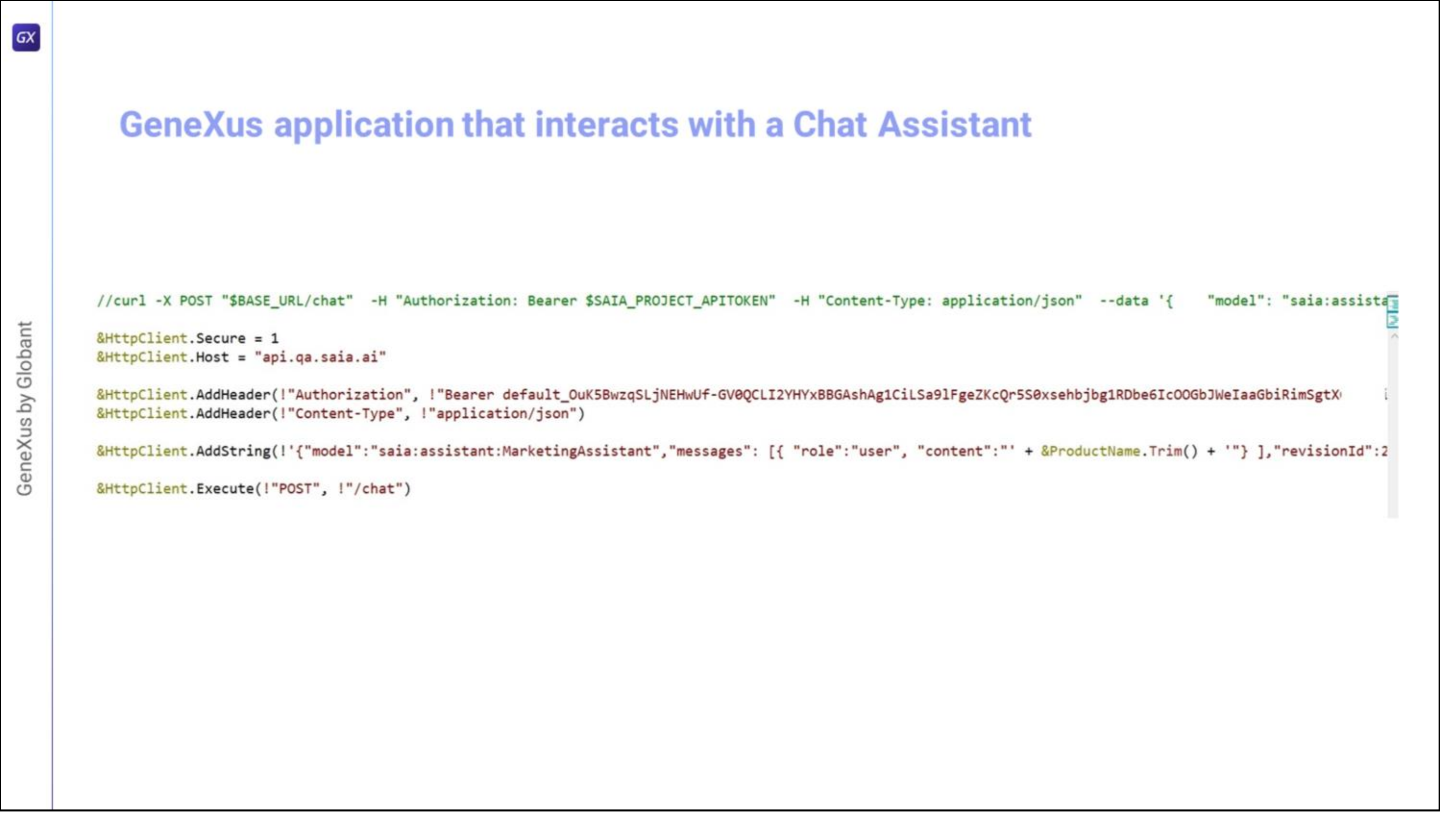
First of all, we indicate the name of the assistant, which in this case is MarketingAssistant.

Now let's think about this: Our assistant must return the description of the product specified by the user; therefore, the product must be received by parameter in this procedure.

Therefore, we declare the Parm rule so that the procedure receives the product name and returns the description.

We return to the source, and in the Messages group we see the “content” which, as we know, corresponds to the user input. We modify it so that it is parameterized and takes in each case the name of the product that is received by parameter.

We also indicate the revision of the assistant we want to use.



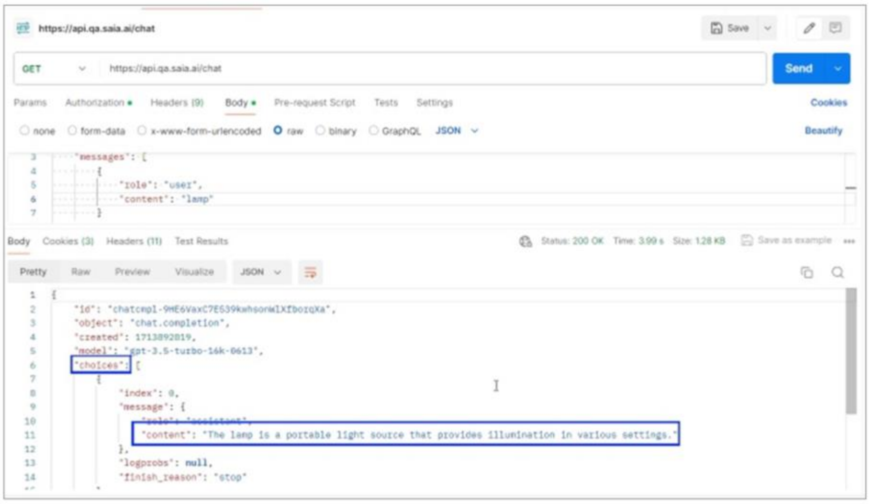
GeneXus application that interacts with a Chat Assistant

```
//curl -X POST "$BASE_URL/chat"  -H "Authorization: Bearer $SAIA_PROJECT_APITOKEN"  -H "Content-Type: application/json"  --data '{    "model": "saia:assista

&HttpClient.Secure = 1
&HttpClient.Host = "api.qa.saia.ai"

&HttpClient.AddHeader(!"Authorization", !"Bearer default_OuK5BwzqSLjNEHwUf-GV0QCLI2YHYxBBGAshAg1CiLSa9lFgeZKcQr5S0xsehbjbg1RDbe6IcOOGbJWeIaaGbiRimSgtX
&HttpClient.AddHeader(!"Content-Type", !"application/json")

&HttpClient.AddString(!'{"model":"saia:assistant:MarketingAssistant","messages": [{ "role":"user", "content":"' + &ProductName.Trim() + '"} ],"revisionId":2

&HttpClient.Execute(!"POST", !"/chat")
```

We must now define the execution of the POST, and for that we must complete the path with /chat.

We have already defined the execution of the query; what should we do now? Receive the response from the assistant and return only the product description.

GeneXus application that interacts with a Chat Assistant



```
    "messages": [
        {
            "role": "user",
            "content": "lamp"
        }
```

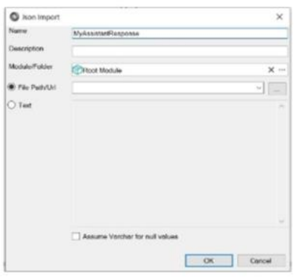
```
{
    "id": "chatcmpl-9HE6VaxC7E539kwhsonWlXfborqXa",
    "object": "chat.completion",
    "created": 1713892819,
    "model": "gpt-3.5-turbo-16k-0613",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "The lamp is a portable light source that provides illumination in various settings."
            },
            "logprobs": null,
            "finish_reason": "stop"
```

If we look at the execution of the query and the response received in Postman, we see that it looks like a structure, where the description we are interested in is stored in the "content" item of the first element of the Choices collection.

We save it as JSON.

GeneXus by Globant

GeneXus application that interacts with a Chat Assistant


Json Import

Name: MyAssistantResponse

Description:

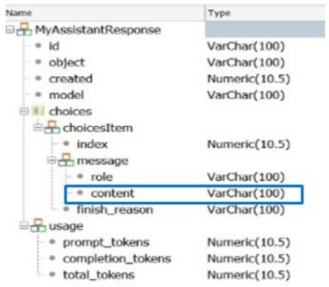
Module/Folder: Root Module

(●) File Path/Url

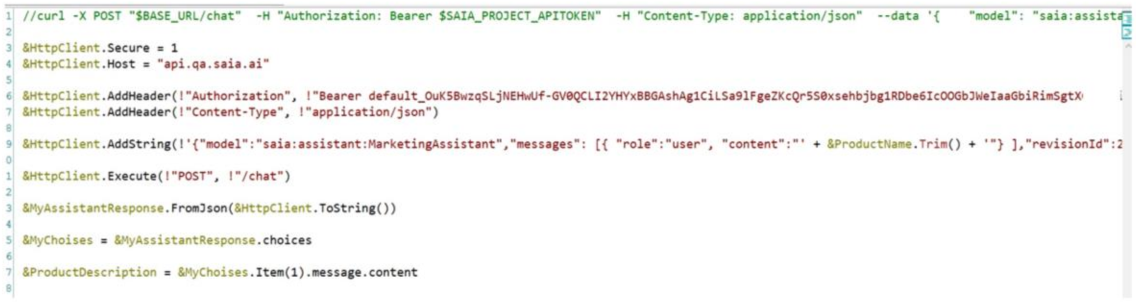
( ) Text

☐ Assume Varchar for null values

OK | Cancel



Name	Type
MyAssistantResponse	
id	VarChar(100)
object	VarChar(100)
created	Numeric(10.5)
model	VarChar(100)
choices	
choicesItem	
index	Numeric(10.5)
message	
role	VarChar(100)
content	VarChar(100)
finish_reason	VarChar(100)
usage	
prompt_tokens	Numeric(10.5)
completion_tokens	Numeric(10.5)
total_tokens	Numeric(10.5)



```
//curl -X POST "$BASE_URL/chat"  -H "Authorization: Bearer $SAIA_PROJECT_APITOKEN"  -H "Content-Type: application/json"  --data '{    "model": "saia:assista

&HttpClient.Secure = 1
&HttpClient.Host = "api.qa.saia.ai"

&HttpClient.AddHeader(!"Authorization", !"Bearer default_OuK5BwzqSLjNEHwUf-GV0QCLI2YHYxBBGAshAg1CiLSa9lFgeZKcQr5S0xsehbjbg1RDbe6IcOOGbJWeIaaGbiRimSgtX

&HttpClient.AddHeader(!"Content-Type", !"application/json")

&HttpClient.AddString(!'{"model":"saia:assistant:MarketingAssistant","messages": [{ "role":"user", "content":"' + &ProductName.Trim() + '"} ],"revisionId":2

&HttpClient.Execute(!"POST", !"/chat")

&MyAssistantResponse.FromJson(&HttpClient.ToString())

&MyChoises = &MyAssistantResponse.choices

&ProductDescription = &MyChoises.Item(1).message.content
```

We go back to our KB and import this JSON file so that GeneXus automatically creates the structured data type that represents it. For that, we go to Tools / Application Integration / Json Import.

We name it MyAssistantResponse and load the file.

We select OK and see the corresponding SDT created. Let's look at its structure. Here is the item we are looking for.

Then we define the &MyAssistantResponse variable based on this same data type... and load it with the response, applying the FromJson method that requires a string.

Let's look at the structured data again. To get to the “content” item we must retrieve the Choices collection.

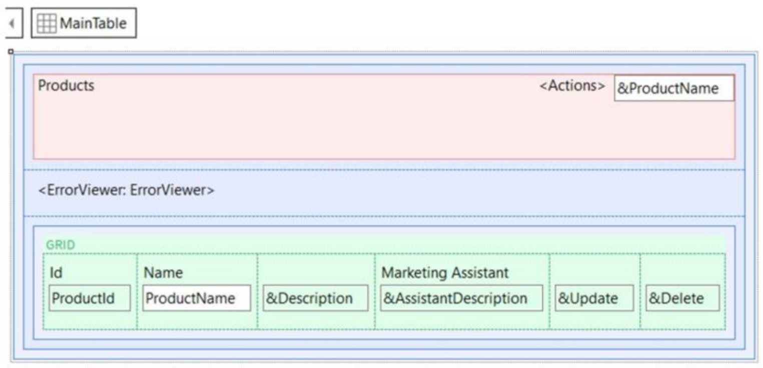
So we define the &MyChoices variable based on the data type &MyAssistantResponse.choicesItem.

It must be a collection, so we select the IsCollection checkbox.

We load the collection. Finally, in the &ProductDescription variable, which is the return variable of the procedure, we load the value of “content” of item 1 of the collection, which is where the answer we want from the assistant is stored.

GeneXus by Globant

GeneXus application that interacts with a Chat Assistant



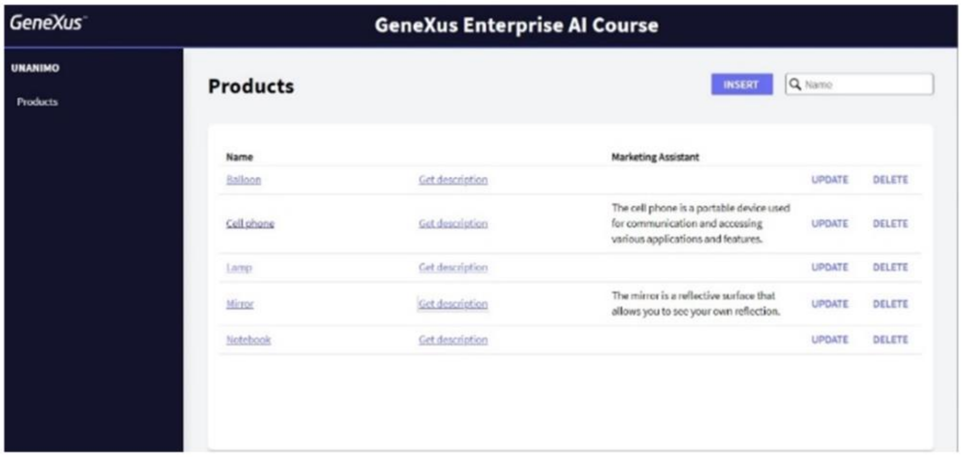
```
Event &Description.click()
      &AssistantDescription = MyChatAssistant(ProductName)
endevent
```

Good. What should we do now? Call this procedure from the WWProduct web panel.

To do so, in the events tab, we define the Click event applied to the &Description variable.

To test it, we press F5.

GeneXus application that interacts with a Chat Assistant



Products

Name		Marketing Assistant		
Balloon	Get description		UPDATE	DELETE
Cell phone	Get description	The cell phone is a portable device used for communication and accessing various applications and features.	UPDATE	DELETE
Lamp	Get description		UPDATE	DELETE
Mirror	Get description	The mirror is a reflective surface that allows you to see your own reflection.	UPDATE	DELETE
Notebook	Get description		UPDATE	DELETE

We run the web panel, choose a product and click on Get description.

Then we see the answer returned by the assistant.

GeneXus by Globant

GeneXus™ by Globant

training.genexus.com