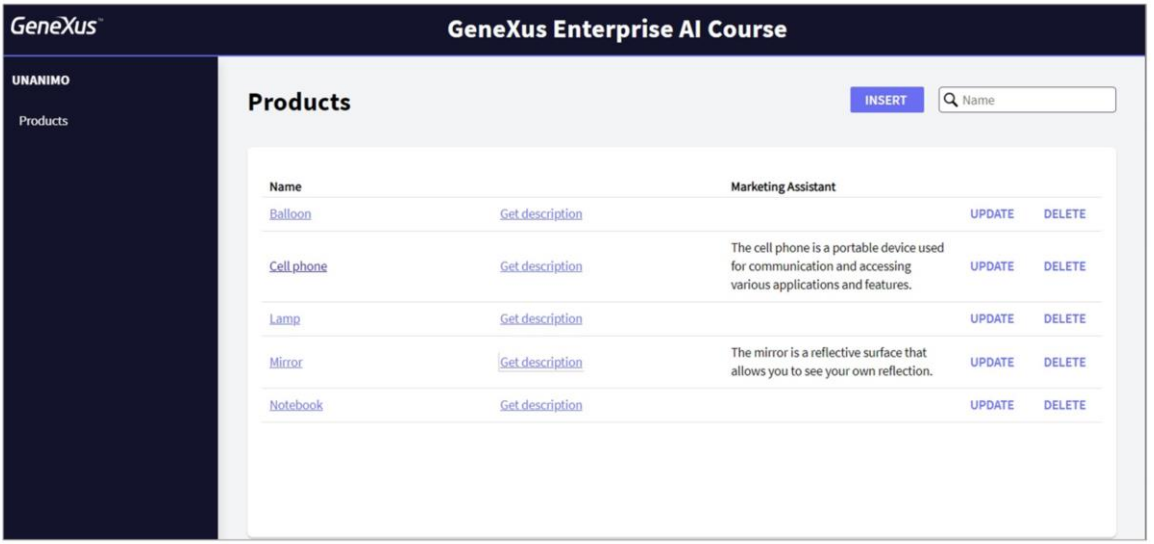
GeneXus by Globant

Aplicativo GX que interage com um Chat Assistant

Alejandra Caggiano

Neste ponto já sabemos interagir com GeneXus Enterprise AI, conhecemos seu Backoffice e seu Frontend, sabemos como criar assistentes e como testá-los, seja editando o prompt, através do Playground ou via api.

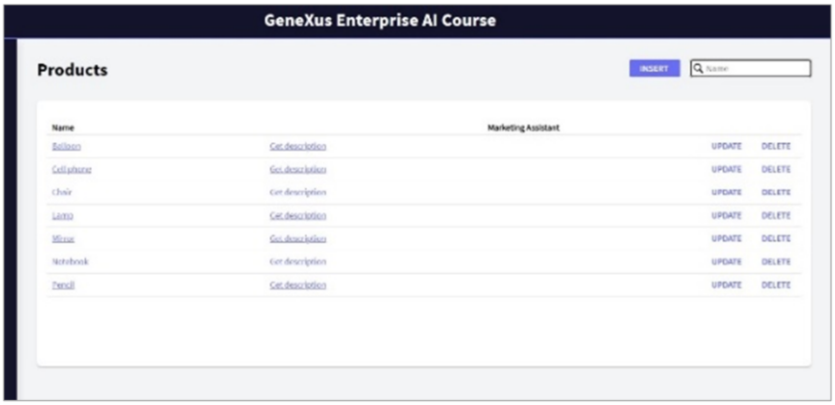
Aplicativo GX que interage com um Chat Assistant



Veremos agora um exemplo de uso para interagir com nossos assistentes a partir de uma base de conhecimento GeneXus. O objetivo é poder interagir com nosso MarketingAssistant para que retorne a descrição formal do produto indicado pelo usuário.

Aplicativo GX que interage com um Chat Assistant

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



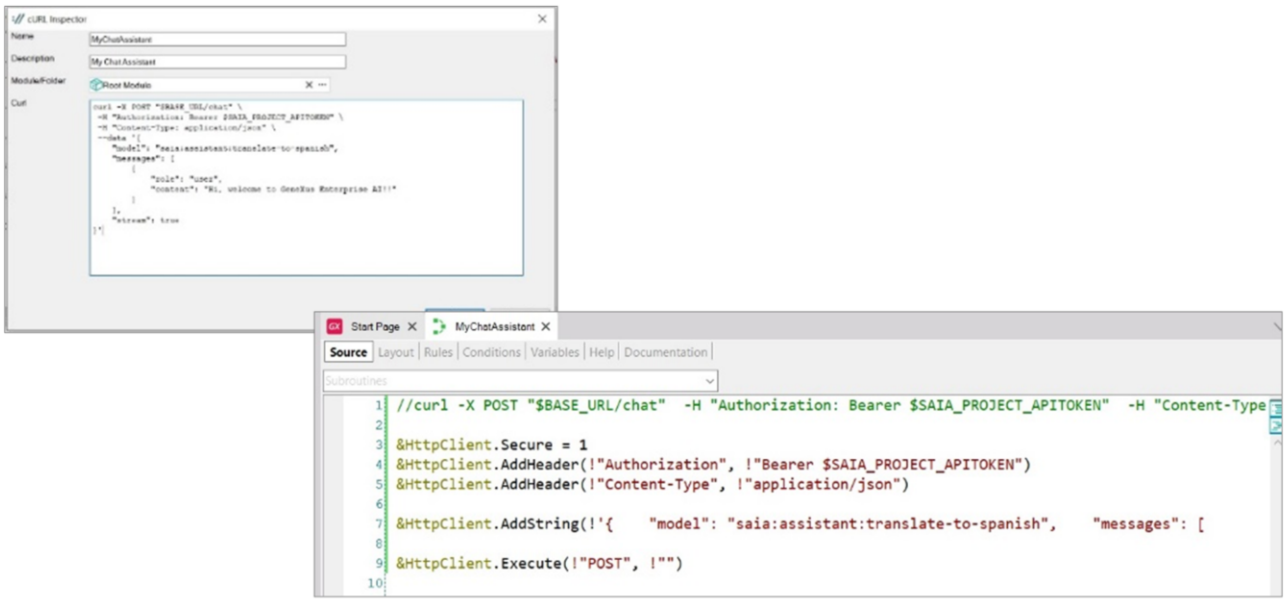
Bem. Em nossa KB temos a transação Produto, bem simples, com Id e Nome à qual aplicamos o pattern Work With for Web.

Nossa ideia é poder solicitar a descrição do produto a partir da tela principal gerada pela aplicação deste pattern.

Para isso, na instância do pattern aplicado, definimos duas variáveis:

- A variável &Description, que é sobre a qual o usuário irá clicar, e possui o texto “Get description”.
- E a variável &AssistantDescription que é a que receberá a descrição retornada pelo assistente.

Aplicativo GX que interage com um Chat Assistant



Bem. O objetivo agora é definir o procedimento que estabelecerá a interação com o assistente MarketingAssistant e retornará a descrição pesquisada.

Embora pudéssemos criar um procedimento em branco e começar a definir os dados de conexão através de uma variável do tipo HttpClient, o que faremos é fornecer ao GeneXus o cURL sample que precisamos para fazer o POST e GeneXus nos retornará a estrutura base para esta definição.

Vamos então para o menu Tools / Application Integration, e escolhemos cURL Inspector. Colocamos como nome MyChatAssistant e colamos o sample, disponível, como já sabemos, na documentação técnica de GeneXus Enterprise AI.

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

Pressionamos Ok e vemos na janela KB Explorer que foi criado o procedimento denominado MyChatAssistant, que já nos oferece a base do que necessitamos:

Vemos que foi definida uma variável &HttpClient baseada no tipo de dado de mesmo nome.

Este tipo de dado permite criar um request, uma solicitação, a envia para uma URL e lê os resultados.

Aplicativo GX que interage com um Chat Assistant

```
1 //curl -X POST "$BASE_URL/chat"  -H "Authorization: Bearer $SAIA_PROJECT_APITOKEN"  -H "Content-Type: application,
2
3 &HttpClient.Secure = 1
4 &HttpClient.Host = "api.qa.saia.ai"
5
6 &HttpClient.AddHeader(!"Authorization", !"Bearer default_OuK5BwzqSLjNEHwUf-GV0QCLI2YHYxBBGAshAg1CiLSa9lFgeZKcQr5S
7 &HttpClient.AddHeader(!"Content-Type", !"application/json")
8
9 &HttpClient.AddString(!'{"model":"saia:assistant:MarketingAssistant","messages": [{ "role":"user", "content":"' +
0
```

O que devemos fazer agora?

Substituir os parâmetros com a informação de nosso contexto.

A primeira coisa que devemos especificar é se a chamada será realizada através de um protocolo seguro ou não.

Para isso devemos declarar o método Secure. O valor 0, que é o valor padrão, indica que será utilizado o protocolo HTTP, e o valor 1 indica que será utilizado o protocolo HTTPS.

Indicamos então o valor 1.

Bem. Declaramos então o Project api token, que previamente, como já sabemos, o copiamos a partir da plataforma.

Vamos agora nos concentrar no Body, o corpo da solicitação.

O que faremos é semelhante ao que fizemos a partir do Postman quando testamos o assistente via api.

Em primeiro lugar indicamos o nome do assistente, que neste caso é MarketingAssistant.

Vamos pensar agora no seguinte. Nosso assistente deve retornar a

descrição do produto que o usuário indique, portanto, o produto deve ser recebido por parâmetro neste procedimento.

Então declaramos a regra Parm para que o procedimento receba o nome do produto e retorne a descrição.

Voltamos ao source, e no grupo Messages vemos o “content” que, como já sabemos, corresponde ao input do usuário. O modificamos para que fique parametrizado e tome em cada caso o nome do produto que é recebido por parâmetro.

Indicamos também a revisão do assistente que queremos utilizar.

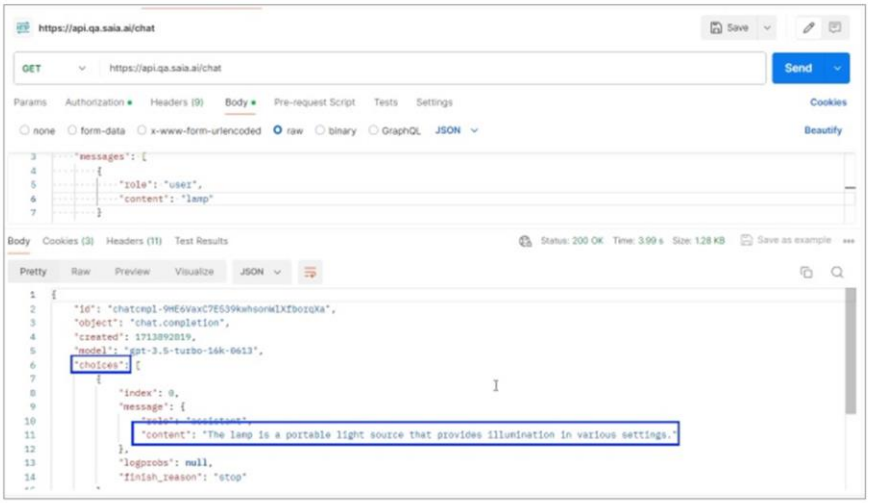
Aplicativo GX que interage com um Chat Assistant

```
//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")
```

Devemos agora definir a execução do POST, e para isso devemos completar o path com /chat.

Bem. Já temos definida a execução da consulta. O que devemos fazer agora? Receber a resposta do assistente e retornar apenas a descrição do produto.

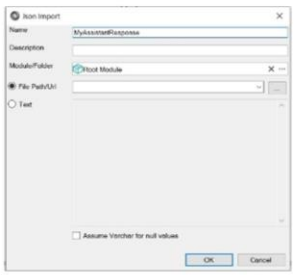
Aplicativo GX que interage com um Chat Assistant

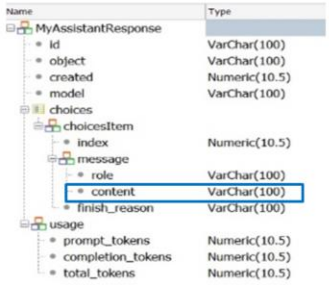


Se observarmos no Postman a execução da consulta e a resposta recebida, vemos que tem a forma de uma estrutura, onde a descrição que nos interessa fica armazenada no item “content”, do primeiro elemento da coleção Choices.

A salvamos então como json.

Aplicativo GX que interage com um 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")

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

&MyChoises = &MyAssistantResponse.choices

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

Voltamos à nossa KB e importamos este arquivo json para que GeneXus crie automaticamente o tipo de dado estruturado que o representa. Para isso vamos para Tools / Application Integration / Json import.

Colocamos como nome MyAssistantResponse e carregamos o arquivo.

Pressionamos Ok e vemos criado o sdt correspondente. Observemos sua estrutura. Temos aqui o item que estamos procurando.

Definimos então a variável &MyAssistantResponse baseada neste mesmo tipo de dado….. e a carregamos com a resposta, aplicando o método FromJson, que por sua vez requer uma string.

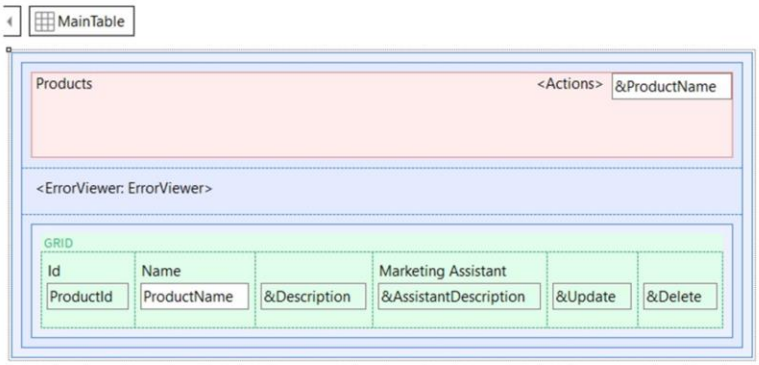
Observemos novamente o dado estruturado. Para chegar ao item “content” devemos recuperar a coleção Choices.

Portanto, definimos a variável &MyChoices baseada no tipo de dado &MyAssistantResponse.choicesItem.

Deve ser uma coleção, por isso marcamos a caixa IsCollection. Carregamos a coleção

Finalmente, na variável &ProductDescription, que é a variável de retorno do procedimento, carregamos o valor de “content” do item 1 da coleção, que é onde é salva a resposta que procuramos do assistente.

Aplicativo GX que interage com um Chat Assistant



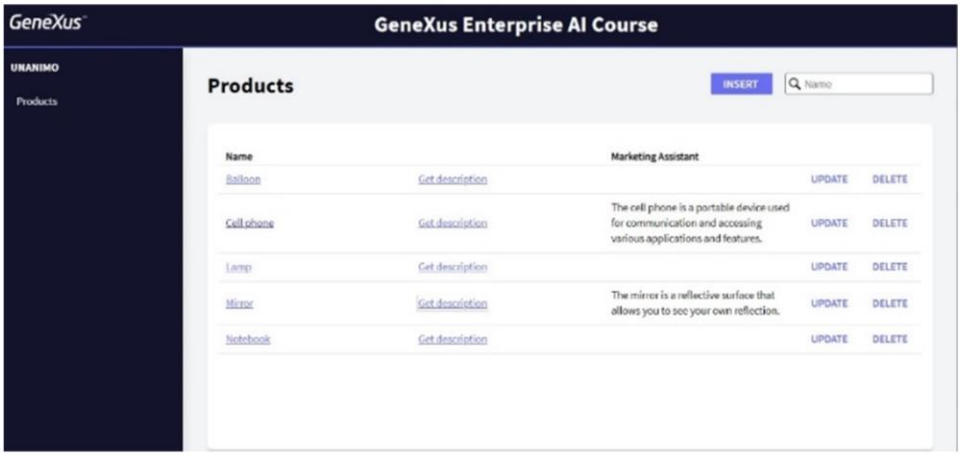

```
Event &Description.click()
    &AssistantDescription = MyChatAssistant(ProductName)
endevent
```

Bem. O que devemos fazer agora? Chamar este procedimento a partir do web panel WWProduct.

Para isso, na aba de eventos, definimos o evento Click aplicado à variável &Description.

Para testar pressionamos F5.

Aplicativo GX que interage com um Chat Assistant



Executamos o web panel. Escolhemos um produto e clicamos em Get description.

Vemos então a resposta retornada pelo assistente.

GeneXus by Globant

GeneXus™
by Globant

training.genexus.com