Demonstrations

These are working examples of calling (Chat)GPT from OmniScript using LLMkit. The presume you have a named credential called OpenAI that will provide the OpenAI API Key. If you're not familiar with how to do that, please see this newsletter article on the topic.

The demonstrations available are:

Demonstration	Description
LifeIns	Demonstrates recommending the best type of life insurance for a customer based upon their description of their needs.
ChatCPQ	Demonstrates using GPT-4 to validate that an order is properly configured.

Each demonstration has its own README with instructions on installation and running.

LifeIns

This demonstration was designed to illustrate a couple of concepts:

1. The System prompt asks for a response not as free text, but in a JSON structure that is defined by the prompt.
2. It demonstrates that you can use GPT's "knowledge" to perform a "zero-shot" analysis

The basic flow of the demo is that a user types in an explanation for why they think they need Life Insurance, and the system responds with a specific recommendation.

Installation

The demo consists of an OmniScript and a "service definition" JSON file.

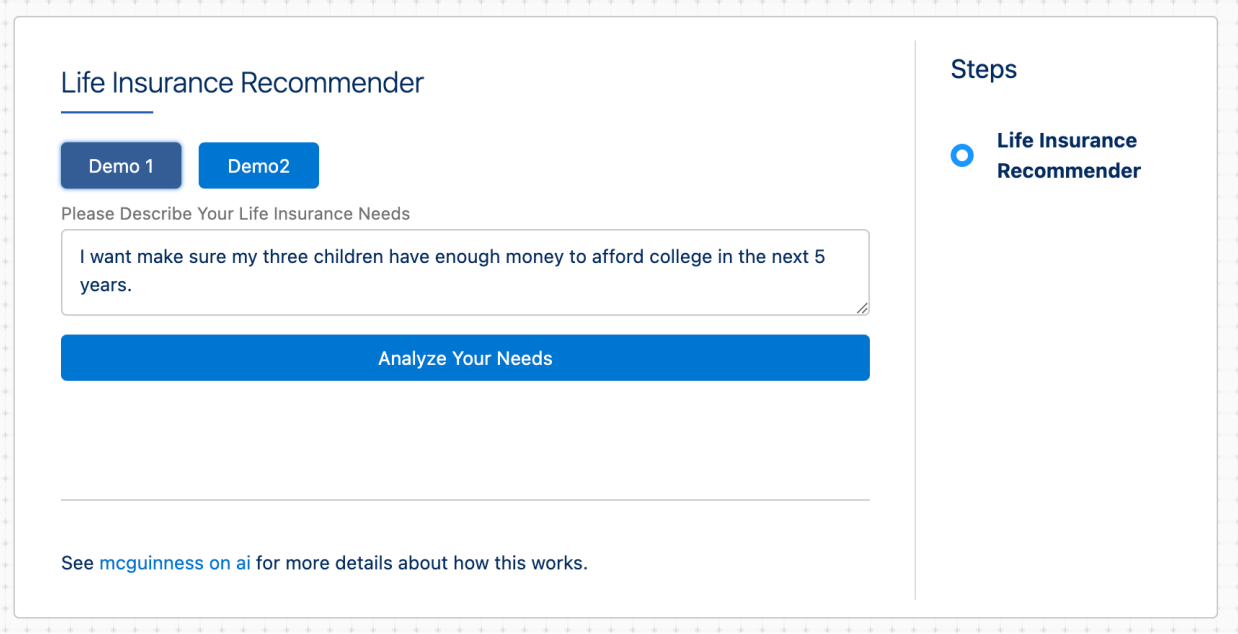
The OmniScript needs to be imported into your environment (pick either the *Life Ins Recommender OmniStudio.json* or *Life Ins Recommender Vlocity.json* file depending upon the namespace used in your org) but do not activate it.

For the service definition file, you will need to create a new static resource called LifeInsService and upload the file LifeInsService.json to it.

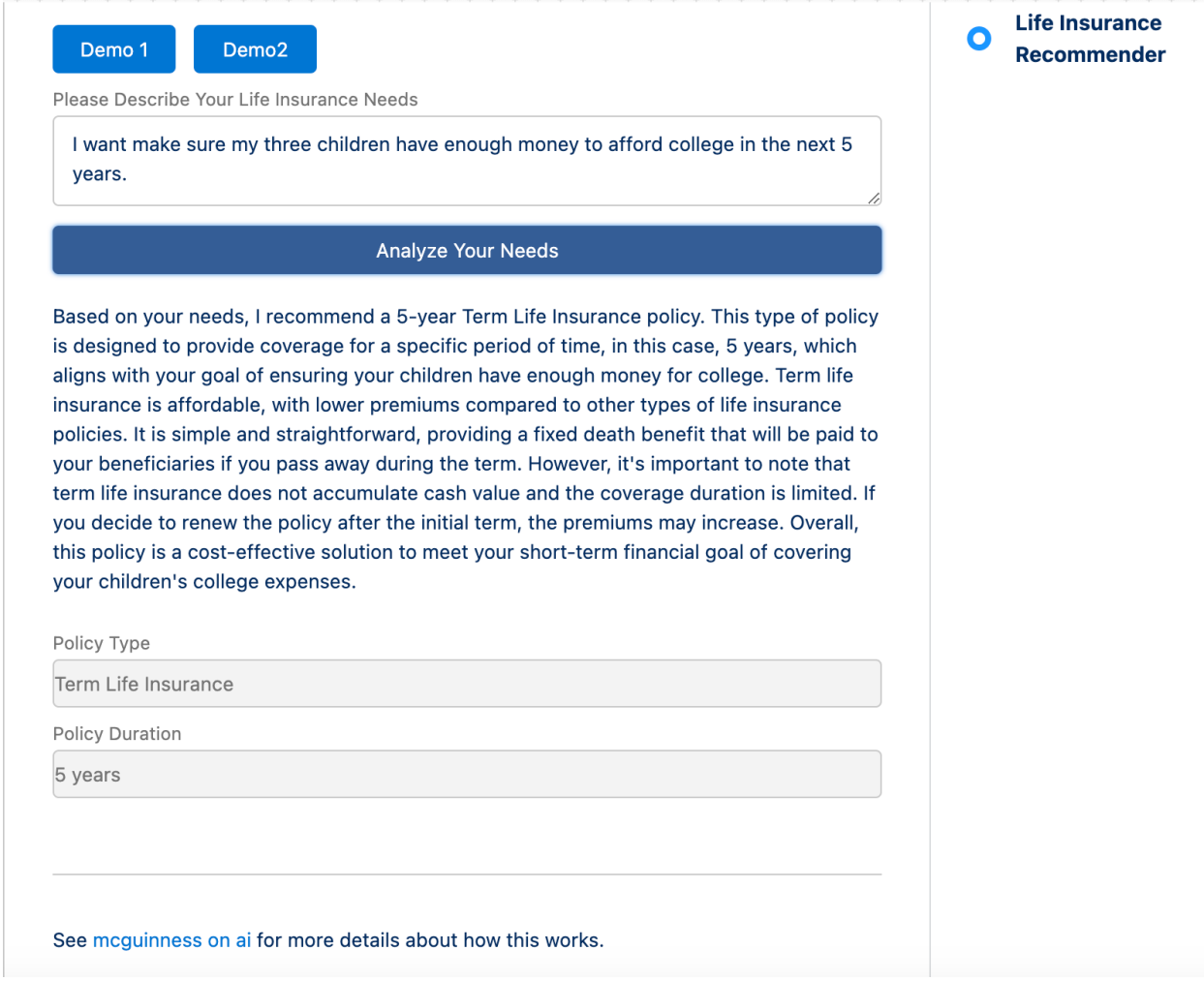
Apart from that, you will need to have a named credential called "OpenAI" for the code to work.

Running the Demo

When you run the OmniScript, you are presented a screen like this:



You can enter any text you like or pick one of two Demo texts to populate the text box. Press Analyze Your Needs to see the response:



Demo Notes

Here are some things to notice:

- The templates used in this demo are contained in the service definition file you uploaded as a static resource
- The Analyze Your Needs button is a remote action which calls the LLMkit and indicates which service definition to use.
- This is not extensively tested, and may produce incorrect answers if stressed. Take that as a challenge to improve the prompts if you like.
- The reponse is parsed as JSON before it is returned to the OmniScript, which is how the answer shows up in multiple (and correct) fields. The service definition file is where the instruction to parse the results is given.

Service.json file

```
{
    "system_template": "You are an experienced life insurance agent.  It is your job to recommend the best policy type for your customer, based upon their needs. In addition, for the type of policy you recommend, you should describe it's pluses and minuses in detail and be explicit about how it works. A customer will tell you what they need in the way of life insurance.   From their needs, you should be able to understand how long they need insurance for, what the purpose is, and whether they are looking for the cheapest insurance or one that is more expensive but with more value. Your response should be in JSON, and include the following elements:* policy_type* policy_duration* purpose* pluses, an array of advantages* minuses, an array of disadvantages* full_text, the complete explanation of all elements of the recommendation",
    "user_template": "{{ stepOne.taNeeds }}",
    "model": "gpt-4",
    "max_tokens": 1024,
    "named_credential": "OpenAI",
    "json_response": "true",
    "temperature": "0.0"
  }
```

ChatCPQ

This demonstrates a simplistic CPQ running on ChatGPT. The rules are thus:

- Widgets come in black, white, red, blue, green, cyan, yellow, and magenta.
- You are free to order as many cyan, yellow, and magenta widgets as you like, but for each one, you must also buy two primary colored widgets: cyan=blue+green, magenta=blue+red, yellow=red+green
- You are free to order as many additional primary colored widgets as you like above the minimum required by the secondary colored ones.
- You can buy as many black and white widgets as you like.

Installation

The demo consists of an OmniScript and a "service definition" JSON file.

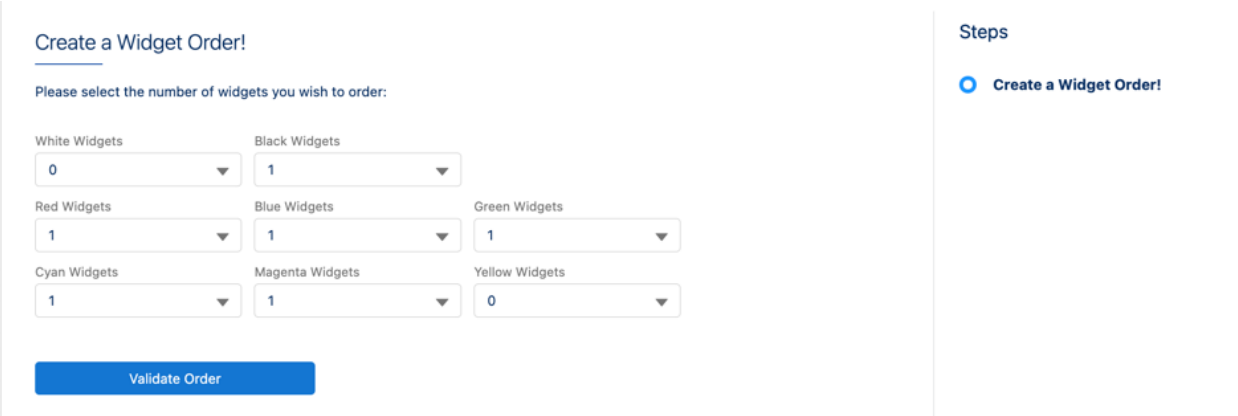
The OmniScript needs to be imported into your environment (pick either the *ChatCPQ OmniStudio.json* or *ChatCPQ Vlocity.json* file depending upon the namespace used in your org) but do not activate it.

For the service definition file, you will need to create a new static resource called ChatCPQ and upload the file ChatCPQ_service.json to it.

Apart from that, you will need to have a named credential called "OpenAI" for the code to work.

Running the Demo

When you run the OmniScript, you are presented a screen like this:



You can pick the quantities you wish to test, or there's a demo button (not shown) that you can click as well. When the quantities are right, press Validate Order. It is typical that it can take up to a minute for the response to occur. When it does, you should see something like:

Let's check if this order follows the rules by looking at each of the secondary colors and adding up the required primary colors.

1. Cyan: 1
   For every cyan widget, the customer must also buy one more blue and one more green widget.
   Required: Blue: 1, Green: 1

2. Magenta: 1
   For every magenta widget, the customer must also buy an additional red and a blue widget.
   Required: Red: 1, Blue: 1

3. Yellow: 0
   Since no yellow widgets are being purchased, there are no additional requirements for red and green widgets.

Now, let's add up the required primary colors:
Red: 1
Blue: 2
Green: 1

Now, let's compare the required primary colors to the ordered primary colors:
Ordered: Red: 1, Blue: 1, Green: 1
Required: Red: 1, Blue: 2, Green: 1

The order does not follow the rules because the customer needs to buy at least 2 blue widgets to meet the requirements for purchasing cyan and magenta widgets. The customer has only ordered 1 blue widget.

Demo Notes

Here are some things to notice:

- The templates used in this demo are contained in the service definition file you uploaded as a static resource
- The Valid Order button is a remote action which calls the LLMkit and indicates which service definition to use.
- This is not extensively tested, and may produce incorrect answers if stressed. Take that as a challenge to improve the prompts if you like.

service.json file

```
{
    "named_credential": "OpenAI",
    "system_template": "You are an order management specialist at a widget factory.  \nThere are 8 different types of widgets: \n\n* The primary colors: red, blue, and green\n* The secondary colors: yellow, magenta, and cyan\n* Two desaturated colors: black, and white.\n\nFor every cyan widget someone buys, they must also buy one more blue and one more green widget.\n\nFor every yellow widget someone buys, they must also buy an additional red and green widget.\n\nFor every magenta widget someone buys, they must also buy an additional red and a blue widget.\n\nThe widgets in red, blue, green, black, and white have no restrictions on them.\n\nThe required additional widgets are additive. Thus, if you buy two secondary colored widgets, \nyou have to buy the corresponding four primary colored widgets to meet the requirements.\n\nFor example, if you buy yellow and magenta, you will must but two reds, one green, and one blue in your order.\n\nYour job is to look at an order a customer wants to make and to verify that it follows the rules.\n\nThe way to do this is to look at each of the secondary colors, add up the required primary colors,\nand make sure the order has at least that many of each primary color.\n \nExplain your reasoning step by step.",
    "user_template": "I would like to buy widgets in the following colors and quantities:\n\nWhite: {{ Step1.selWhite }}\nBlack: {{ Step1.selBlack }}\nRed: {{ Step1.selRed }}\nBlue: {{ Step1.selBlue }}\nGreen: {{ Step1.selGreen }}\nCyan: {{ Step1.selCyan }}\nMagenta: {{ Step1.selMagenta }}\nYellow: {{ Step1.selYellow }}\n\nIs that allowed?",
    "model": "gpt-4",
    "temperature": "0.0"
}
```