OpenAI | WeWeb Documentation

|  |  |
| --- | --- |
| URL source | <https://docs.weweb.io/plugins/extensions/open-ai.html> |
| Date de scraping | 2025-04-08T15:55:41.409786+00:00 |

## OpenAI ​

You can find the OpenAI plugin in Plugins > Extensions :  
  
With this plugin, you will be able to tap into all of OpenAI's products, including but not limited to:  
text completion, image generation, and chat completion, better known as chatGPT.  
In this article, we will explain how to setup the OpenAI plugin in WeWeb and give you a few real life examples to get you started.  
To fully leverage the WeWeb OpenAI plugin, make sure to take the time to learn more about how OpenAI's APIs work .

### Images

<https://docs.weweb.io/assets/open-ai-1.Bi9V6us-.png>

## Plugin setup ​

When you add the OpenAI plugin to WeWeb, you will be invited to enter your OpenAI API key and, if you wish, you can add prompts at plugin level:  
  
You may wonder why we added prompts in the plugin configuration. The answer is simple: this ensures your prompts are secure.  
Indeed, you could make API calls to OpenAI via our REST API plugin but you really shouldn't.  
Why?  
Because the calls you make with the REST API plugin are visible in the inspector of the user's browser.  
This is completely normal by the way.  
However, when you make an API call that requires a secret API key or includes sensitive data you want to keep private, it's crucial that you make calls through a backend.  
In the case of OpenAI, there are 2 pieces of information you want to key private:  
your secret API key, and your prompts.

### Images

<https://docs.weweb.io/assets/open-ai-2.DpcKQ9P9.png>

## Secured prompts ​

When you build a product on top of AI, the value of the product is in the work you put in designing the system and prompts.  
That's why it's important that you protect that intelligence by adding your prompts at plugin level.  
Let's compare an API call made with the REST API plugin vs an API call made using WeWeb's native integration with OpenAI.  
First, let's make a request to OpenAI with a standard frontend API call:  
In the example above, you can see:  
in the Network tab of the user's browser, you can find the API request, look at the payload of the request, see the prompt we sent to OpenAI, and the private API key linked to our OpenAI account.  
Now, let's make a request to OpenAI using WeWeb's native integration.  
First, we'll want to define the prompts we want to keep private at plugin level:  
Then, we'll make the request using an OpenAI action in a Workflow:  
In the example above, you can see:  
in the Network tab of the user's browser, you can find the API request, look at the payload of the request, see the id of the prompt we configured at plugin level, but not its content  
So if we had put in a lot of work coming up with "Propose a name for a {fill\_in\_the\_blank}" to make our product unique, we would be very happy we protected that prompt.  
Note that you can see the animal variable we included in our prompt and the value of that variable. That's completely fine since the user just filled out that information themselves. It's not confidential or proprietary information.  
Ok, now that we're clear on why we should keep our prompts secure, let's learn how to use OpenAI actions in Workflows.

### Images

<https://docs.weweb.io/assets/open-ai-16.31MfG7zN.png>

<https://docs.weweb.io/assets/open-ai-17.DOsF3Suq.png>

<https://docs.weweb.io/assets/open-ai-18.Bf4UnSPk.png>

## Text completion example ​

First, let's have a look at the text completion product.  
How does it work?  
In the words of OpenAI: "You input some text as a prompt, and the model will generate a text completion that attempts to match whatever context or pattern you gave it."  
You can ask a simple question like "Write a tagline for an ice cream shop" and it will return something like the answer in highlighted below:  
  
But to leverage the true power of OpenAI, you'll often want to spend some time writing the perfect prompt with detailed instructions on the how you want the answer to look.  
Let's look at this example:  
  
We go to the Open AI API playground We choose the Complete mode to use text completion We choose the text-davinci-003 model because it's the most recent at the time of writing We set the Temperature to 0 on a scale from 0 to 2 because we want the AI to be mathematical, not creative at all We set the Maximum length of the answer to 2000 characters because we don't want the answer to be restricted to the default 256 We write a prompt that asks OpenAI to return a regular expression patterns for emails in a certain format that we can easily reuse in our app to validate user inputs in an email field, i.e. the regular expression itself and an explanation of how it works We click on Submit Open AI returns an answer  
As you can see, a lot of thought went into the prompt:  
we said we wanted users to input email addresses we asked OpenAI to return the regular expression pattern in a specific format we asked OpenAI to include a text to explain how the regex was built  
So how would it work in WeWeb?  
Well, you would take the prompt you engineered in the OpenAI playground and copy it in the plugin:  
  
To add a prompt, you'll need to click on Add item . For each prompt, you can add as many variables as you want.  
When you click on Add variable , a new variable will be added to your prompt content with the default {{var1}} value:  
  
You'll want to give it a more descriptive name so that, when you build your API request to OpenAI, you remember what value is expected.  
In our Return Regex prompt above, we had a variable named {{need}} .  
When we configure the API call to OpenAI in a Workflow, it's easy for us to replicated what we had in the OpenAI API playground:  
  
In the example above, you can see:  
we're making a request to OpenAI's Complete API based on the text-davinci-003 model using the Return Regex prompt we configured in the plugin binding the {{need}} variable we set in the plugin to a variable on our page with a maximum token of 2000 to ensure we get a reply, and a temperature of 0 to ensure that the reply is not creative  
To learn more about how to work with these settings, please refer to the OpenAI API documentation .

### Images

<https://docs.weweb.io/assets/open-ai-4.D1YSpxDN.png>

<https://docs.weweb.io/assets/open-ai-3.CRwKXHn4.png>

<https://docs.weweb.io/assets/open-ai-5.BtABabRU.png>

<https://docs.weweb.io/assets/open-ai-6.BX7Wi6oS.png>

<https://docs.weweb.io/assets/open-ai-7.CE0mN6lP.png>

## Chat completion example ​

Now let's look at an example with OpenAI's chat completion features 🙂  
  
Back in the OpenAI playground, we:  
changed the Mode to Chat gave the AI a personality and a few instructions simulated a user's first question submitted the question to get an answer from Open AI based on the system's answer, we decided to change a few settings, we chose the latest Model available to us, in this case gpt-4 increased the Temperature to 0.9 so the AI's answer would be more creative decreased the Maximum length to 150 tokens so the answer wouldn't be too long (it's not nice to scroll when you need a screenshot for the user docs ^^)

### Images

<https://docs.weweb.io/assets/open-ai-9.CZoinkXY.png>

## Chat completion system configuration in WeWeb ​

Once you're happy with your setup in the OpenAI playground, you can go back to the OpenAI plugin in WeWeb.  
Contrary to the Text completion use case, you won't need to add any prompts at plugin level since the end-users of the chat you build will be asking their own questions.  
However, you'll need to add the instructions you came up with to define the AI's System . In the example below, we copy / pasted the personality and instructions we want the chatGPT to follow:

### Images

<https://docs.weweb.io/assets/open-ai-10.M0y3TGoN.png>

## Chat completion workflow in WeWeb ​

The first thing to understand when working with the OpenAI's chat completion API is that the API expects to receive a list of message and each message will contain several pieces of information.  
In developer terms, the chat completion API expects an array of objects :  
  
What you can see in the screenshot above is that:  
the first item on the list is always the personality and instructions we defined as the system the rest of the list alternatives between messages from the user and answers from the AI assistant  
This is important because it tells us how we need to structure the input we send to chatGPT to get helpful results.  
In WeWeb, we'll create a workflow that does the following when a user asks a question:  
we update our list of messages with the user's question we send that list of messages to OpenAI when OpenAI answers, we update our list of messages with the AI's answer  
Three things to keep in mind:  
if we don't send the expected format (i.e. an array of objects), the API will return an error message, so we need to make sure we include the role and content of each message if we don't send the entire message sequence, the AI will not have the entire context of the conversation if we send the entire message sequence but don't follow the sequential order of the conversation, the AI will not have the right context  
Let's take a look at a simple example to make things clearer.

### Images

<https://docs.weweb.io/assets/open-ai-11.H6PKEUsA.png>

## Step 1 - Update a messages array variable with the user's input: ​

Note that we are sending the information shown in the OpenAI documentation for a user message, i.e. {"role": "user", "content": "Who won the world series in 2020?"}  
Though you may have noticed that we don't include the first system message in our list of messages. This is because the messages variable will need to go back and forth in the frontend (i.e. the user's browser) every time a user asks a question to the AI.  
Instead, what we did is we defined the system in a secured prompt at plugin level and we'll call that system in step 2. That way, its content will not be visible in the user's navigator like the list of user and AI messages:

### Images

<https://docs.weweb.io/assets/open-ai-12.DlehW7si.png>

<https://docs.weweb.io/assets/open-ai-15.ClRf9a49.png>

## Step 2 - Make a request to OpenAI's API: ​

This screenshot shows us that:  
our second action in the workflow uses the chat completion action of the OpenAI plugin with the gpt-4 model with the system we defined in WeWeb's OpenAI plugin secured prompts and sends it the messages variable we have in our project and updated in step 1

### Images

<https://docs.weweb.io/assets/open-ai-13.DCgW69w3.png>

## Step 3 – Update the messages variable with the AI's answer ​

That's it!  
You should be good to go 🙂  
To learn more about how to use OpenAI's Chat completion in WeWeb, we recommend reading OpenAI's API documentation and reverse-engineering WeWeb's TemplateGPT project:

### Images

<https://docs.weweb.io/assets/open-ai-14.Cnt25TXh.png>

<https://docs.weweb.io/assets/open-ai-8.CA3Qnv1o.png>