
# AI Assistant API Documentation

***Version:*** *v3*

***Last Updated****: February 10, 2025*

***Written By:*** *Eden Asipov*

## Table of Contents

# Introduction

The EmbedGPT.chat API allows you to create and manage custom AI assistants for your websites or applications. These assistants can be configured with custom instructions and additional data, such as website content or uploaded documents, to support specific business scenarios and reflect your brand.

This documentation covers two main areas:

- **Management API** – for programmatically creating and configuring AI assistants.
- **Chat API** – for sending messages to an assistant and retrieving responses, enabling integration into custom interfaces like CRMs, dashboards, or mobile apps.

## Purpose

Use this API when you need programmatic control over assistant setup and interaction. With it, you can:

- Create and configure AI assistants at scale.
- Automate website crawling and document ingestion.
- Integrate the assistant into your own UI using the API-based messaging API.
- Support advanced use cases that go beyond embedding the default JavaScript widget.

## Scenario

Here are two common ways developers and partners use the EmbedGPT.chat API:

### Support Multiple Businesses or Websites

If you provide services to multiple businesses – each with their own website – you can:

- Create a dedicated AI assistant for each business.
- Configure each assistant using that business's website content and documentation.
- Use the Chat API to send and receive messages directly from your own support dashboard or custom application.

**Example:**

You can create an assistant named `Client1Bot`, index content from

`client1store.com`, and programmatically fetch answers to questions like *"What's your return policy?"* – without embedding the JavaScript chat widget.

### White-label Integration

If you're a partner offering AI assistants as part of your own platform, you can:

- White-label the BizDriver.ai service under your own brand.
- Programmatically create and manage assistants for your customers via the API.
- Build your own UI for assistant creation and configuration.
- Use the Chat API to power assistant conversations behind the scenes.

For example, your platform might let customers create their own assistant by providing a website URL or uploading documents. Your system would then call the EmbedGPT.chat API in the background to create and configure each assistant – delivered under your own brand.

# Quick Start: Python Examples for Programmatic Management

These Python examples are tailored for customers and partners who want to programmatically create and manage an AI assistant using the EmbedGPT.chat management API or integrate an AI assistant's functionality into custom systems without relying on the EmbedGPT.chat-provided chat widget.

The following examples demonstrate creating an AI assistant, initiating a crawl, verifying readiness, and sending messages – all via API calls for backend or custom integration purposes.

## Obtain an API Key

All requests to the EmbedGPT.chat REST API require authentication using an API key.

To get started, follow these steps to obtain your API key from the EmbedGPT.chat portal:

1. **Create an Account**: Visit EmbedGPT.chat's website and click **Try it Free** to sign up with your email and password.
2. **Access My Account**: Log in and go to the **My Account** section in the dashboard.
3. **Get API Key**: In the **Chatbots** tab, select or create an AI assistant and open the **Add to Website** tab to find your API keys.

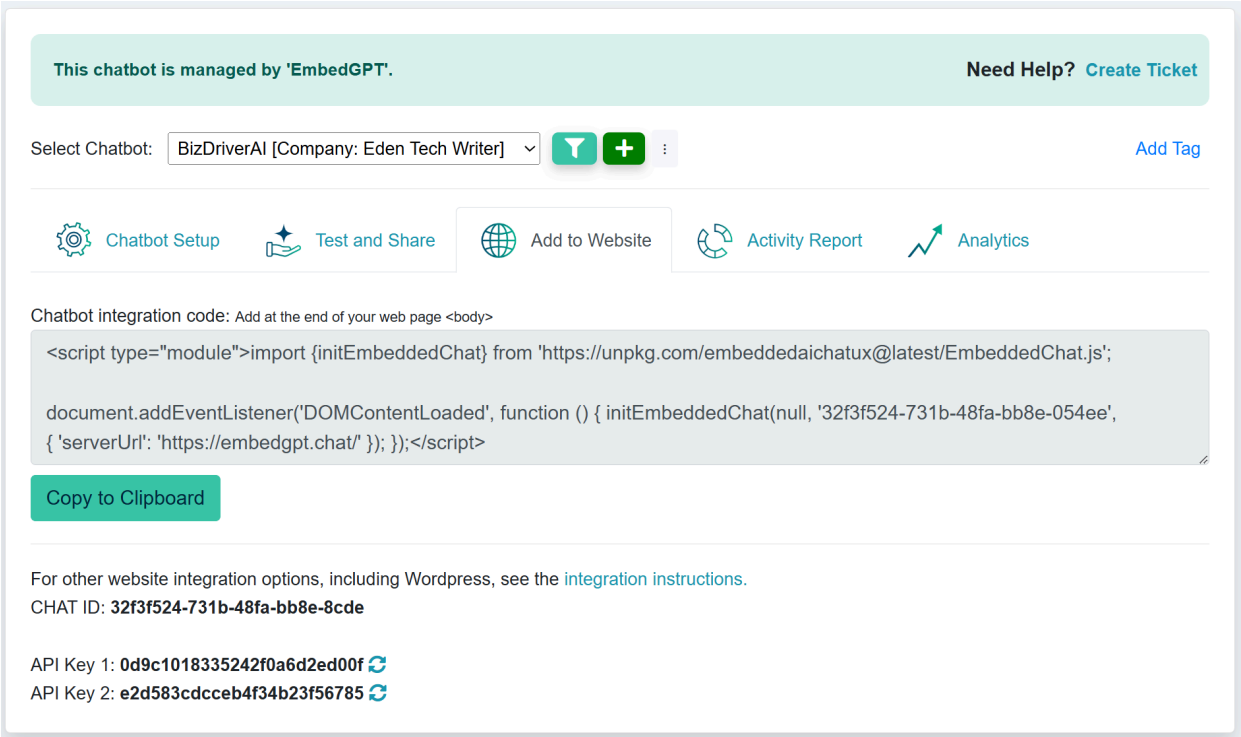
You'll see:

- **API Key 1**
- **API Key 2**

Use **API Key 1** (e.g., `<your-api-key>`) for all the examples in this documentation.

**Figure 1:** Add to Website screen

*Note the **API Key 1** and **API Key 2** fields at the bottom, which are required for authenticating API requests.*




# Prerequisites

- Python 3.8+
- `requests` library (`pip install requests`)
- Your EmbedGPT.chat **API Key 1**

# Programmatically Create an AI Assistant

The following example demonstrates how to create an AI assistant named **Client1Bot** by calling the API.

This assistant will be configured to crawl and index content from `https://client1store.com`, and will respond with a personalized welcome message.

```
import requests
import json

BASE_URL = "https://api.embedgpt.chat"
# NOTE: Replace with your actual API Key 1 from EmbedGPT.chat
API_KEY = "<your-api-key>"

def create_chatbot():
    url = f"{BASE_URL}/api/Chatbot/Create"
    headers = {
        "apiKey": API_KEY,
        "Content-Type": "application/json"
    }
    payload = {
        "crawlUrl": "https://client1store.com",
        "initialMessage": "Welcome to Client1Bot! How can I assist
you?",
        "chatbotName": "Client1Bot"
    }

    response = requests.post(url, headers=headers,
data=json.dumps(payload))

    if response.status_code == 200:
        result = response.json()
        chatbot_id = result.get("chatbotId")
```

```
        if chatbot_id:
            print("Client1Bot created successfully:", result)
            return chatbot_id
        else:
            print("Chatbot created, but chatbotId not found in
response.")
            return None
    else:
        print(f"Error creating Client1Bot: {response.status_code} -
{response.text}")
        return None

# Execute
chatbot_id = create_chatbot()
```

**Output**:

```
Client1Bot created successfully:
{
  "chatbotId": "daa69e50-eabb-426f-8ad6-e50c4adaf233",
  "crawledPageCount": 0,
  "isCompleted": False,
  "createdAt": "2025-03-26T14:00:00Z"
}
```

# Start Website Crawling Programmatically

Once you've created an AI assistant like **Client1Bot**, you need to initiate crawling so it can fetch and index relevant website content. This prepares the assistant to answer questions using information from the specified site.

The following Python example shows how to start the crawling process for a given `chatbotId` using the API:

```
def start_crawling(chatbot_id):
    url = f"{BASE_URL}/api/Chatbot/{chatbot_id}/Crawl/Start"
    headers = {"apiKey": API_KEY}

    response = requests.post(url, headers=headers)

    if response.status_code == 200:
        print("Crawling started for Client1Bot:", response.json())
    else:
        print(f"Error starting crawl: {response.status_code} -
{response.text}")

# Execute
start_crawling(chatbot_id)
```

**Output**:

```
Crawling started for Client1Bot: {'message': 'Crawling started
successfully.'}
```

## Verify the AI Assistant’s Readiness

Before sending user messages to your AI assistant, ensure that its knowledge base is ready by confirming the crawling process is complete. You can do this by polling the **Get Crawling Status** endpoint.

The following Python snippet demonstrates how to check the crawl status periodically until it is completed:

```
import time

def check_crawl_status(chatbot_id):
    url = f"{BASE_URL}/api/Chatbot/{chatbot_id}/Crawl/Status"
    headers = {"apiKey": API_KEY}

    while True:
        response = requests.post(url, headers=headers)
        if response.status_code == 200:
            status = response.json()
            print("Client1Bot crawl status:", status)
            if status["isCompleted"]:
                print("Client1Bot is ready for integration!")
                break
        else:
            print(f"Error checking status: {response.status_code} -
{response.text}")
            break
        time.sleep(10)  # Check every 10 seconds
```

```
# Execute
check_crawl_status(chatbot_id)
```

**Output**:

```
Client1Bot crawl status: {'crawledPageCount': 20, 'isCompleted':
False, 'createdAt': '2025-03-26T14:00:00Z'}
Client1Bot crawl status: {'crawledPageCount': 40, 'isCompleted': True,
'createdAt': '2025-03-26T14:00:00Z'}
```

**Client1Bot** is now ready for integration!

## Send Messages and Integrate Responses

After confirming your assistant is ready, you can begin sending user messages and retrieving AI-generated responses from your own system, such as a custom customer support dashboard or CRM integration.

This example demonstrates how to start a new conversation and send a message to **Client1Bot**, returning a response that can be displayed directly in your application.

**Note**: This operation may take 3 – 10 seconds, depending on the model used and the complexity of the response.

```
import uuid


def send_message(chatbot_id, message):
```

```
    conversation_id = str(uuid.uuid4())  # New conversation ID
    url =
f"{BASE_URL}/api/Chatbot/{chatbot_id}/Conversation/{conversation_id}/M
essage"
    headers = {
        "apiKey": API_KEY,
        "X-User-IP": "192.168.1.1",  # Optional
        "Content-Type": "application/json"
    }

    response = requests.post(url, headers=headers,
data=json.dumps(message))

    if response.status_code == 200:
        reply = response.json()["response"]
        print("Client1Bot response (for custom integration):", reply)
        return reply  # Return for use in your system
    else:
        print(f"Error sending message: {response.status_code} -
{response.text}")
        return None

# Execute
send_message(chatbot_id, "What’s your return policy?")
send_message(chatbot_id, "Do you ship internationally?")
```

**Output**:

```
Client1Bot response (for custom integration): Returns are accepted
within 30 days with a receipt.
Client1Bot response (for custom integration): Yes, we ship
internationally; rates vary by location.
```

# Authentication

All API requests require authentication using an API key.

## How to Authenticate

Include your API key in the `apiKey` HTTP header:

```
GET /api/Chatbot/{chatbotId} HTTP/1.1
Host: api.embedgpt.chat
apiKey: <your-api-key>
```

# API Endpoint Details

The following API reference provides documentation for integrating with the EmbedGPT.chat service. It enables you to create and manage AI assistants, initiate and maintain chat conversations, retrieve assistant responses, and interact with website crawling and indexed content data.

# Chat Operations

## Send a Chat Message

```
POST /api/Chatbot/{chatbotId}/Conversation/{conversationId}/Message
```

Initiates a message exchange within a specific conversation context. This operation sends a user message to the AI assistant identified by `chatbotId`, within an existing conversation session identified by `conversationId`. The assistant processes the input and returns an appropriate response based on its configuration and contextual understanding of the conversation history.

This API is used to programmatically interact with an AI assistant for dynamic conversational experiences, such as integrating into support systems, e-commerce assistants, or knowledge bots.

### Parameters

| Name | Location | Type | Required | Description |
|---|---|---|---|---|
| `chatbotId` | Path | String | Yes | AI assistant ID |
| `conversationId` | Path | String | Yes | Conversation ID |
| `apiKey` | Header | String | Yes | API key |
| `X-User-IP` | Header | String | No | User IP (optional) |

### Request Body

```
"What’s your return policy?"
```

### Response

```
{
  "response": "Returns are accepted within 30 days with a receipt."
}
```

# AI Assistant Management

## Create and Configure a New AI Assistant

```
POST /api/Chatbot/Create
```

Creates a new AI assistant and configures it with an initial website to crawl and a default welcome message. This operation initializes the assistant and prepares it for content indexing.

Use this endpoint to programmatically provision AI assistants for your own website, customers, or platform users.

### Request Body

```
{
  "crawlUrl": "https://example.com",
  "initialMessage": "Hello, how can I assist you?",
  "chatbotName": "SupportBot"
}
```

### Response

```
{
  "chatbotId": "daa69e50-eabb-426f-8ad6-e50c4adaf233",
  "crawledPageCount": 0,
  "isCompleted": false,
  "createdAt": "2025-03-26T14:00:00Z"
}
```

## Start Website Crawling

```
POST /api/Chatbot/{chatbotId}/Crawl/Start
```

Initiates the crawling process for a specific AI assistant. This operation fetches content from the specified website and prepares the assistant's knowledge base for answering user questions.

Use this endpoint after creating and configuring the assistant to begin indexing the website content.

### Response

```
{
  "message": "Crawling started successfully."
}
```

## Get Crawling Status

```
POST /api/Chatbot/{chatbotId}/Crawl/Status
```

Retrieves the current progress of the website crawling process for the specified AI assistant. This endpoint is typically used to poll for completion before enabling user interaction with the assistant.

Use this operation after initiating a crawl to monitor when the assistant is ready to respond with complete, indexed content.

### Response

```
{
  "crawledPageCount": 40,
  "isCompleted": true
}
```

## Delete the AI Assistant’s Configuration

```
DELETE /api/Chatbot/{chatbotId}
```

Permanently removes the specified AI assistant and its associated configuration from the system. This operation is irreversible and should be used with caution.

Use this endpoint to decommission an assistant that is no longer needed.

### Response

```
{
  "message": "Chatbot configuration deleted successfully."
}
```

## Get Page Index Data

```
GET /api/Chatbot/{chatbotId}/PageIndexData
```

Retrieves structured index information about pages that were crawled for a given AI assistant.

This data is useful for auditing, debugging, and enhancing transparency of what content is available to the assistant during chat operations.

### Query Parameters

| Name | Location | Type | Required | Description |
|---|---|---|---|---|
| `startIndex` | Query | Integer | No | Start index |
| `count` | Query | Integer | No | Number of entries |

| | | | | |
|---|---|---|---|---|
| `crawlType` | Path | String | Yes | Crawl type (e.g., "full") |
| `apiKey` | Header | String | Yes | API key |

Response

```
{
  "id": "page-001",
  "url": "https://example.com/page1"
}
```

## Error Handling

| HTTP Status Code | Description |
|---|---|
| 200/201 | Success |
| 400 | Bad request |
| 401 | Unauthorized / Invalid API Key |
| 403 | Forbidden |
| 404 | Not Found |
| 500 | Internal Server Error |

## Notes on Widget Integration

If you prefer the EmbedGPT.chat-provided widget, use the JavaScript snippet from the **Add to Website** tab.

## Glossary

| Term | Description |
| --- | --- |
| AI Assistant | A chatbot created using the API that responds to user messages. |
| `chatbotId` | Unique identifier for the assistant. |
| `conversationId` | A session identifier for grouping related messages. |
| Crawl | The process of fetching and indexing website content for assistant use. |
| Widget | Pre-built EmbedGPT.chat chat interface, added via JavaScript snippet. |

## Next Steps

To begin integrating with the EmbedGPT.chat API:

- Replace `API_KEY` and `chatbot_id` in your requests with the actual values provided in your EmbedGPT.chat dashboard.
- For advanced use cases or access to partner features, contact EmbedGPT.chat's support.