

Diseño del Prompt para el monitoreo de Logs

Estoy necesitando implementar un proyecto ASP.NET Core Web Api con .NET 9 para el monitoreo de unas Tables de Azure Storage para dos aplicaciones AppSalud y LinaChatbot

El objetivo del proyecto es leer unos Logs que están en unas tablas, para detectar:

- errores de servicios externos (outbound)
- errores internos recurrentes de la aplicación
- tiempos de respuesta elevados según el tipo de servicio

Criterio de aceptación 1: Identificación errores y tiempos de respuestas elevados

Dado que se requiere analizar los Logs en las diferentes tablas del Azure Storage,

Cuando se obtengan los registros de las tablas,

Entonces se debe implementar un LLM que analice los registros obtenidos para identificar errores de servicio externos, internos, y tiempos de respuestas elevados.

Criterio de aceptación 2: Identificación errores y tiempos de respuestas elevados

Dado que se requiere enviar una notificación

Cuando cuando el criterio de aceptación 1 se cumpla

Entonces se debe enviar una notificación ya sea por (Telegram, correo, etc.)

Recursos

Para AppSalud este son los datos que se tienen:

Cuneta de almacenamiento:

DefaultEndpointsProtocol=https;AccountName=appsaluddevstorage;AccountKey=twg633R2bnHIWajnuhgkEUaLk0tUU6bv0MSjkplvBQYpz2ZcYSMQWN/ngwQDjMLzTQynp5VgDyOAg==;EndpointSuffix=core.windows.net;

Listado de tablas a analizar: AppLog, Log, LogCsAuthenticate

Para LinaChatbot este son los datos que se tienen:

Cuneta de almacenamiento:

DefaultEndpointsProtocol=https;AccountName=Linadevstorage;AccountKey=twg633R2bnHIWajnuhgkEUaLk0tUU6bv0MSjkplvXRTBQYpz2ZcYSMQWN/ngwQDjMLzTQynp5VgDyOAg==;EndpointSuffix=core.windows.net;

Listado de tablas a analizar: AppLog, Log, LogCsAuthenticate

En cuanto al LLM esto es lo que se tiene:

gpt-5-mini

[Abrir en el área de juegos](#) [Solicitar cuota](#) [Editar](#) [Eliminar](#)

Punto de conexión

URL de destino
`https://orquestador-foundry.cognitiveservices.azure.com/openai/responses?ap...`

Clave
.....

Información de implementación

Nombre gpt-5-mini	Estado de aprovisionamiento Correcto
Tipo de implementación Estándar global	Creado el 2025-09-02T19:13:14.9896421Z
Creado por estiverson.ortega@softwareone.com	Modificado el Sep 2, 2025 2:13 PM
Modificado por estiverson.ortega@softwareone.com	Directiva de actualización de versiones Una vez que haya disponible una nueva versión predeterminada
Límite de velocidad (tokens por minuto) 20,000	Límite de velocidad (peticiones por minuto) 20
Nombre del modelo gpt-5-mini	Versión del modelo 2025-08-07
Estado del ciclo de vida Generalmente disponible	Fecha de creación Aug 6, 2025 7:00 PM
Fecha de actualización Aug 6, 2025 7:00 PM	Fecha de retiro del modelo Aug 7, 2026 7:00 PM

Idioma C# **SDK** Azure OpenAI SDK **Tipo de autenticación** Key Authentication [Abrir en VS Code](#)

Introducción

Vinculo al paquete NuGet y algunos ejemplos. Para obtener información adicional sobre el SDK de Azure OpenAI, consulte la [documentación](#) completa y [ejemplos](#).

1. Autenticación mediante la clave de API

Para los puntos de conexión de API de OpenAI, implemente el Modelo para generar la URL del punto de conexión y una clave de API para autenticarse con el servicio. En esta muestra, el punto de conexión y la clave son cadenas que contienen la dirección URL del punto de conexión y la clave de API.

La dirección URL del punto de conexión de API y la clave de API se pueden encontrar en la página Implementaciones y punto de conexión una vez implementado el modelo.

Para crear un cliente con el SDK de OpenAI usando una clave de API, inicie el cliente pasando su clave de API a la configuración del SDK. Esto le permite autenticarse e interactuar con los servicios de OpenAI sin problemas.

```
var endpoint = new Uri("https://orquestador-foundry.cognitiveservices.azure.com/");
var model = "gpt-5-mini";
var deploymentName = "gpt-5-mini";
var apiKey = "your-api-key";

AzureOpenAIClient azureClient = new(
    endpoint,
    new AzureKeyCredential(apiKey));
ChatClient chatClient = azureClient.GetChatClient(deploymentName);
```

2. Instalar dependencias

1. Instalación de dotnet runtime

2. Ejecutar un ejemplo de código básico

En este ejemplo se muestra una llamada básica a la API de finalización de chat. La llamada es sincrónica.

```
using Azure;
using Azure.AI.OpenAI;
using Azure.AI.OpenAI.Chat;
using OpenAI.Chat;

var endpoint = new Uri("https://orquestador-foundry.cognitiveservices.azure.com/");
var deploymentName = "gpt-5-mini";
var apiKey = "your-api-key";

AzureOpenAIClient azureClient = new(
    endpoint,
    new AzureKeyCredential(apiKey));
ChatClient chatClient = azureClient.GetChatClient(deploymentName);

// Support for this recently-launched model with MaxOutputTokens parameter requires
// Azure AI OpenAI 2.2.0-beta.4 and SetNewMaxCompletionTokensPropertyEnabled
var requestOptions = new ChatCompletionOptions()
{
    MaxOutputTokenCount = 10000,
};

// The SetNewMaxCompletionTokensPropertyEnabled() method is an [Experimental] opt-in to use
// the new max_completion_tokens_750k property instead of the legacy max_tokens property.
// This extension method will be removed and unnecessary in a future service API version;
// please disable the [Experimental] warning to acknowledge.
#pragma warning disable ADAL001
requestOptions.SetNewMaxCompletionTokensPropertyEnabled(true);
```

