# Challenge 6: The Colonel Needs a Promotion

In this challenge, you and your team will add a new capability by creating a couple of new plugins for Semantic Kernel.

You have lately become a big fan of Shakespeare's sonnets. You love how they convey details and information, and you want your copilot to provide details about your company's policies in the poetic form of Shakespearean language.

Your challenge is correctly handling requests about your company's policies (return or shipping), including when users ask to get them as poems written in Shakespearean style.

**Objective:**

1. Create a new Semantic Kernel plugin that will retrieve only memories related to company policies.
2. Create a new Semantic Kernel plugin that will present any information as a poem written in Shakespearean language.
3. Register the plugins with Semantic Kernel.
4. Create a plan to respond to requests.
5. Replace the logic in the SemanticKernalRAGService.cs GetResponse method with one that will first make a plan to decide if your functions should be used or not, and then execute the completion request accordingly.

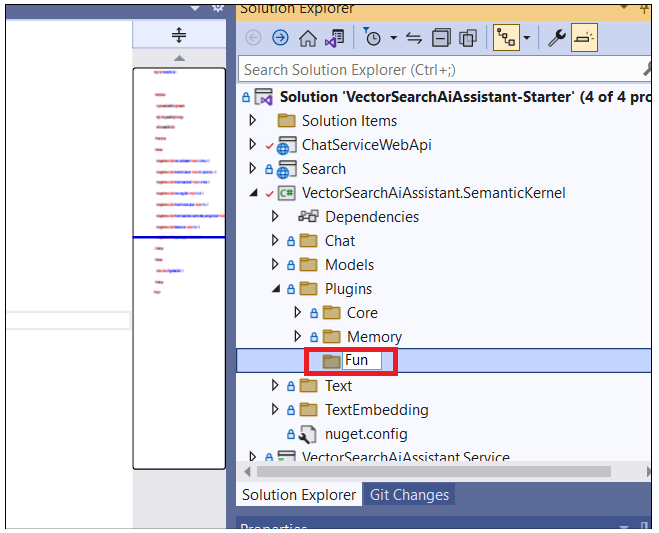
**Task 1 : create new plugins for Semantic Kernel**

1. Switch back to Visual Studio, and go to **VectorSearchAiAssistant.SemanticKernel -> Right-click on Plugins -> select Add- > New Folder**.

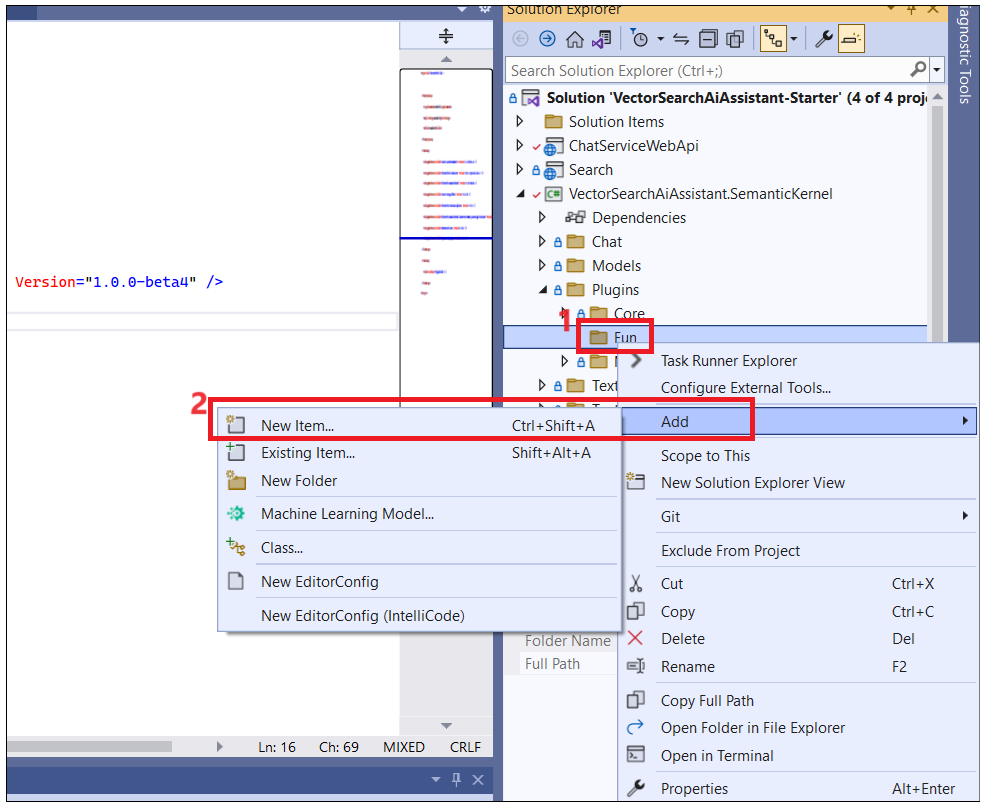
A screenshot of a computer

Description automatically generated

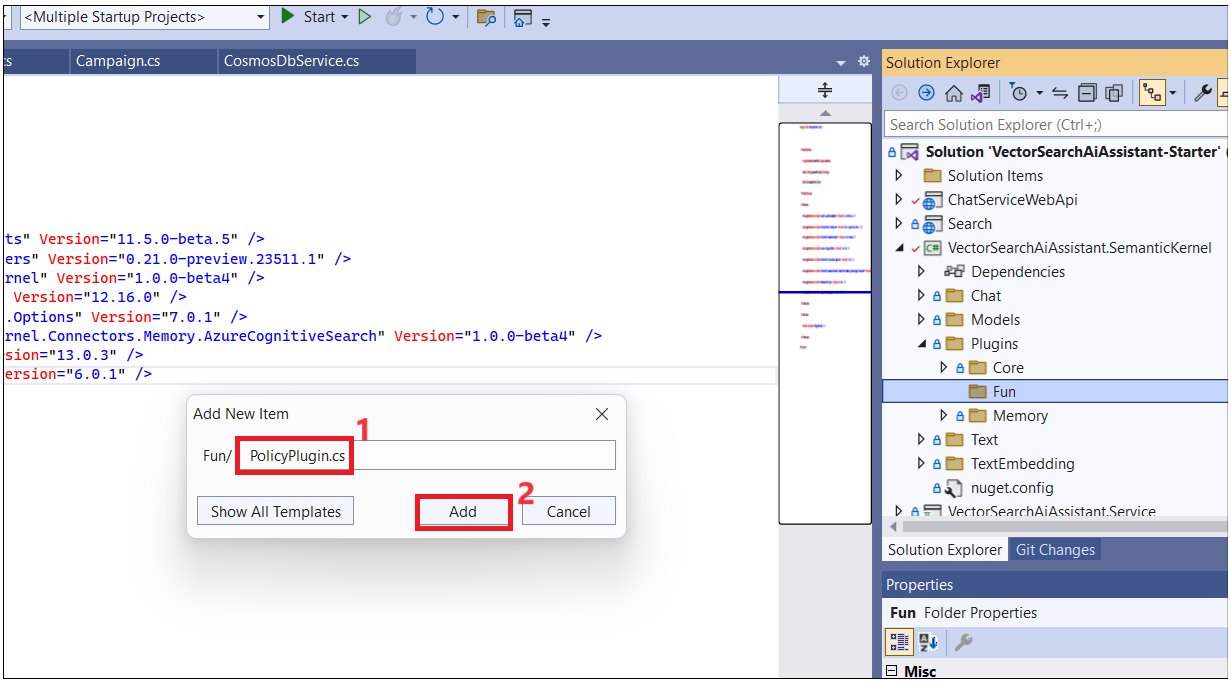
1. Enter the name as ++**Fun++** .



1. Right-click on the new folder **Fun -> Add- > New Item**.



1. Enter the file name as ++**PolicyPlugin.cs++** and then click on **Add**. Semantic Kernel for policy memories to enable the planner to pick it up.



1. Replace the code with the code below and save the file. You can also find this file in **C:\Labfiles folder.**

using Microsoft.Extensions.Logging;

using Microsoft.SemanticKernel.Memory;

using Microsoft.SemanticKernel;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using VectorSearchAiAssistant.SemanticKernel.Plugins.Memory;

using System.Text.Json;

namespace VectorSearchAiAssistant.SemanticKernel.Plugins.Fun

{

public class PolicyPlugin

{

private const double DefaultRelevance = 0.7;

private const int DefaultLimit = 1;

private readonly VectorMemoryStore \_policyMemory;

private readonly ILogger \_logger;

/// <summary>

/// Creates a new instance of the TextEmbeddingMemorySkill

/// </summary>

public PolicyPlugin(

VectorMemoryStore policyMemory,

ILogger logger)

{

\_policyMemory = policyMemory;

\_logger = logger;

}

[SKFunction, Description("Get information product policies.")]

public async Task<string> RecallAsync(

[Description("The input text to find related memories for")] string text,

[Description("The relevance score, from 0.0 to 1.0, where 1.0 means perfect match"), DefaultValue(DefaultRelevance)] double? relevance,

[Description("The maximum number of relevant memories to recall"), DefaultValue(DefaultLimit)] int? limit)

{

relevance ??= DefaultRelevance;

limit ??= DefaultLimit;

var policyMemories = await \_policyMemory

.GetNearestMatches(text, limit.Value, relevance.Value)

.ToListAsync()

.ConfigureAwait(false);

if (policyMemories.Count == 0)

{

\_logger.LogWarning("The policies memory store does not contain any matching memories.");

return string.Empty;

}

\_logger.LogTrace("Done looking for policy memories");

var result = string.Join("\n", policyMemories.Select(m => m.Metadata.Text));

return result;

}

}

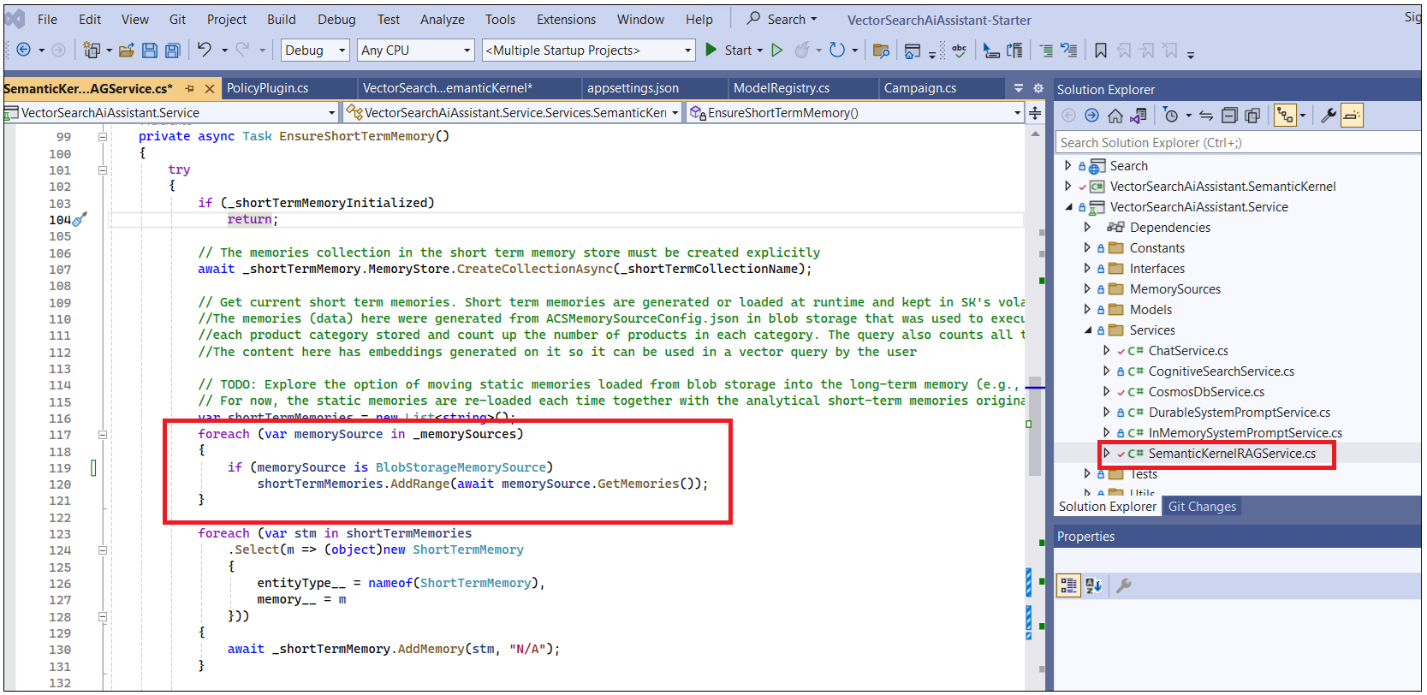
}

A screenshot of a computer

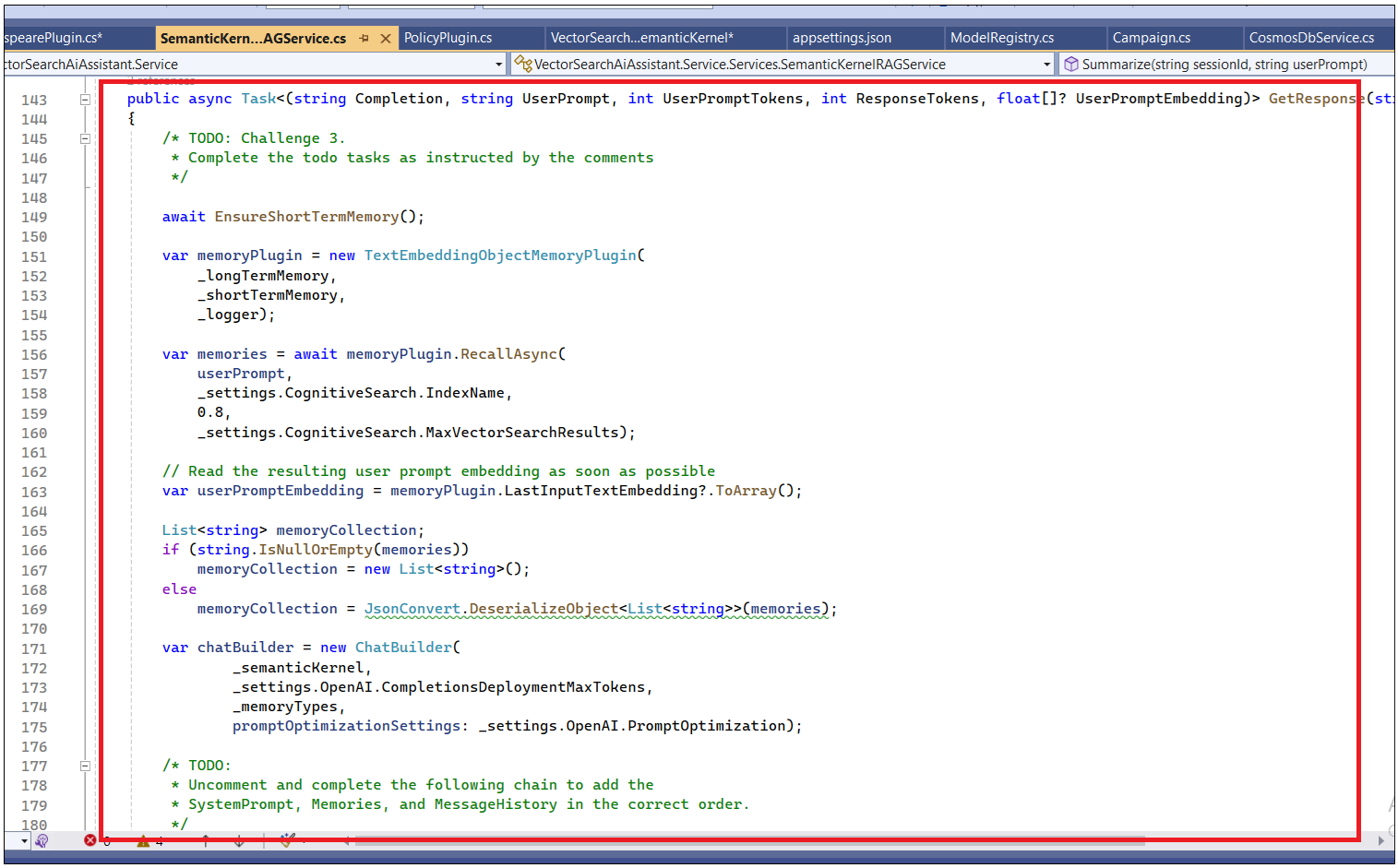
Description automatically generated

1. Construct the dedicated policy memory to find out the short-term memories that are loaded, and open SemanticKernelRAGService.cs from **VectorSearchAiAssistant.Service -> Service.** Insert the below code in Line number 119.

**if (memorySource is BlobStorageMemorySource)**

****

1. Replace the body of the GetResponse method in the SemanticKernelRAGService class with the following code:

****

**await EnsureShortTermMemory();**

**var policyPlugin = new PolicyPlugin(**

**\_shortTermMemory,**

**\_logger);**

**\_semanticKernel.ImportFunctions(policyPlugin);**

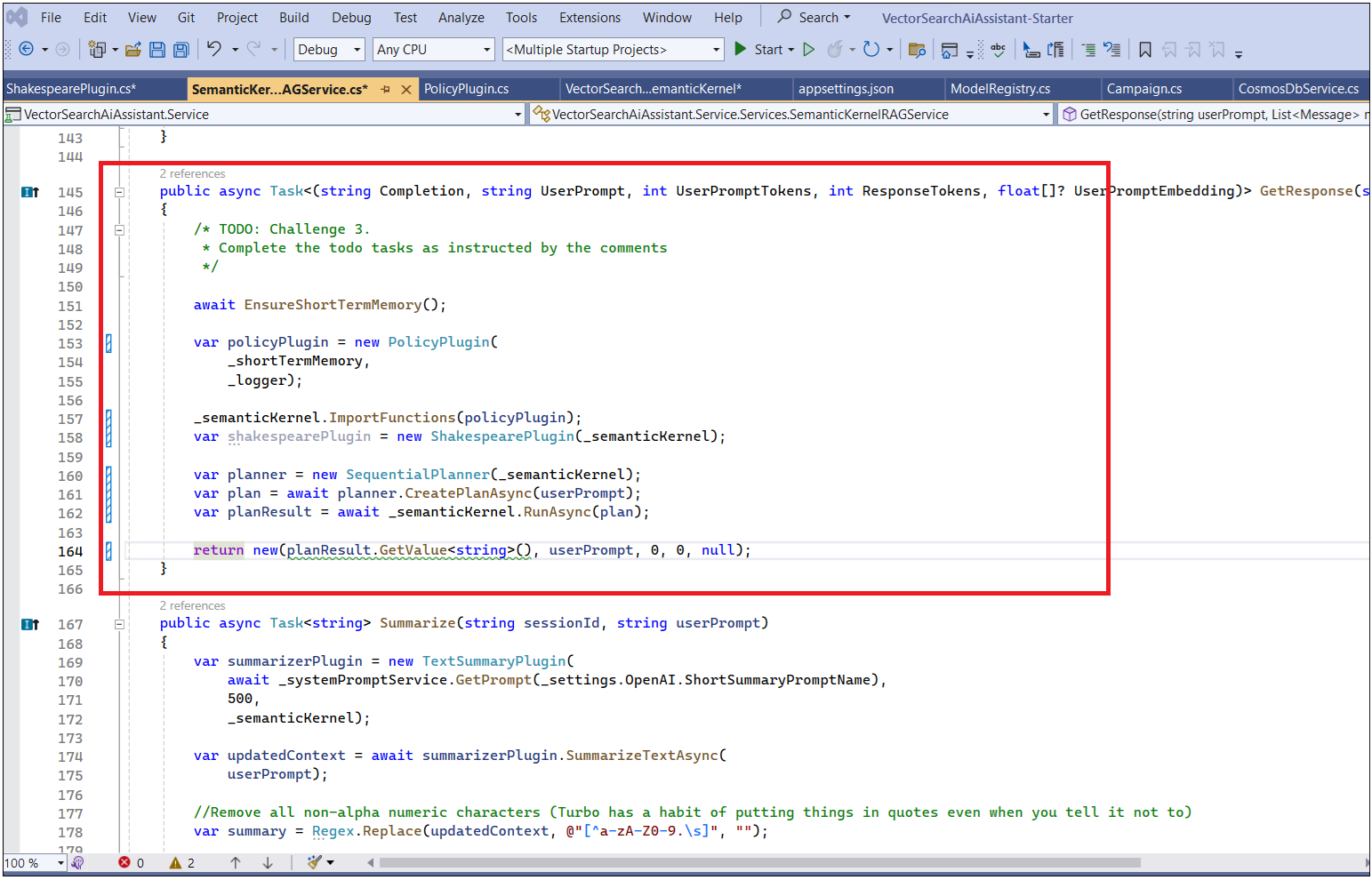
**var shakespearePlugin = new ShakespearePlugin(\_semanticKernel);**

**var planner = new SequentialPlanner(\_semanticKernel);**

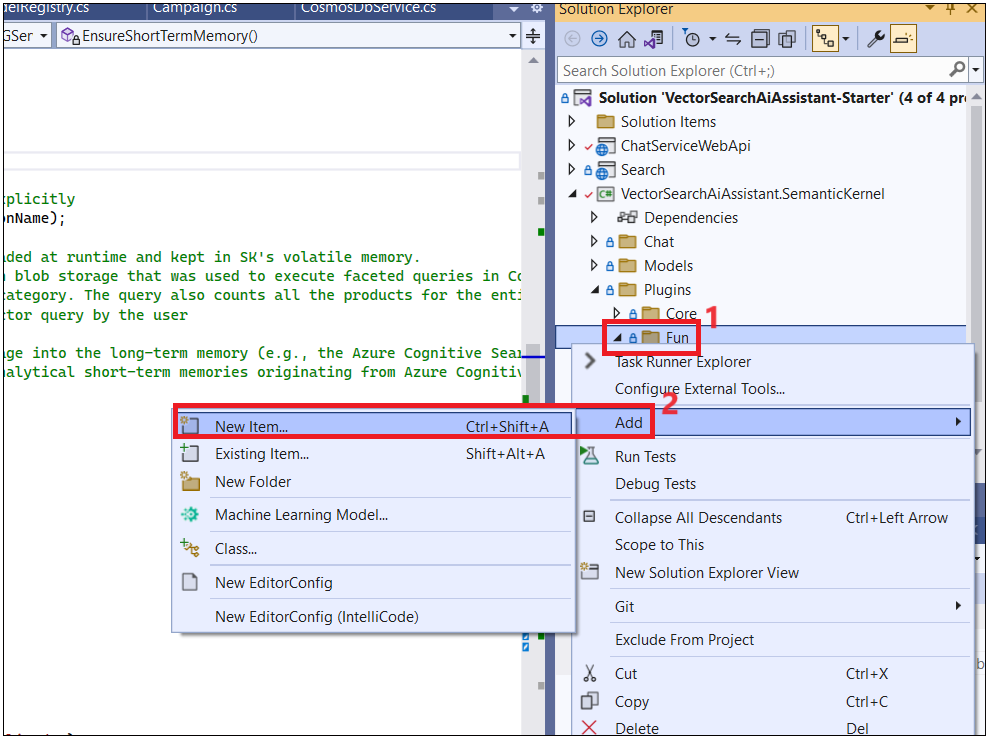
**var plan = await planner.CreatePlanAsync(userPrompt);**

**var planResult = await \_semanticKernel.RunAsync(plan);**

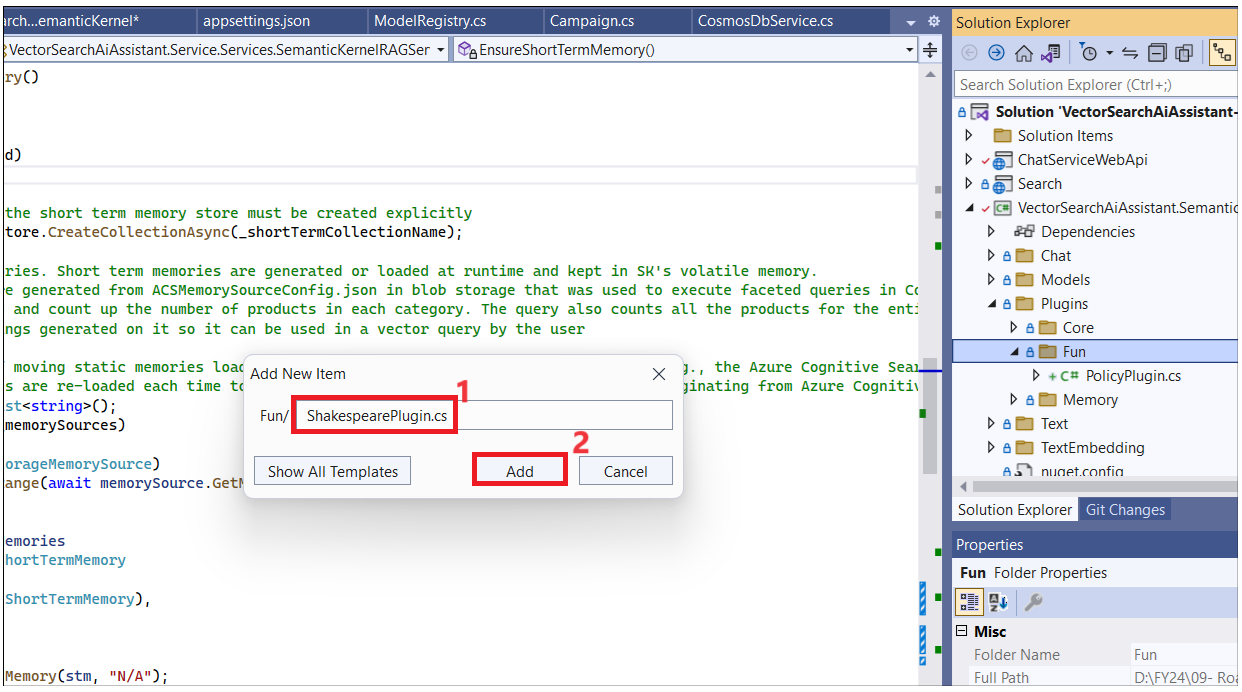
**return new(planResult.GetValue<string>(), userPrompt, 0, 0, null);**

****

1. Create a Semantic Kernel plugin class for transforming text into Shakespearean poems. under **VectorSearchAiAssistant->Plugins>Fun. Right click ->. Add- > New Item.**



3. Name the class file as ++**ShakespearePlugin.cs++** and click on **Add**.

****

1. Replace the code with the below code and save all files.

**using Microsoft.SemanticKernel.Connectors.AI.OpenAI;**

**using Microsoft.SemanticKernel;**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace VectorSearchAiAssistant.SemanticKernel.Plugins.Fun**

**{**

**public class ShakespearePlugin**

**{**

**private readonly ISKFunction \_shakespeare;**

**private readonly IKernel \_kernel;**

**public ShakespearePlugin(**

**IKernel kernel)**

**{**

**\_kernel = kernel;**

**\_shakespeare = kernel.CreateSemanticFunction(**

**"""**

**{{$input}}**

**Rewrite the above as a poem in the style of Shakespeare.**

**""",**

**pluginName: nameof(ShakespearePlugin),**

**description: "Produces text in the style of Shakespeare.",**

**requestSettings: new OpenAIRequestSettings**

**{**

**MaxTokens = 2000,**

**Temperature = 0.1,**

**TopP = 0.5**

**});**

**}**

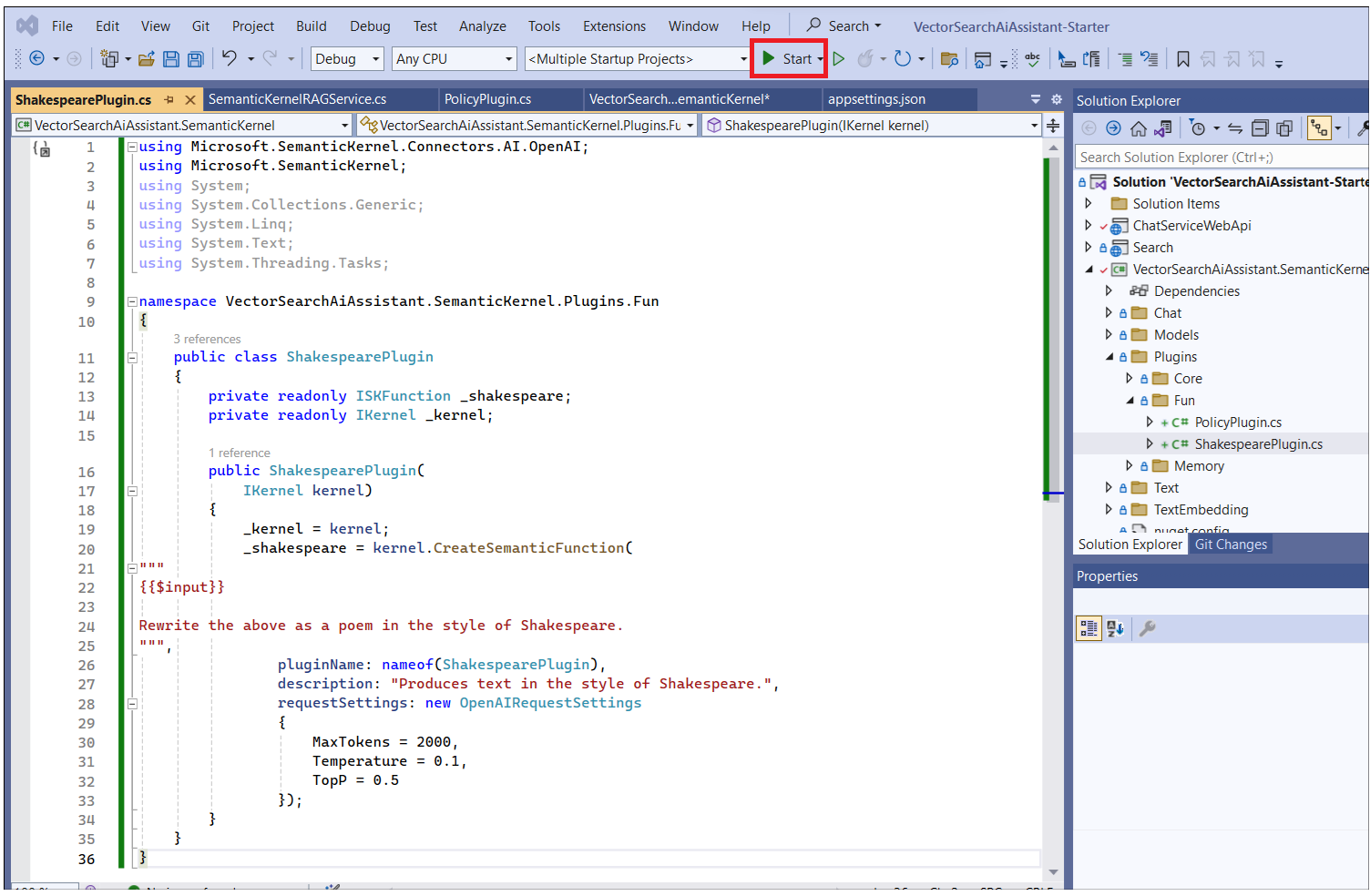
**}**

**}**

A screenshot of a computer

Description automatically generated

1. **Start** the solution now.



1. Go back to the chat interface and test it with the following user prompts:

**++Tell me about your return policy. Present it as a poem written by Shakespeare.++**

**++Tell me about your shipping policy. Present it as a poem written by Shakespeare.++**

A screenshot of a chat

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Run the solution in debug mode, and set a breakpoint in the GetResponse method of the SemanticKernelRAGService class. Inspect the plan variable to see the plan that was created by the planner.

**NOTE:** Notice how the GetResponse method is now much simpler because the planner is doing all the heavy lifting. At the same time, the actual token consumption values are not retrieved. For teams that are advancing quickly through the challenge, you might consider extending the challenge with a requirement to retrieve the token consumption values from the plan result.

1. **Stop** the solution

**Summary**

• executed to produce the completion using new plugins