Lab 04: This Challenge is Questionable

Woodgrove Bank is excited about the chance to use conversational AI to allow customers to interact with their bank accounts using natural language. They want to build a chatbot that will allow customers to ask questions about their transactions, which is where you come in.

In this task you add functionality to the transactions screen that allows you to ask questions about a member's transactions using natural language and viewing the results.

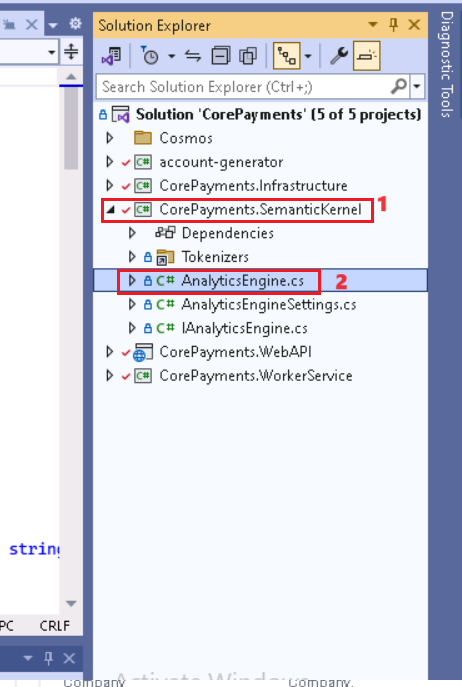
You will use the Microsoft Semantic Kernel with Azure OpenAI to create a SemanticFunction that will respond with the results of the users question.

**Objective**

* Review the semantic kernel code that accepts a user's question, sends transaction data for context, instructs the Copilot on how to answer and format the question, and returns the answer to the user.
* In the web app, use the **Analyze Transactions** button in the account transactions view to ask questions about the transactions.

## Task 1: Understanding the semantic kernel code

1. From visual Studio Solution Explorer, select **CorePayments.SemanticKernel -> Repository -> AnalyticsEngine.cs**.

****

1. Go through the **ReviewTransactions** method to understand the copilot configuration. This method has got the sample prompt. It instantiates the KernelBuilder and creates a new semantic function and creates a new context from the kernel and set the transaction context to your transaction data and the query context to your user query.

public async Task<string> ReviewTransactions(IEnumerable<Transaction> transactions, string query)

{

var builder = new KernelBuilder();

builder.WithAzureChatCompletionService(

\_settings.OpenAICompletionsDeployment,

\_settings.OpenAIEndpoint,

\_settings.OpenAIKey);

// TODO: Create a new kernel instance from the builder.

var kernel = builder.Build();

string skPrompt = @"

You are an analyst bot that helps staff summarize data about account transactions by processing a list of transactions.

You are provided the list of transactions in the JSON format, as well as the query submitted by the user.

You can return your results in JSON or the format specified by the user in the query.

For example:

+++

[INPUT]

Transaction Data:

[

{

""id"": ""9b0e2f75-6316-4d88-aa74-46ae5d4aef7b"",

""accountId"": ""0909090907"",

""description"": ""Item refund"",

""merchant"": ""Tailspin Toys"",

""type"": ""deposit"",

""amount"": 38.26,

""timestamp"": ""2023-06-20T23:13:00.9725896Z""

},

{

""id"": ""0bb8f13f-65b1-4611-83f5-c2c028ee6545"",

""accountId"": ""0909090907"",

""description"": ""Online purchase"",

""merchant"": ""Tailspin Toys"",

""type"": ""debit"",

""amount"": 38.26,

""timestamp"": ""2023-06-20T22:39:45.3257116Z""

}

]

User Query:

How many transactions does the accountId ""0909090907"" have?

[END INPUT]

Provide your response by completing the following bullet:

- Result: 2

+++

[INPUT]

Transaction Data:

{{$transactionData}}

User Query:

{{$query}}

[END INPUT]

Provide your response by completing the following bullet on a new line:

- Result:

";

var reviewer = kernel.CreateSemanticFunction(skPrompt, "review", "ReviewSkill", description: "Review the input", maxTokens: 2000, temperature: 0.0);

JsonSerializerOptions ser\_options = new()

{

WriteIndented = true,

MaxDepth = 20,

AllowTrailingCommas = true,

PropertyNameCaseInsensitive = true,

ReadCommentHandling = JsonCommentHandling.Skip,

};

// Optimize the transaction data we send as context to the semantic function.

var contextData = transactions.Select(t => new

{

t.description,

t.merchant,

t.type,

t.amount,

t.timestamp

});

var transactionData = JsonSerializer.Serialize(contextData, ser\_options);

var context = kernel.CreateNewContext();

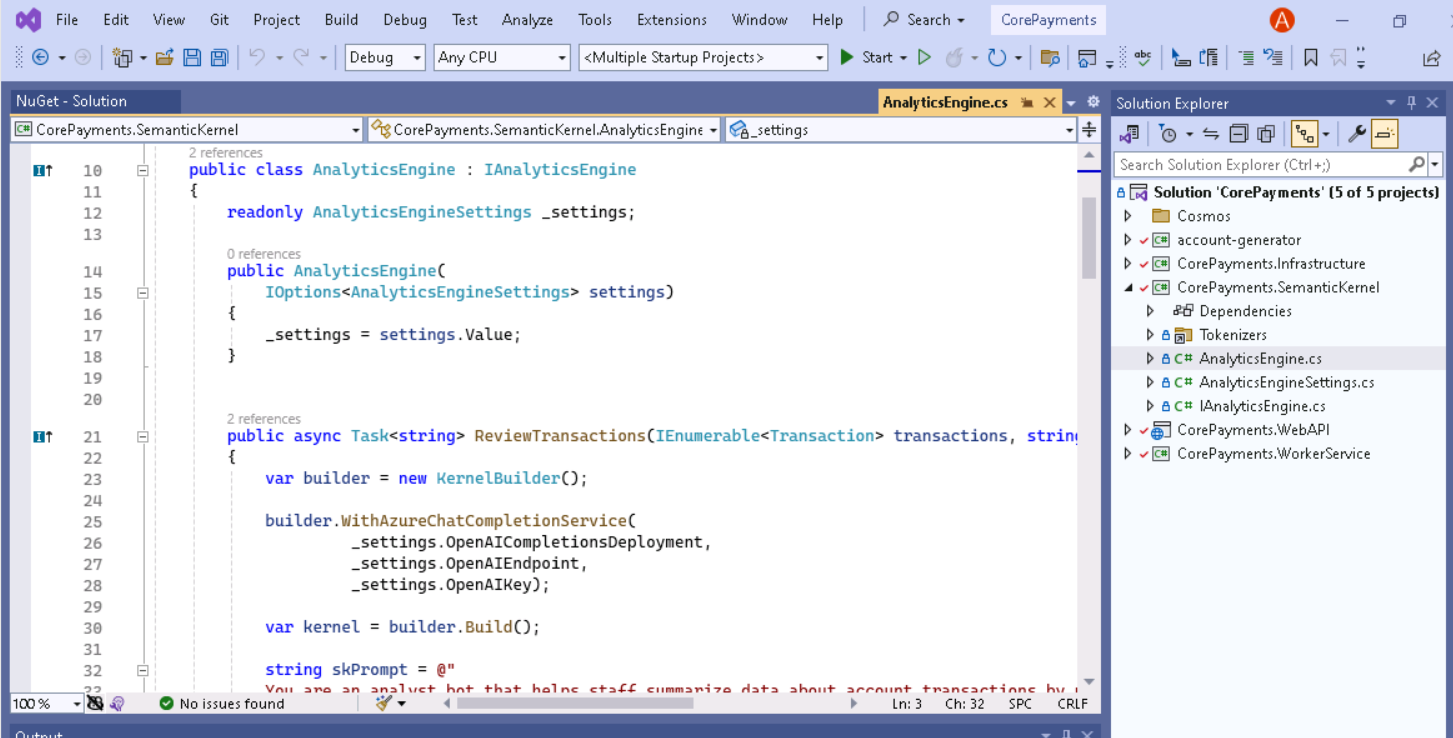
context["transactionData"] = transactionData;

context["query"] = query;

var result = (await reviewer.InvokeAsync(context)).Result;

return result;

}



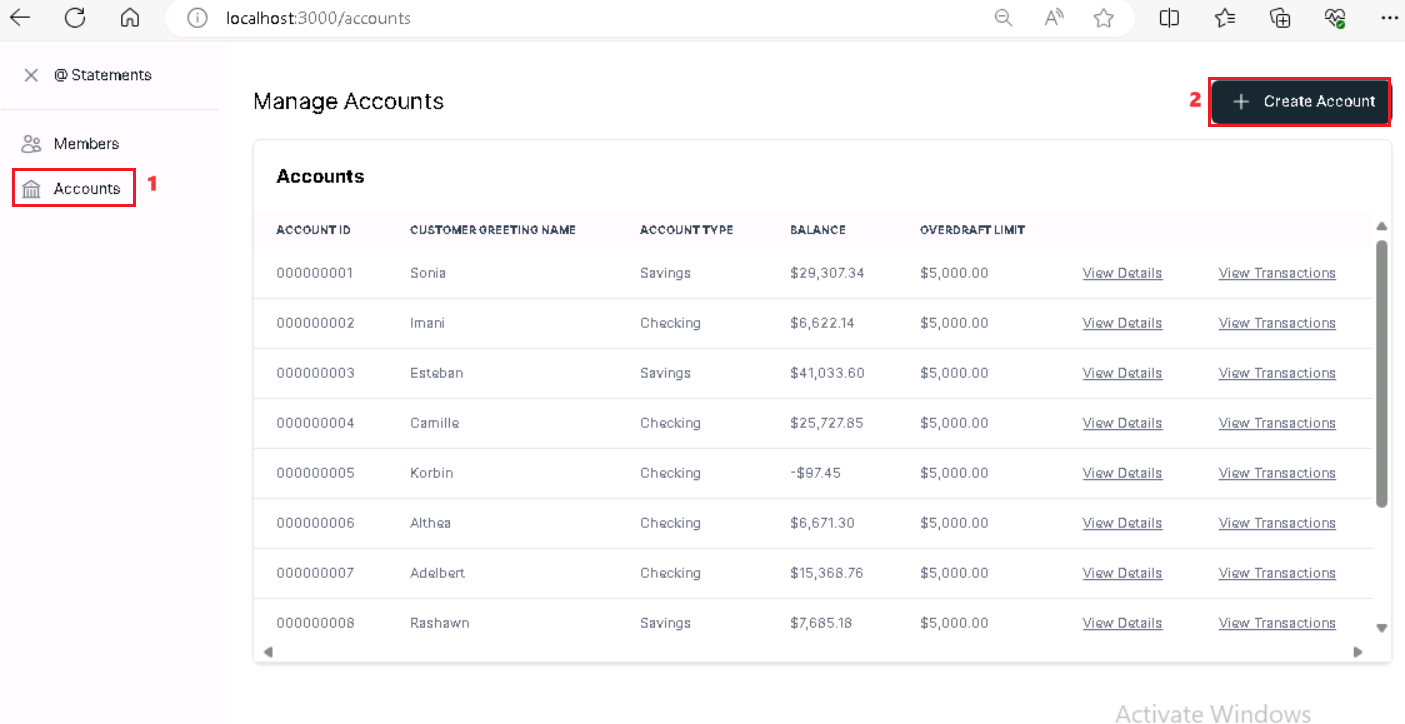
## Task 2:Test the application

1. Open [http://localhost:3000](http://localhost:3000/)

A screenshot of a computer

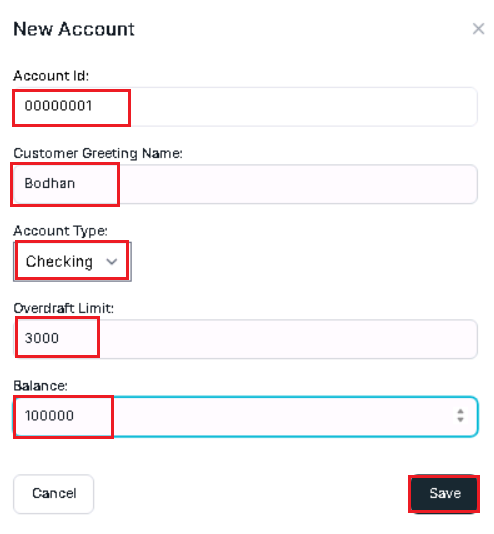
Description automatically generated

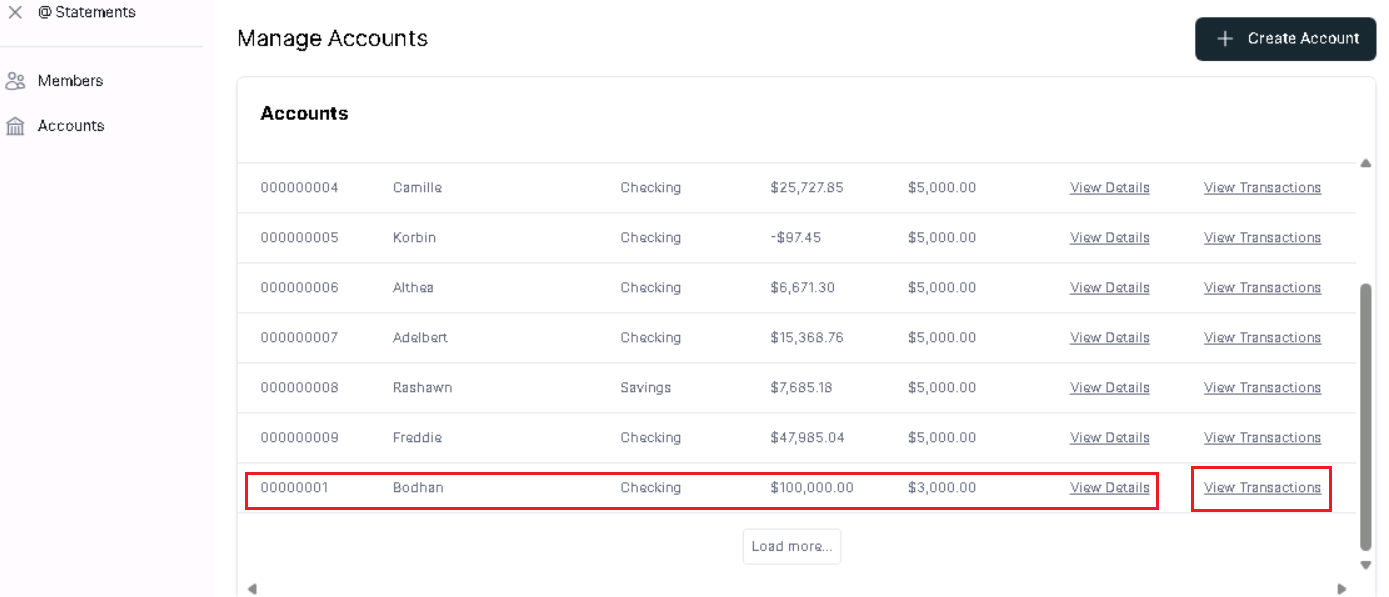
1. In the localhost page, navigate and click on **Accounts** and click on **+Create Account**.

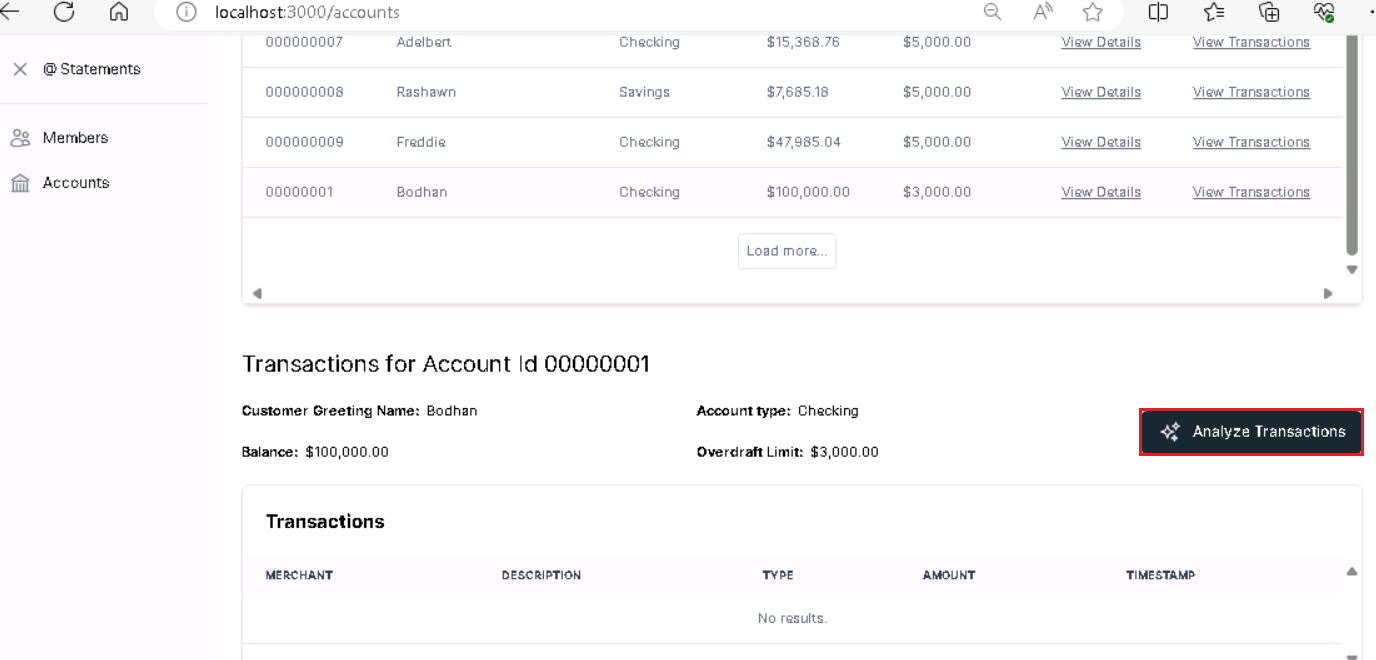


1. In the New Account dialog, enter the below details and click on **Save** button.

* Account id – +++**0000000001**+++
* Customer Greeting Name – +++**Bodhan**+++
* Account Type – **Checking**
* Overdraft limit – +++**5000**+++
* Balance – +++**150000**+++

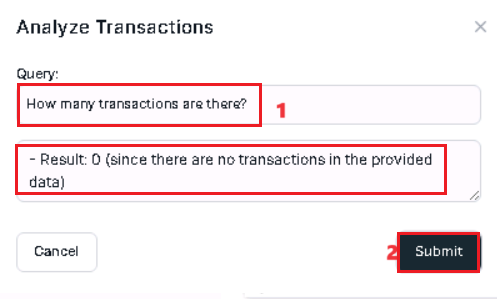


1. Select the **view transaction** of the new created account.
2. Select **Analyze Transactions**.

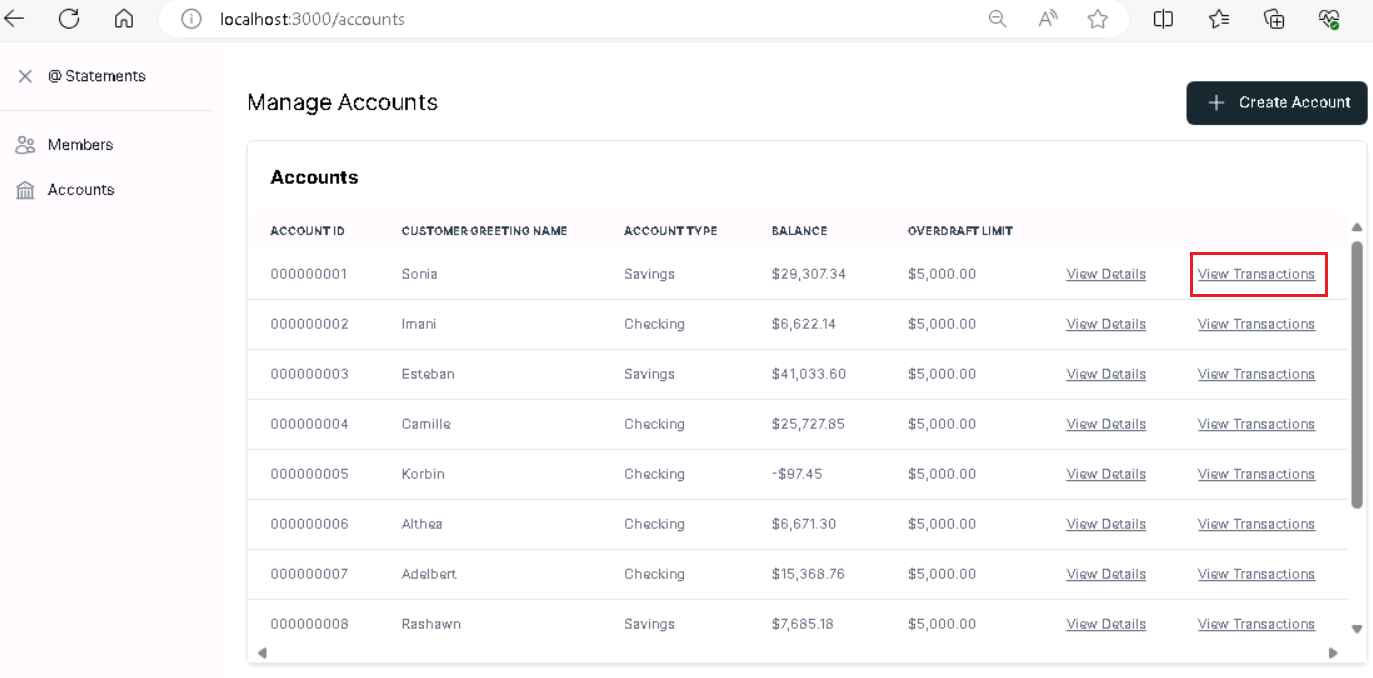


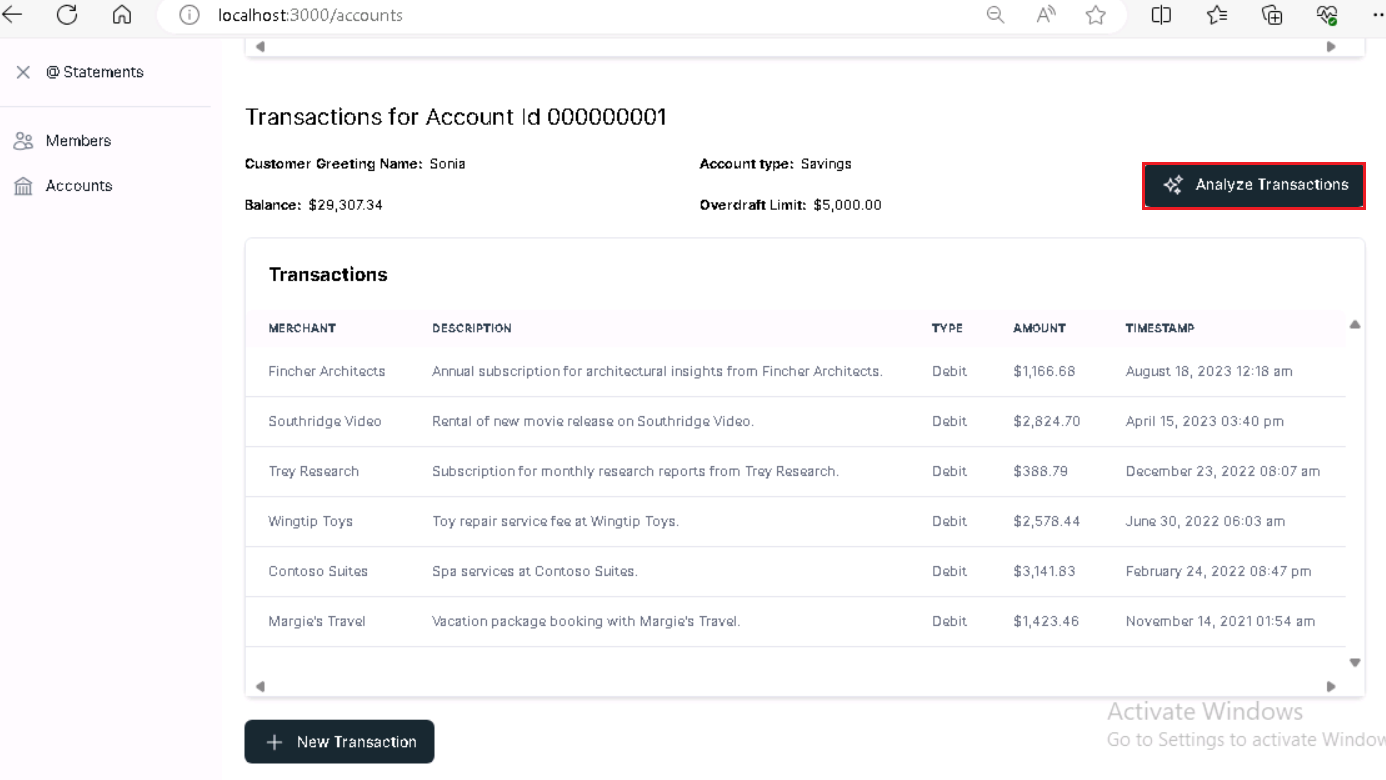
1. In Analyze Transactions tab , in the **Query** box, replace the current text with the following statement and click on **Submit** button. Review the response.

+++ **How many transactions are there?**+++



1. In **Manage Accounts** pane , select the **View Transactions** of existing account and click on the **Analyze Transactions.**





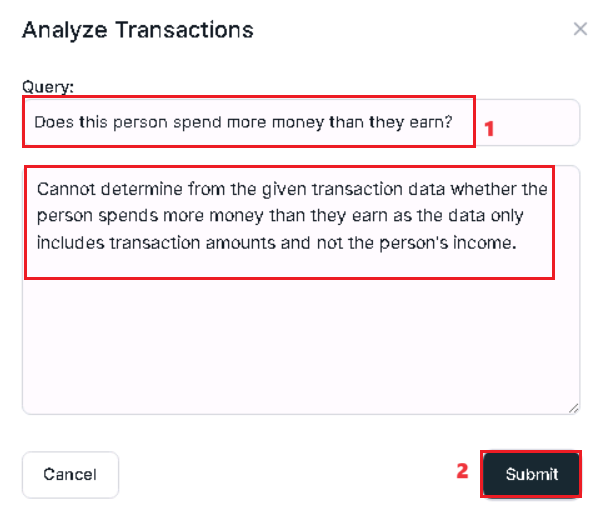
1. In **Analyze Transactions** tab ,in the **Query** box, replace the current text with the following statement and click on **Submit** button. Review the response.

+++ **How many transactions are there?**+++



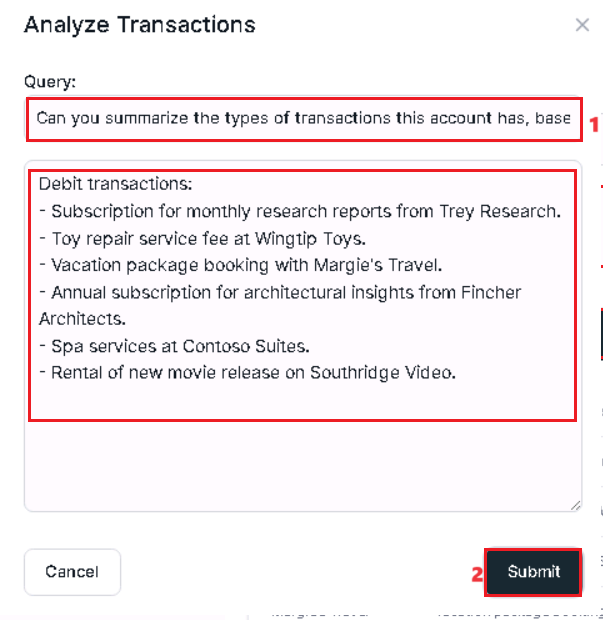
1. In the **Query** box, replace the current text with the following statement and click on **Submit** button. Review the response.

+++ **Does this person spend more money than they earn?**+++



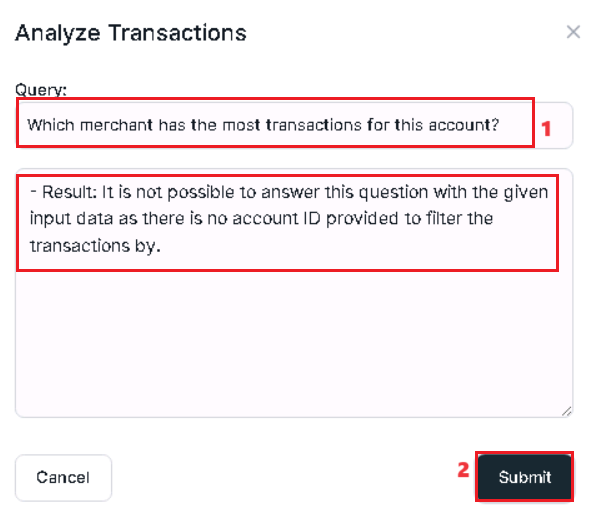
1. In the **Query** box, replace the current text with the following statement and click on **Submit** button. Review the response.

+++**Can you summarize the types of transactions this account has, based on the description?**+++



1. In the **Query** box, replace the current text with the following statement and click on **Submit** button. Review the response.

+++Which merchant has the most transactions for this account?+++



1. Open a browser and go to +++[**https://portal.azure.com**](https://portal.azure.com)+++ and sign in with your Azure login credentials, if not logged in already.
2. On the Home page, click on **Resource groups** tile.

A screenshot of a computer

Description automatically generated

1. Click on your resource group name and then click on **Azure Cosmos DB account** name from the list of resources.

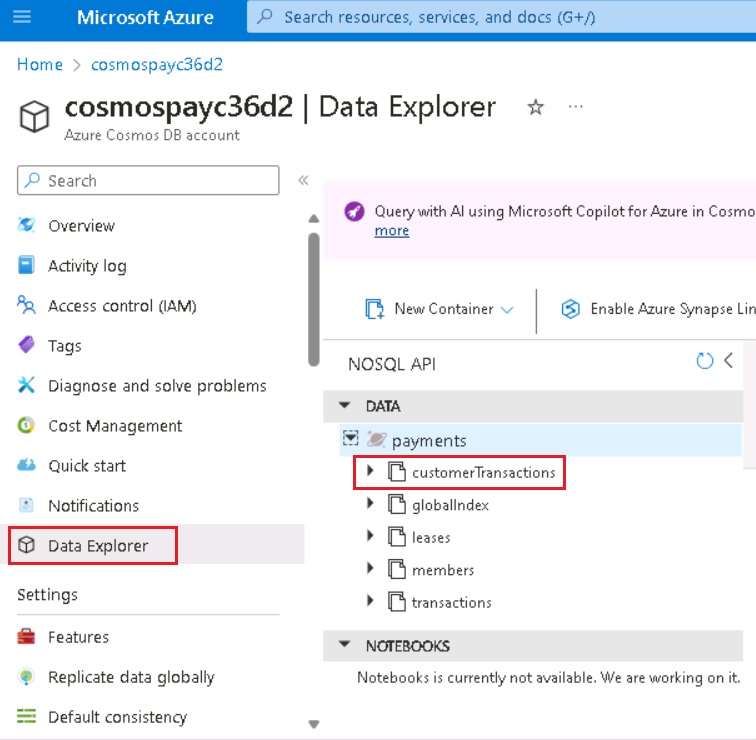
A screenshot of a computer

Description automatically generated

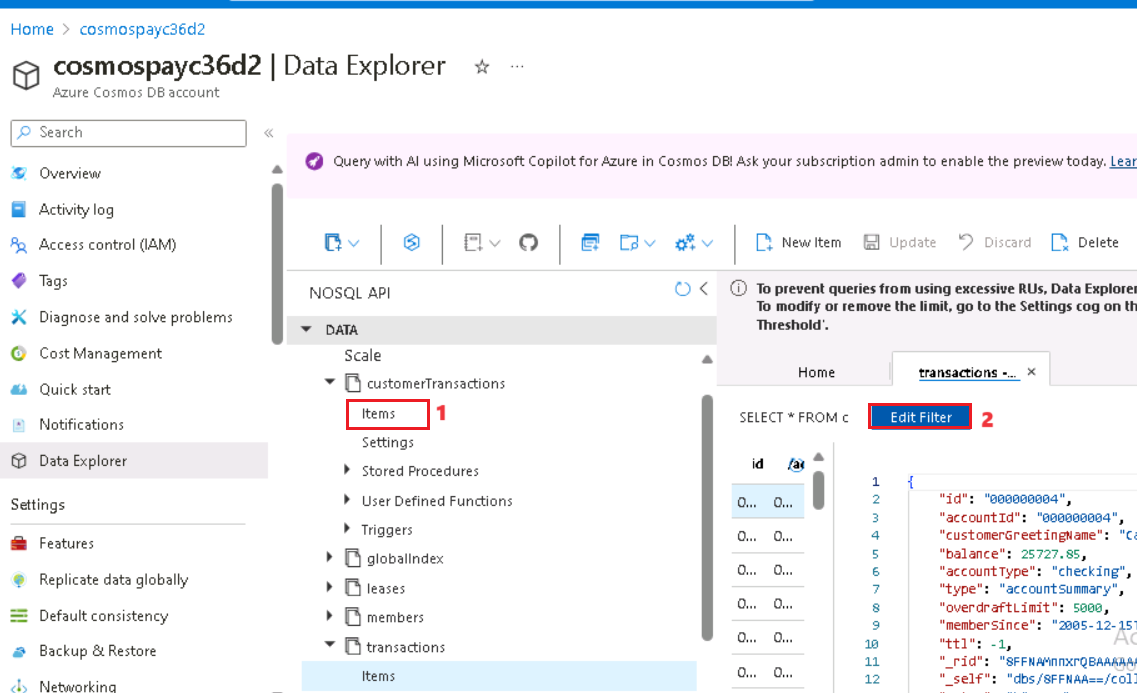
A screenshot of a computer

Description automatically generated

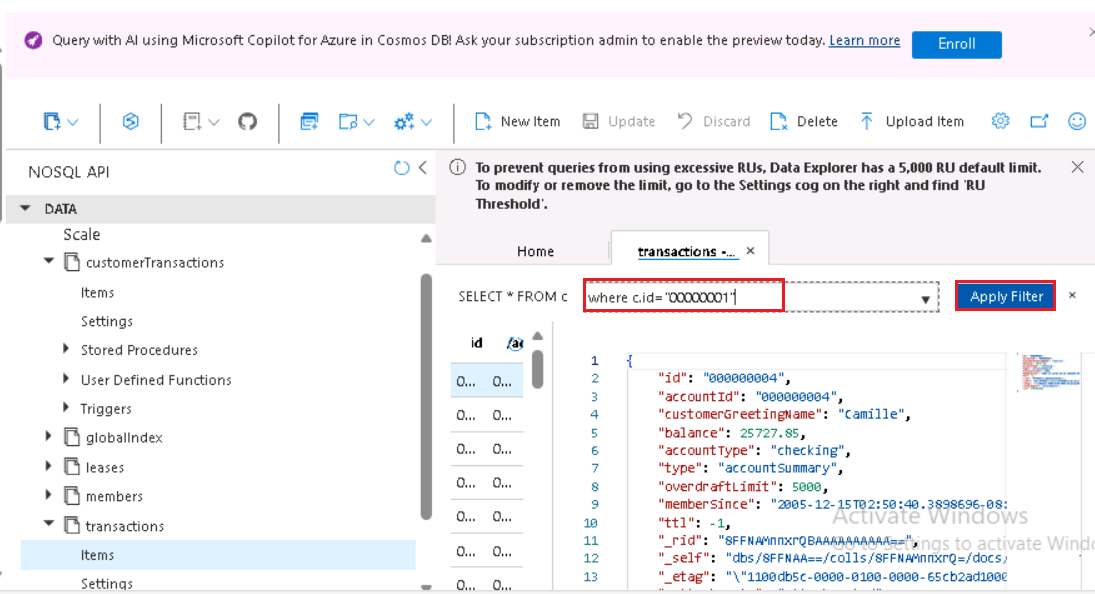
1. Click on **Data Explorer**. Select the **CustomerTransactions** container data.

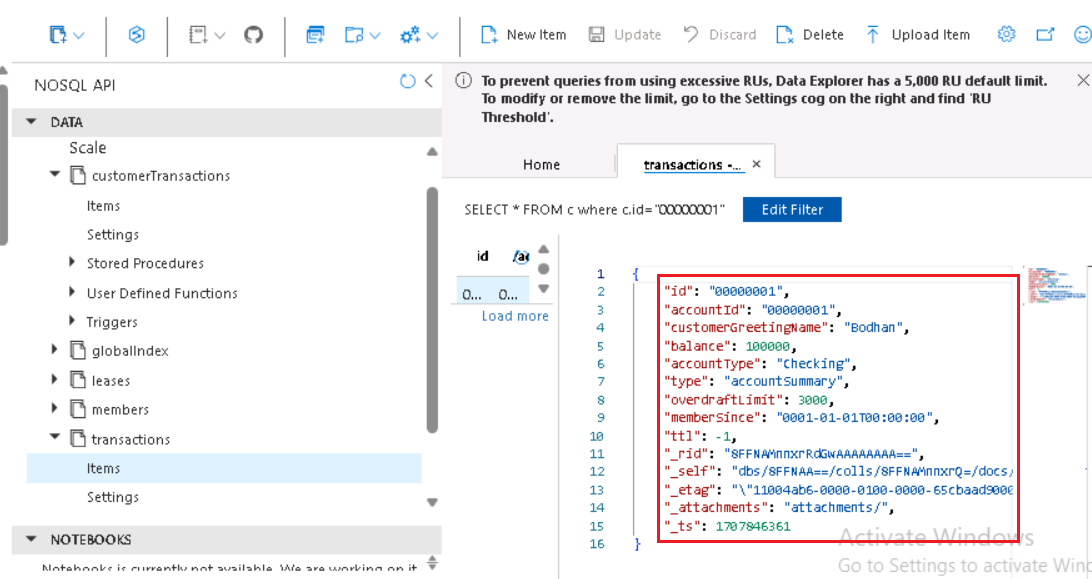


1. Select the **Items** from the **Customer Transactions** and click on **EditFilter.**



1. To check the new added account, Select new added account r using the following code +++ where c.id="0000000001"+++





## Task 3: Remove Copilot functionalities

1. Back in the Visual Studio 2022, from **CorePayments.SemanticKernel**, select the **AnalyticsEngine.cs** file from the **Repository** folder.

**A screenshot of a computer

Description automatically generated**

1. **Comment** out the lines **85**, **108 to 110 and 112** to remove the creation of the semantic function for the review skill.
2. **Comment** the line **var result = (await reviewer.InvokeAsync(context)).Result;**
3. **Add** the below code

+++**string result = string.Empty;**+++

**A screenshot of a computer

Description automatically generated**

## Task 4: Redeploy the code

**Note:** If you already have the **Windows PowerShell** window where the deployment code was executed in Lab 1, ignore the first 3 steps of this exercise and start from Step 4. Else, start from Step 1.

1. Open **Windows PowerShell** as **Administrator**.
2. Run the below command to set the policy to Unrestricted and enter A when asked to change the execution policy.

**+++Set-ExecutionPolicy Unrestricted+++**

A computer screen with white text

Description automatically generated

1. Change the current directory to the **Labfiles** directory and navigate into the project cd .\**Real-time-Payment-Transaction-Processing-at-Scale** folder by running the below commands.

**+++cd\+++**

**+++cd Labfiles+++**

**+++cd .\Real-time-Payment-Transaction-Processing-at-Scale+++**

A computer screen with white text

Description automatically generated

A black and white screen with white text

Description automatically generated

1. Replace **RG\_NAME** with the Resource group name that you created in Task 2 - +++**AOAI-work\_RG**+++XX. Replace the **Subscription ID** with your subscription id, that you noted down in Task 1.

**++./deploy/powershell/Unified-Deploy.ps1 -resourceGroup <RG\_NAME> -subscription <Subscription ID> -locations 'EastUS,WestUS' -deployAks $true++**

1. The above step will re deploy the code, to get the changes we made, updated in the app.

A screenshot of a computer program

Description automatically generated

## Task 5: Test the application

1. Click on the **Accounts** page. Select the **View Transactions** link from any of the entries available.

A screenshot of a computer

Description automatically generated

1. Observe the **Balance** amount and then click on **+ New Transaction**.

A screenshot of a computer

Description automatically generated

1. In the **New Transaction** dialog, enter the below details.

* Name - +++**checkBusinessLogic**+++
* Transaction Type – **Debit**
* Amount - An **amount greater** than the **balance + the overdraft** amount.
* Click on **Save**.

It will pass through since we have **removed** the **business logic** to check the balance and debit/cancel the transaction based on that from the **TransactionRepository.cs**

1. Click on **Analyze Transactions.**

A screenshot of a computer screen

Description automatically generated

1. Click on **Submit** on the **Analyze Transactions** dialog.

A screenshot of a computer

Description automatically generated

1. This will not display any result since we have removed the Analyze Transactions logic from the **AnalyticsEngine.cs**

**Important:** Please **uncomment** the lines of code that you commented in the **Task 3** and comment the line that you added in the **AnalyticsEngine.cs**. After getting the actual code back, please **re-execute Task 4 - Redeploy the code** to have the application as it was earlier.

**Summary:**

In this lab, you have reviewed the code that builds a Copilot to analyze the transactions made by an user. You have tested the functionalities from the website.