# Lab 03: Visualize World (Global) Peace

Now that the data is loaded, it's time to visualize and interact with it. In this lab you wire up the Members, Account Summary, and Transaction Statement functionality to load the data from Cosmos DB and present it in the UI. This gives you the chance to become familiar with core of the application by seeing the end-to-end in action.

Another component to this challenge is completing the Global Index repository. As you know, we're using the NoSQL API. That means that we do not have an enforced schema within the database or containers, and we're also not dealing with a traditional relational database system. The global index container within the Azure Cosmos DB deployment is used to map different entity relationships based on the partition key, ID, and target doc type. This gives us the ability to have flexible lookups and pseudo joins in NoSQL for one-to-few and one-to-one relationships.

The global index gives us a member-to-account mapping based on their IDs and partition keys. When populated, we can search the global index if we want to be able to find all the accounts for a member within the containers for where the member document lives and the account documents live. Without a global index, we would need to execute several cross-partition queries to locate all of the related documents. The global index gives us a quick way to look up that relationship and get the IDs for the item that we're looking for.

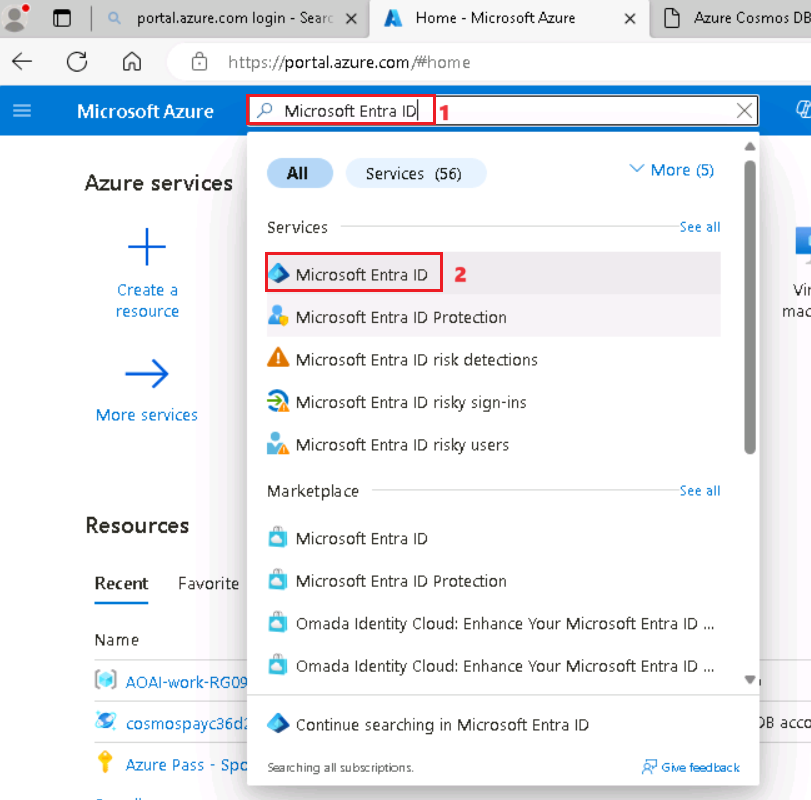
**Objective**

1. Review the Web API code that adds a rule when batch processing a transaction to throw an exception if the transaction amount is greater than the account balance plus the overdraft limit.
2. Review the appropriate repository that implements the global index functionality.
3. Connect the application to the REST API supplied by the Web API.
4. Within the application, explore the members list and details.
5. Create a new member.
   1. Add the new member to an existing account .
   2. Remove the account from the member.
6. Explore the accounts list.
7. For a given account, view the account details.
8. For a given account, view the transactions.
9. Create a debit transaction that is greater than the account balance plus the overdraft limit.

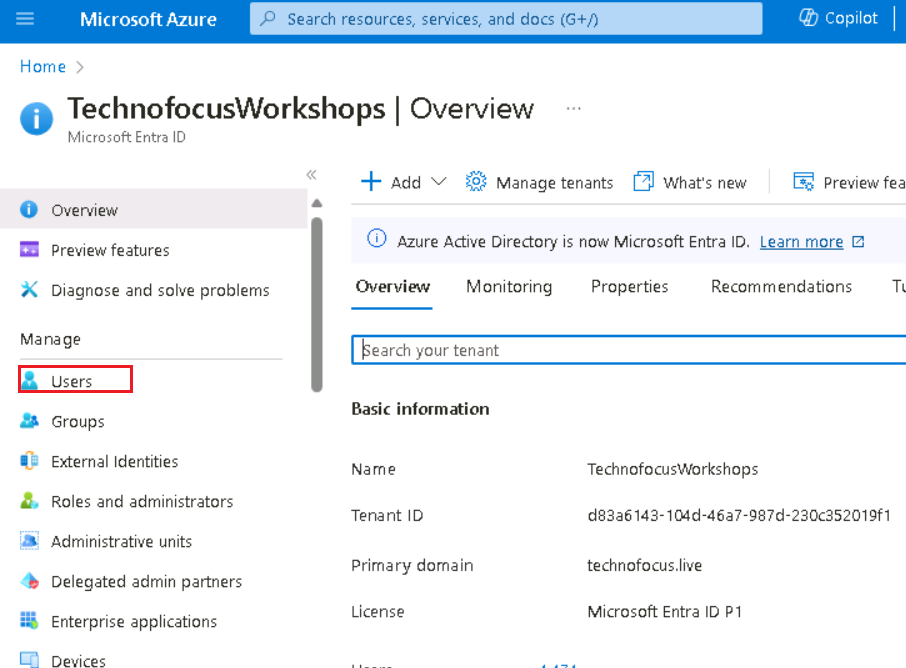
## Task 1: Enabling Role Based Access Control

When you run the solution locally, you will need to set role-based access control (RBAC) permissions on the Azure Cosmos DB account. You can do this by running the following commands in the Azure Cloud Shell or Azure CLI

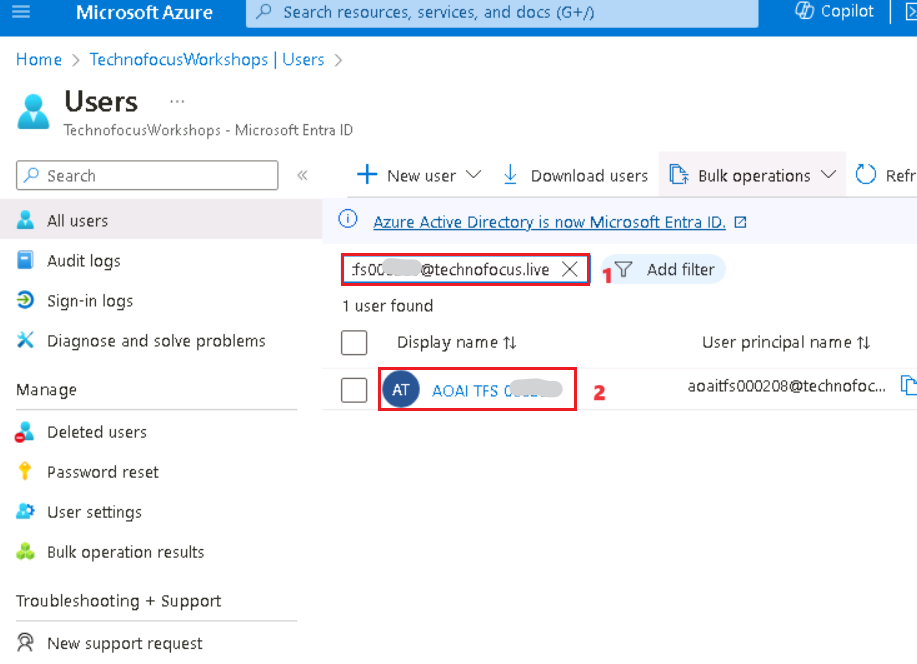
1. Open a browser and go to +++<https://portal.azure.com>+++ and sign in with your Azure login credentials if not logged in already.
2. In the Azure portal search box, type +++**Microsoft Entra ID**+++, then click on **Microsoft Entra ID** under **Services**.



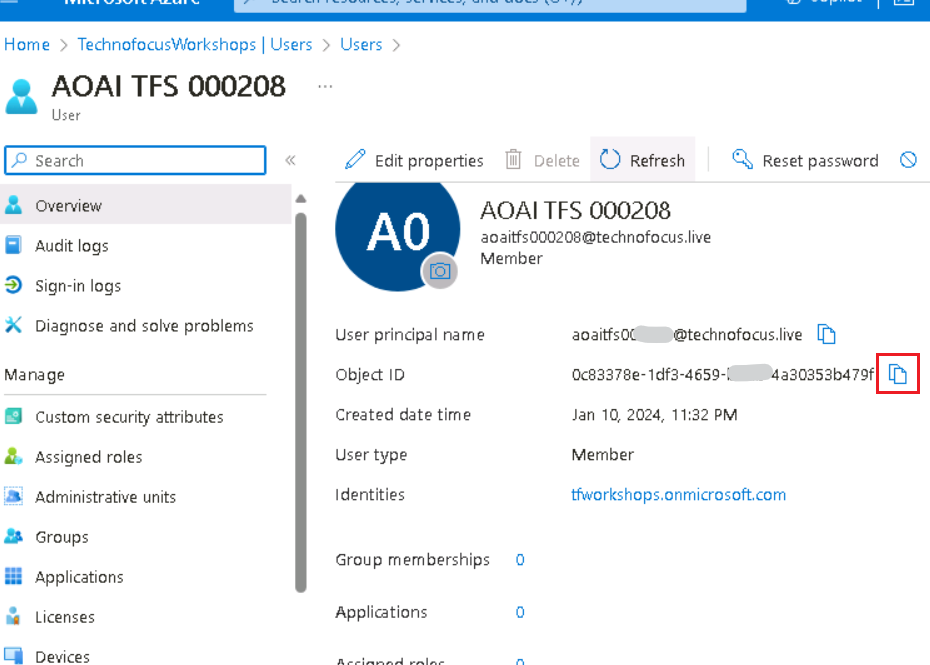
1. In the **Overview** page of Microsoft Entra ID, select **Users** under **Manage** from the left navigation pane.



1. In the Users page, in the search box enter your **azure login credential**(**User name** under **User Credentials** from the **Resources** tab) and select the **DisplayName** with your login id.



1. In the Users page, copy **Object ID** and then **Save** it in the notepad to use the information in the upcoming steps.



1. Open **Windows PowerShell** as administrator, navigate to the **Real-time-Payment-Transaction-Processing-at-Scale** and execute the below command to login to azure.

**Note:** You can ignore this and continue with the next step if you are already logged in from PowerShell and continue from the Step 9.

+**++az login+++**

A screenshot of a computer program

Description automatically generated

1. Now it opens the default browser to sign in with your Azure subscription account.

A screenshot of a computer

Description automatically generated

1. Switch back to PowerShell window, you should see subscription details.

A screenshot of a computer program

Description automatically generated

1. Run the below command to assign the "**Cosmos DB Built-in Data Contributor**" role, replacing the placeholders with your actual values.

**YOUR\_COSMOS\_DB\_ACCOUNT\_NAME**: Specify your Cosmos DB account name.

**YOUR\_RESOURCE\_GROUP\_NAME**: Provide the name of your resource group.

**YOUR\_AZURE\_AD\_PRINCIPAL\_ID**: The Azure AD Object ID (that you noted down earlier in this task)

+++ **az cosmosdb sql role assignment create --account-name YOUR\_COSMOS\_DB\_ACCOUNT\_NAME --resource-group YOUR\_RESOURCE\_GROUP\_NAME --scope "/" --principal-id YOUR\_AZURE\_AD\_PRINCIPAL\_ID --role-definition-id 00000000-0000-0000-0000-000000000002**+++

A blue screen with white text

Description automatically generated

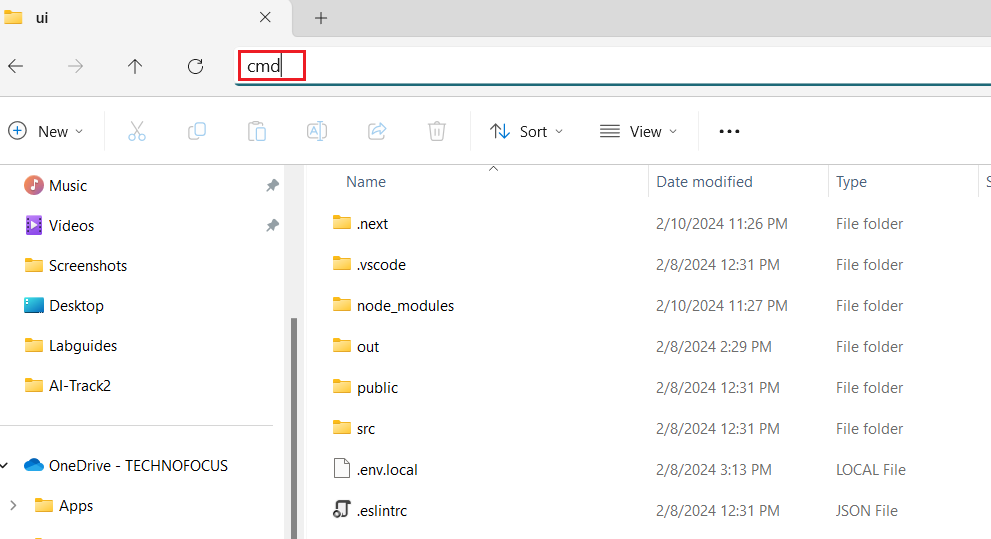
## Task 2: Run the application in the local environment

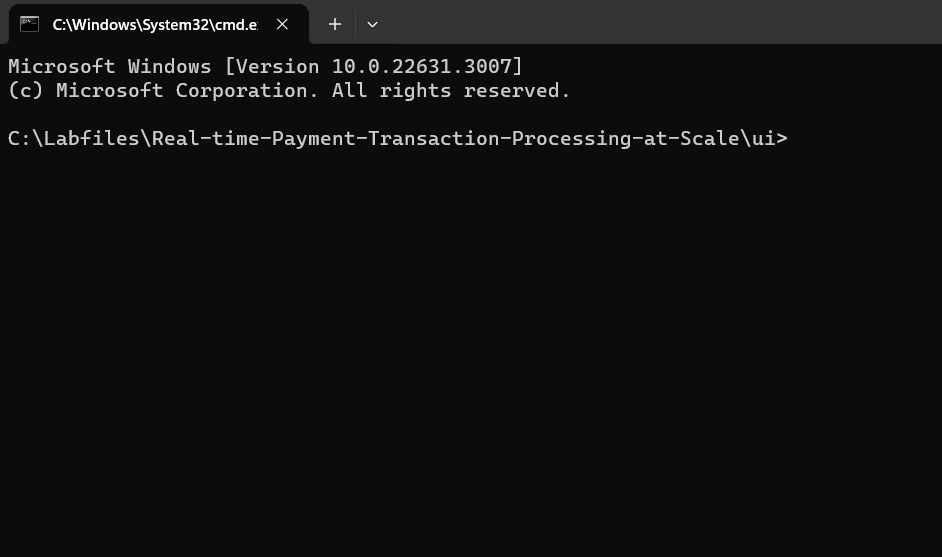
1. Go to **C:\Labfiles\Real-time-Payment-Transaction-Processing-at-Scale** and navigate to the **ui** folder.

A screenshot of a computer

Description automatically generated

1. Type +++**cmd**+++ in the folder path bar to open command prompt window.





1. Run **npm install** to restore the packages.

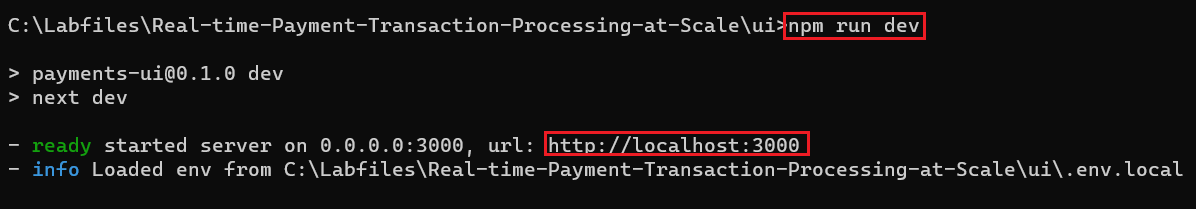
**+++npm install+++**

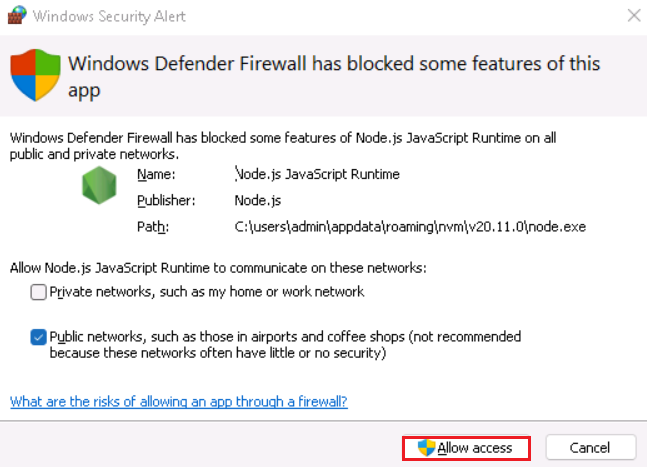
A screenshot of a computer

Description automatically generated

1. Execute the below command and click on **Allow access** when prompted with the security alert.

**+++npm run dev+++**

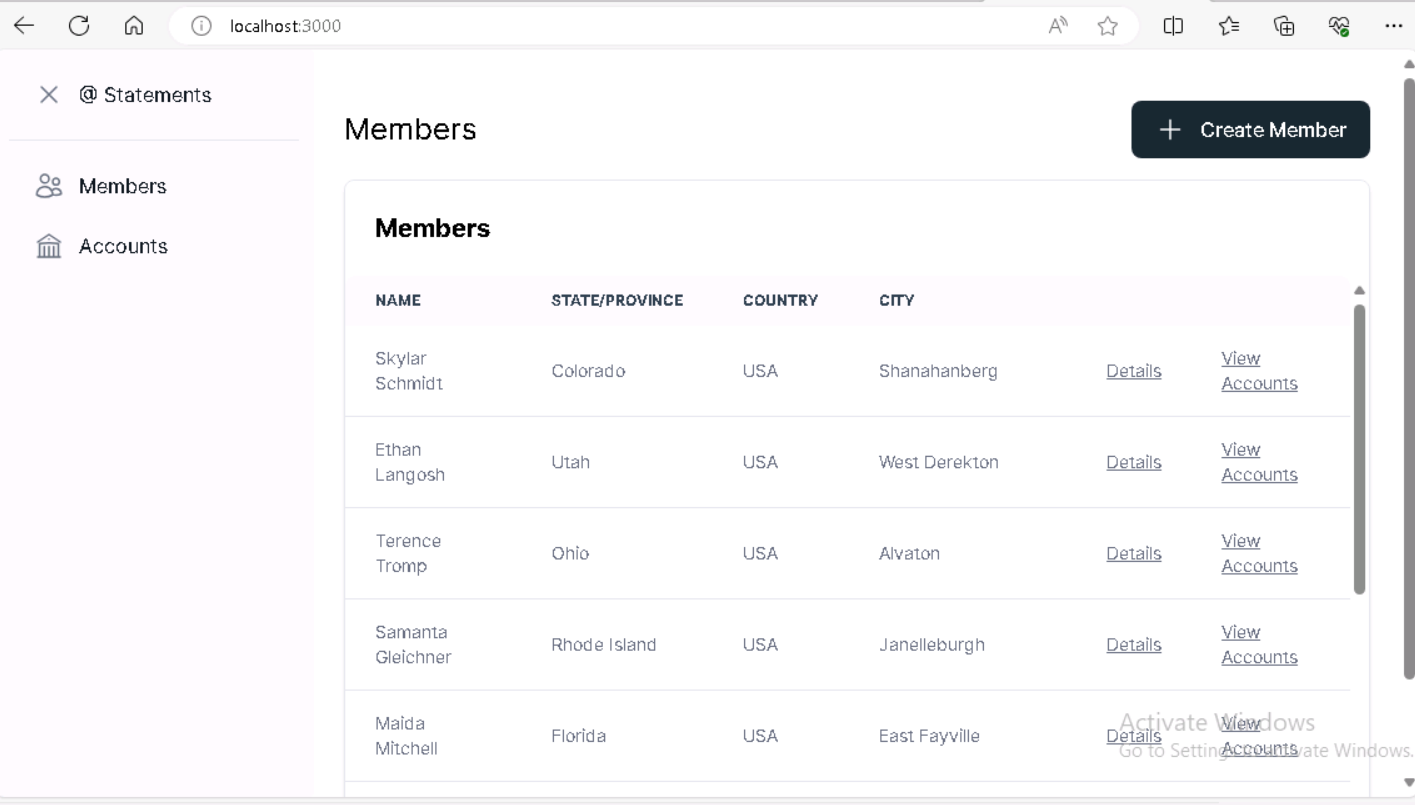




1. Once you get a **ready started server on 0.0.0.0:3000, url:** [**http://localhost:3000**](http://localhost:3000)message**,** open the +++**localhost:3000**+++ in a web browser. You should see the application opened.

A screenshot of a computer screen

Description automatically generated



## Task 3: Configure Visual Studio application settings

To run locally and debug using Visual Studio, open the solution file to load the projects and prepare for debugging.

Before you can start debugging, you need to set the startup projects. To do this,

1. Ensure that the **Docker Desktop** is running. Else, start it.

A screenshot of a computer

Description automatically generated

1. Switch back to visual Studio.

A screenshot of a computer

Description automatically generated

1. Right-click on **Solution ‘CorePayments’** and select **Properties**.

A screenshot of a computer

Description automatically generated

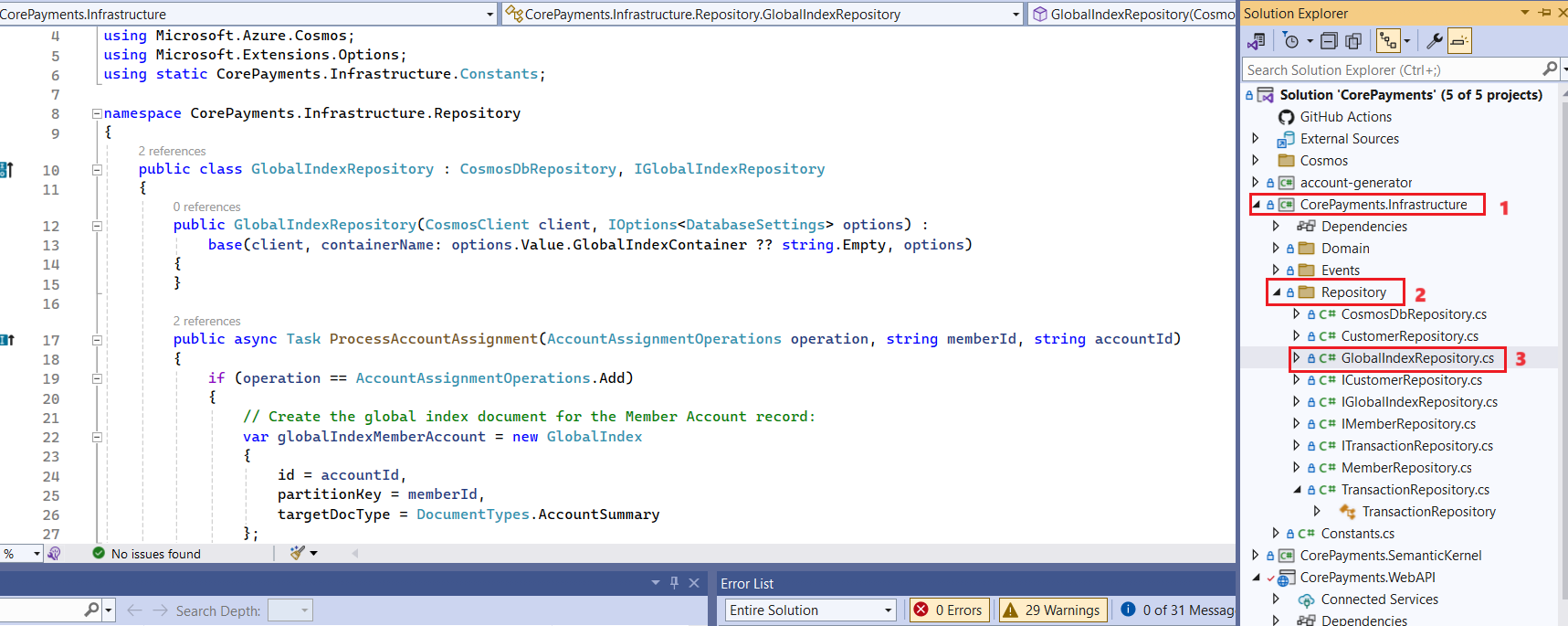
1. In the dialog that opens, select **Multiple startup projects** and set the **Action** for the **CorePayments.WebAPI** and **CorePayments.WorkerService** to **Start**. Click on **Apply** and then select **Ok**.

A screenshot of a computer

Description automatically generated

## Task 4: Understanding the implementation of GlobalIndex and updating transactions

1. In the **CorePayments.Infrastructure**,Select the **GlobalIndexRepository.cs** file from the **Repository** folder



1. Go through the **ProcessAccountAssignment** and **GetAccountsForMember** methods to understand the relationship creation between an Account and a Member and vice versa, perform a read to retrieve the global index document and query based on an Account Summary.

public async Task ProcessAccountAssignment(AccountAssignmentOperations operation, string memberId, string accountId)

{

if (operation == AccountAssignmentOperations.Add)

{

// Create the global index document for the Member Account record:

var globalIndexMemberAccount = new GlobalIndex

{

id = accountId,

partitionKey = memberId,

targetDocType = DocumentTypes.AccountSummary

};

// Create the global index document for the Account Member record:

var globalIndexAccountMember = new GlobalIndex

{

id = memberId,

partitionKey = accountId,

targetDocType = DocumentTypes.Member

};

// Cannot do a batch operation because the primary keys are different.

await Container.CreateItemAsync(globalIndexMemberAccount, new PartitionKey(globalIndexMemberAccount.partitionKey));

await Container.CreateItemAsync(globalIndexAccountMember, new PartitionKey(globalIndexAccountMember.partitionKey));

return;

}

// Perform a point read to retrieve the global index document for the Member Account record if it exists:

var pk = new PartitionKey(memberId);

var responseReadGlobalIndex = await Container.ReadItemAsync<GlobalIndex>(accountId, pk);

var globalIndexMemberAccountToDelete = responseReadGlobalIndex.Resource;

// Perform a point read to retrieve the global index document for the Account Member record if it exists:

pk = new PartitionKey(accountId);

responseReadGlobalIndex = await Container.ReadItemAsync<GlobalIndex>(memberId, pk);

var globalIndexAccountMemberToDelete = responseReadGlobalIndex.Resource;

// Delete the global index records.

if (globalIndexMemberAccountToDelete != null)

{

await Container.DeleteItemAsync<GlobalIndex>(globalIndexMemberAccountToDelete.id,

new PartitionKey(globalIndexMemberAccountToDelete.partitionKey));

}

if (globalIndexAccountMemberToDelete != null)

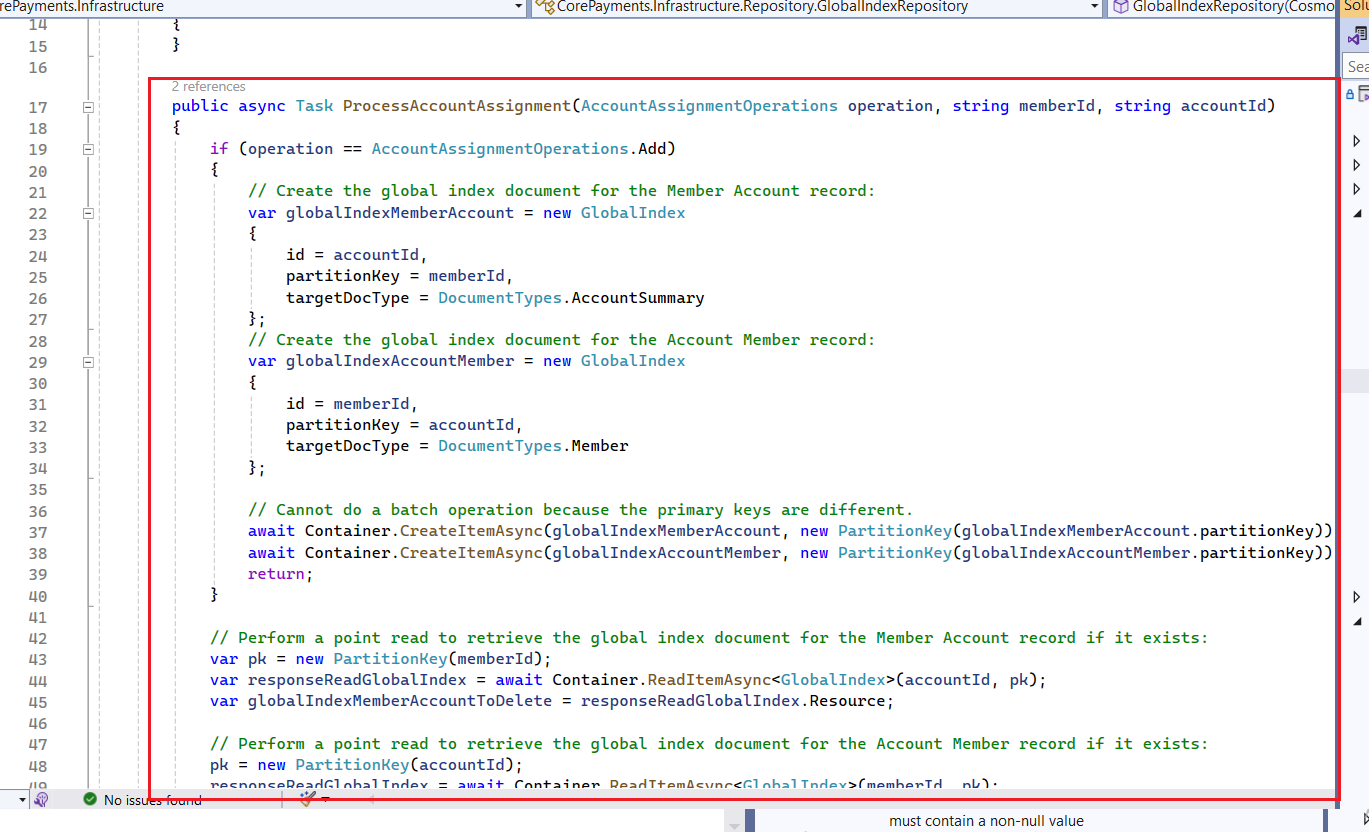
{

await Container.DeleteItemAsync<GlobalIndex>(globalIndexAccountMemberToDelete.id,

new PartitionKey(globalIndexAccountMemberToDelete.partitionKey));

}

}



public async Task<IEnumerable<GlobalIndex>> GetAccountsForMember(string memberId)

{

QueryDefinition query = new QueryDefinition("select \* from c where c.partitionKey = @memberId and c.targetDocType = @docType order by c.id")

.WithParameter("@memberId", memberId)

.WithParameter("@docType", Constants.DocumentTypes.AccountSummary);

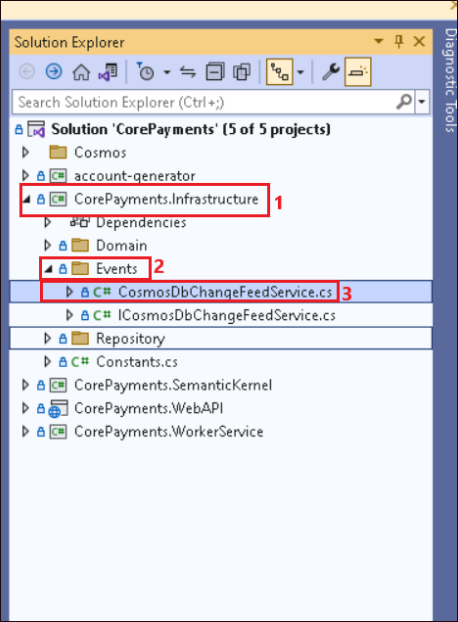
return await Query<GlobalIndex>(query);

//return new List<GlobalIndex>();

}



1. In the **CorePayments.Infrastructure**, select the **CosmosDbChangeFeedService.cs** file from the **Events** folder.



1. Go through the **ProcessCustomerViewChangeFeedHandler** method to understand the

"passthrough" pattern, the method of writing the document from the change feed directly to the **customerTransactions** container with no modifications to the document.

We are taking this step to implement a CQRS pattern, where the **customerTransactions** container is the "read" side of the pattern and the **transactions** container is the "write" side of the pattern. This pattern is used to optimize the read and write operations for each container, minimizing impact on potentially heavy write operations.

private async Task ProcessCustomerViewChangeFeedHandler(

ChangeFeedProcessorContext context,

IReadOnlyCollection<JObject> input,

CancellationToken cancellationToken)

{

using var logScope = \_logger.BeginScope("Cosmos DB Change Feed Processor: ProcessCustomerViewChangeFeedHandler");

\_logger.LogInformation("Cosmos DB Change Feed Processor: Processing {count} changes...", input.Count);

await Parallel.ForEachAsync(input, cancellationToken, async (record, token) =>

{

try

{

await \_customerRepository.UpsertItem(record);

}

catch (Exception ex)

{

//Should handle DLQ

\_logger.LogError(ex.Message, ex);

}

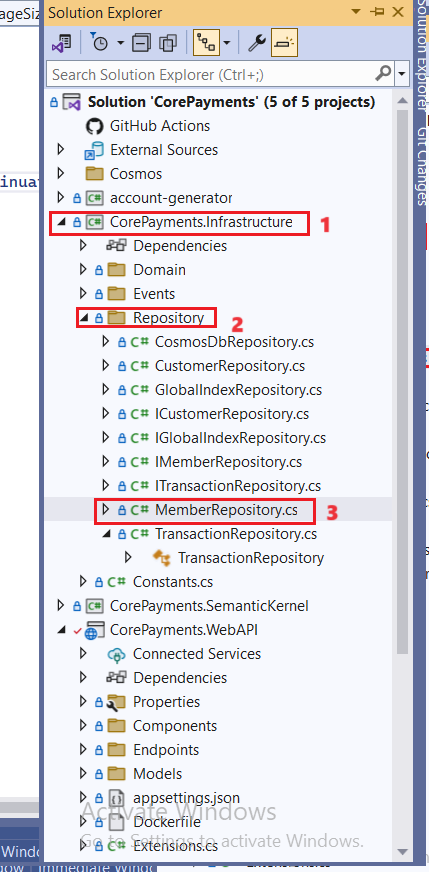
});

}

A screenshot of a computer

Description automatically generated

1. In the **CorePayments.Infrastructure**, select the **MemberRepository.cs** file from the **Repository** folder.



1. Go through the **PatchMember** method which evaluates the item.Path and item.ToString() values and create a new **PatchOperation** with the path and value to add the item.

public async Task<int> PatchMember(Member member, string memberId)

{

JObject obj = JObject.FromObject(member);

var ops = new List<PatchOperation>();

foreach (JToken item in obj.Values())

{

if (item.Path is "id" or "memberId" or "type" || string.IsNullOrEmpty(item.ToString()))

continue;

ops.Add(PatchOperation.Add($"/{item.Path}", item.ToString()));

}

if (ops.Count == 0)

return 0;

var response = await Container.PatchItemAsync<Member>(memberId, new PartitionKey(memberId), ops);

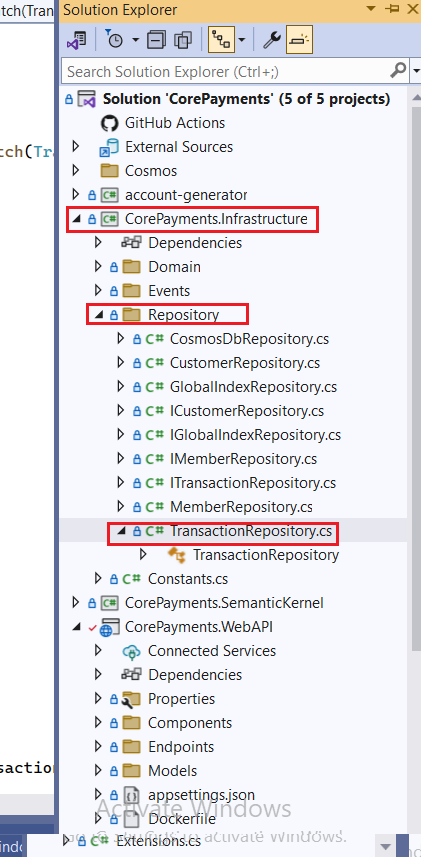
return ops.Count;

}

A computer screen shot of a computer code

Description automatically generated

1. In the **CorePayments.Infrastructure**, select the **TransactionRepository.cs** file from the **Repository** folder.

****

1. Go through the **Task** which creates a business rule to check if the transaction type is "debit" and throw an exception if the transaction amount is greater than the account balance plus the overdraft limit.

public async Task<(AccountSummary? accountSummary, HttpStatusCode statusCode, string message)> ProcessTransactionTBatch(Transaction transaction)

{

var pk = new PartitionKey(transaction.accountId);

var responseRead = await ReadItem<AccountSummary>(transaction.accountId, transaction.accountId);

var account = responseRead.Resource;

if (account == null)

{

return new(null, HttpStatusCode.NotFound, "Account not found!");

}

if (transaction.type.ToLowerInvariant() == Constants.DocumentTypes.TransactionDebit)

{

if ((account.balance + account.overdraftLimit) < transaction.amount)

{

return new(null, HttpStatusCode.BadRequest, "Insufficient balance/limit!");

}

else

{

account.balance -= transaction.amount;

}

}

var batch = Container.CreateTransactionalBatch(pk);

batch.PatchItem(account.id,

new List<PatchOperation>()

{

PatchOperation.Increment("/balance", transaction.type.ToLowerInvariant() == Constants.DocumentTypes.TransactionDebit ? -transaction.amount : transaction.amount)

},

new TransactionalBatchPatchItemRequestOptions()

{

IfMatchEtag = responseRead.ETag

}

);

batch.CreateItem<Transaction>(transaction);

var responseBatch = await batch.ExecuteAsync();

if (responseBatch.IsSuccessStatusCode)

{

account = responseBatch.GetOperationResultAtIndex<AccountSummary>(0).Resource;

return new(account, HttpStatusCode.OK, string.Empty);

}

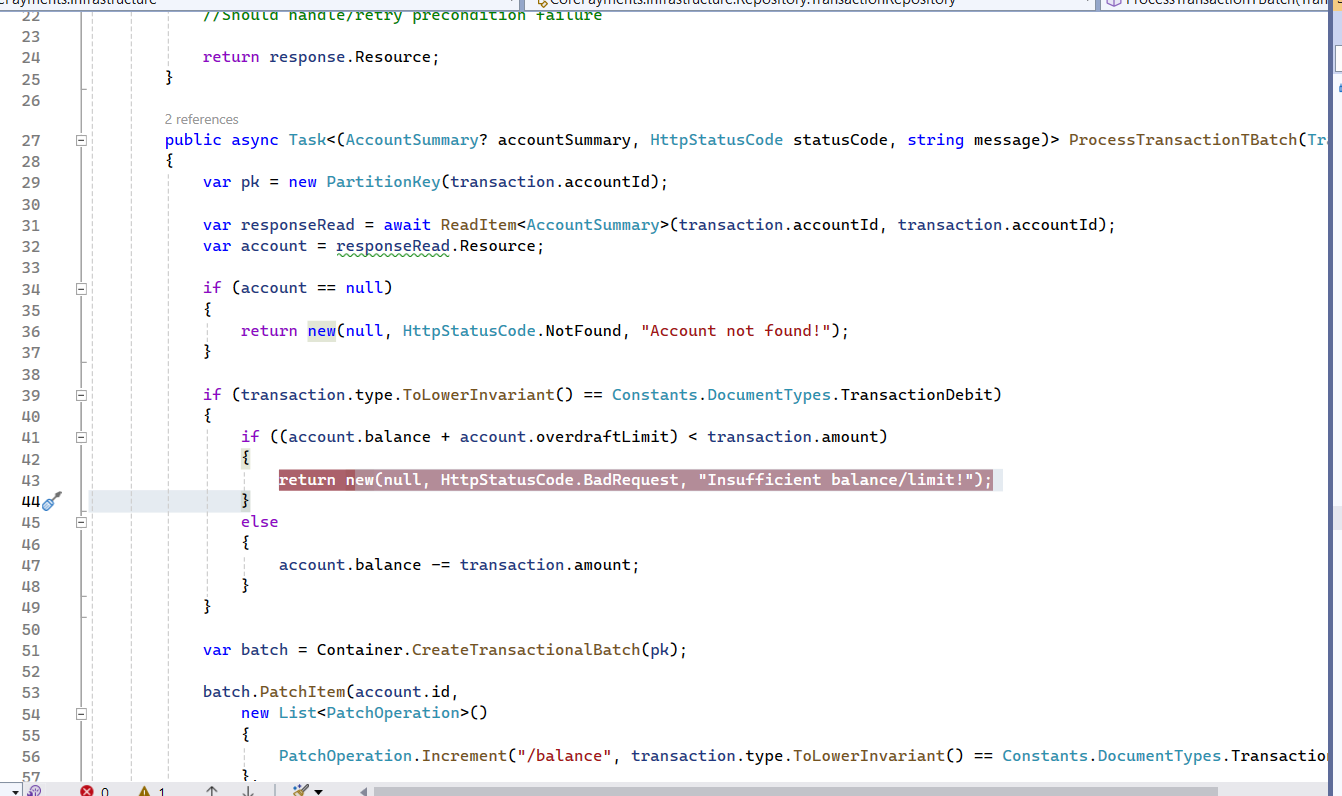
else if (responseBatch.StatusCode == HttpStatusCode.PreconditionFailed)

return new (null, HttpStatusCode.PreconditionFailed, string.Empty);

else

return new (null, HttpStatusCode.BadRequest, string.Empty);

}



1. You are now ready to start debugging the solution locally. To do this, press **F5** or select **Debug > Start Debugging** from the menu.

A screenshot of a computer

Description automatically generated

1. In the Trust ASP.NET Core SSL Certificate dialog, click on **Yes**.

A screenshot of a computer error

Description automatically generated

1. Click on **Yes** in the Security Warning dialog box.

A screenshot of a computer security warning

Description automatically generated

1. If it prompts you to trust the IIS Express SSL certificate then click on **Yes**

A screenshot of a computer

Description automatically generated

1. Click on **Yes** in the Security Warning dialog box.

A screenshot of a computer program

Description automatically generated

1. The build gets started and the app tries to open in the browser.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. First, **Your connection isn’t private** message is displayed. Please wait for some time(around 10 minutes) and refresh and then the app gets opened.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Task 5: Exploring the solution

Run the solution locally and walk through the following:

The application frontend is a React JavaScript single-page app (SPA) that simulates a business-facing web application for managing members, accounts, and transactions.

1. Browse to +++<http://localhost:3000>+++.

A screenshot of a computer

Description automatically generated

1. Observe how paging works (**CosmosDbRepository.PagedQuery** method), which uses continuation tokens to efficiently fetch additional records from Cosmos DB. This is more performant than traditional paging methods, especially once you've paged through a large number of records.
2. Click **Details** for a member and select **Edit** to display the form. We use a Patch operation instead of an update in the back end as opposed to an update on the Cosmos DB record. This is a more efficient operation. Plus, it is easier to reconcile conflicts if more than one person applies updates on a record simultaneously.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

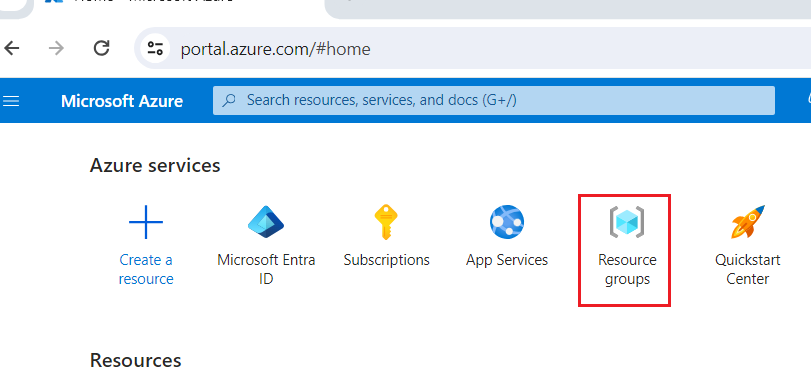
Description automatically generated

1. Edit the member enter the **city** is **+++Salt Lake City+++ , state/Province** is **+++Utah+++** and **Zip code** is +++***72658*+++**. Then click on the **Save** button.

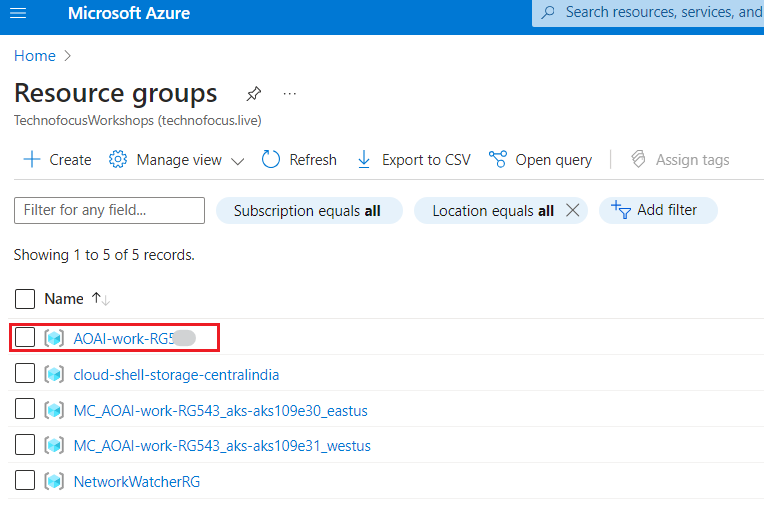
A screenshot of a computer

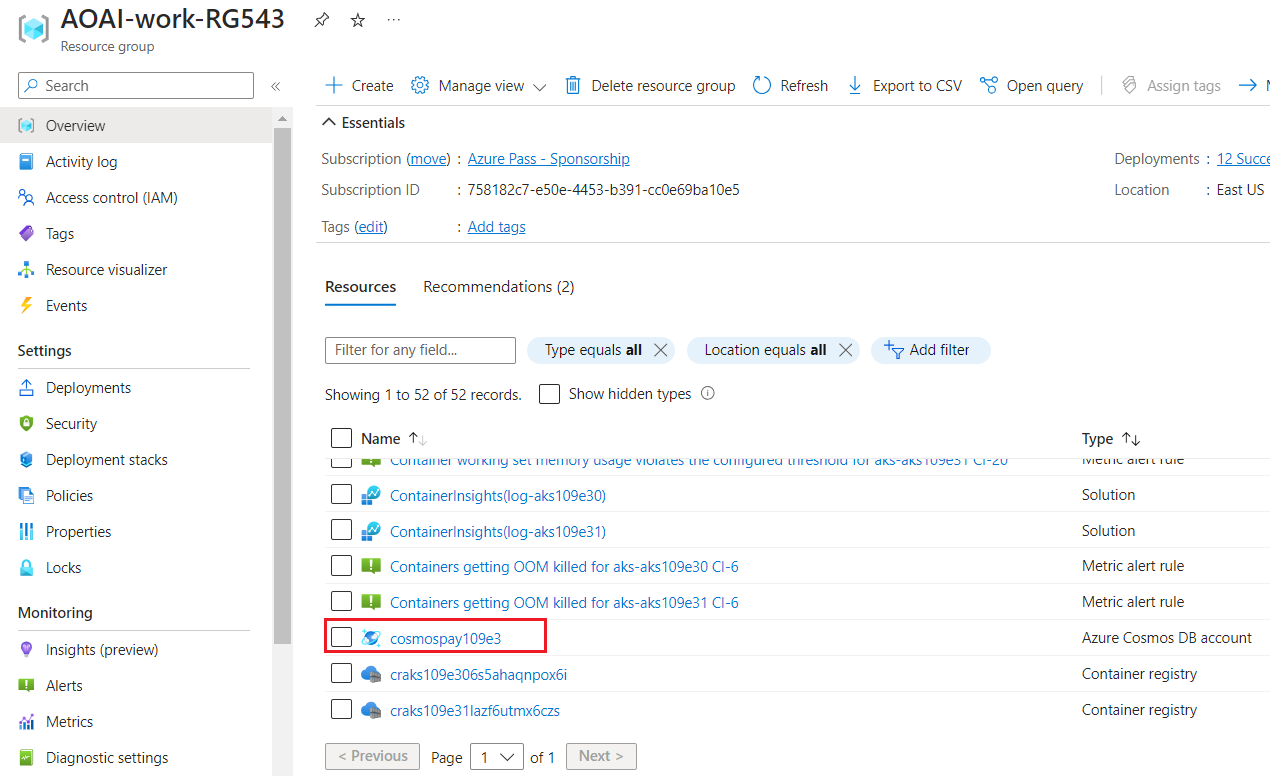
Description automatically generated

1. The changes gets updated in the app.
2. Then, open a browser and go to +++<https://portal.azure.com>+++ and sign in with your Azure login credentials.
3. On the **Home** page, click on **Resource groups** tile.

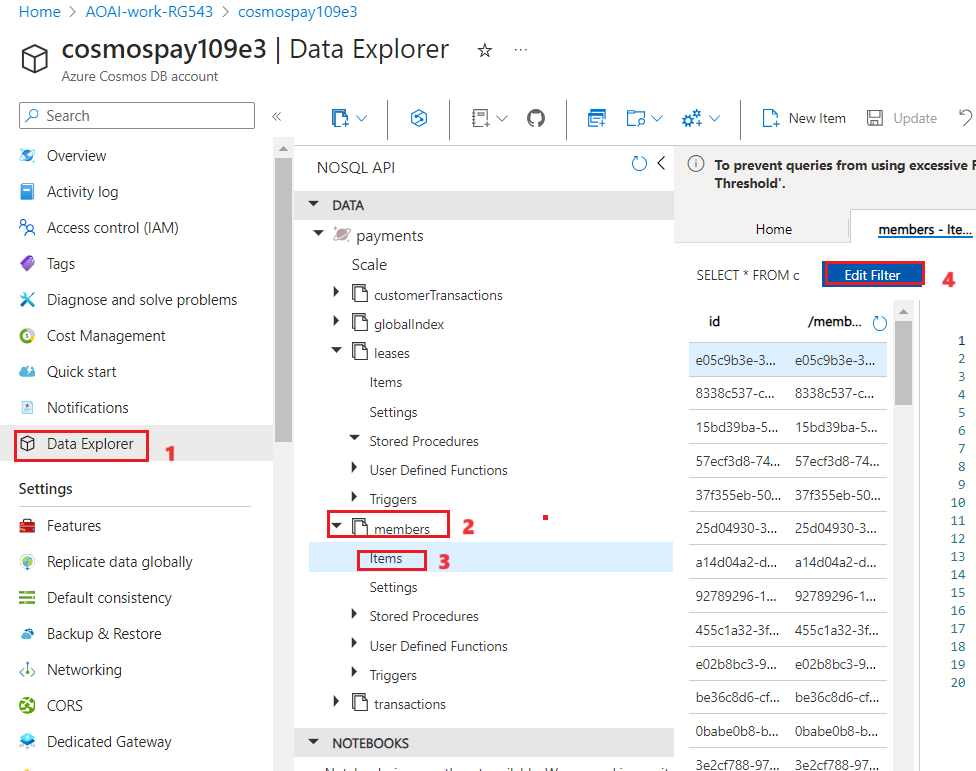


1. Click on your **resource group** name and then click on the **Azure** **Cosmos DB** account name from the list of resources .





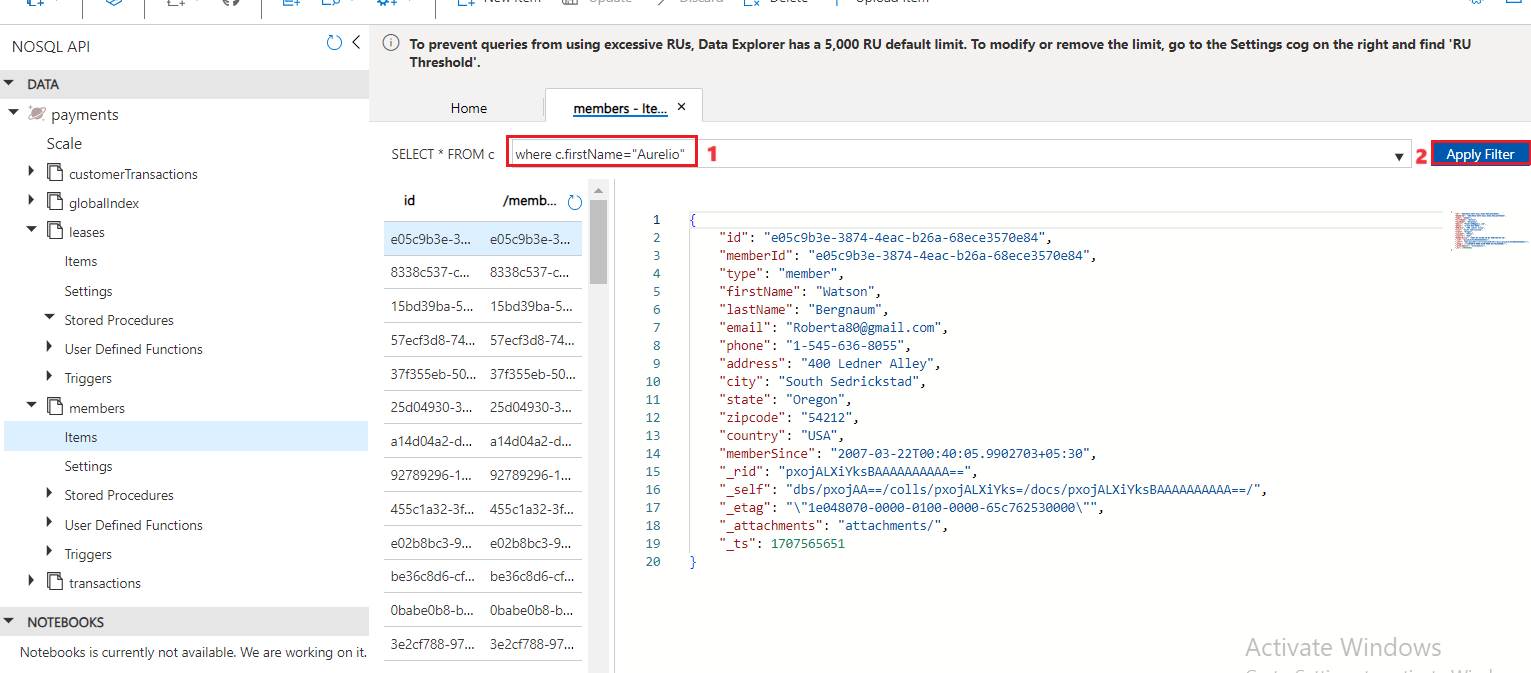
1. Click on **Data Explorer,** Select **Items** under the **members** container. Then, click on **Edit Filter.**



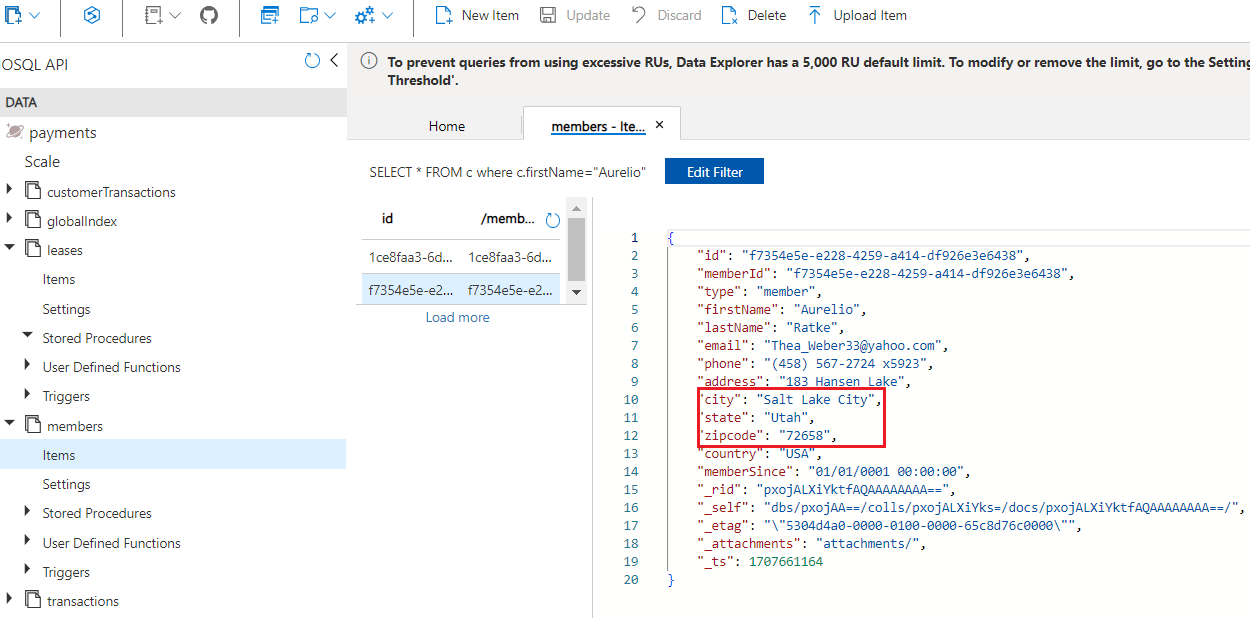
1. Add the following code in the text box that is enabled by clicking the Edit Filter button. Click on **Apply Filter**.

+++ where c.firstName="Aurelio"+++

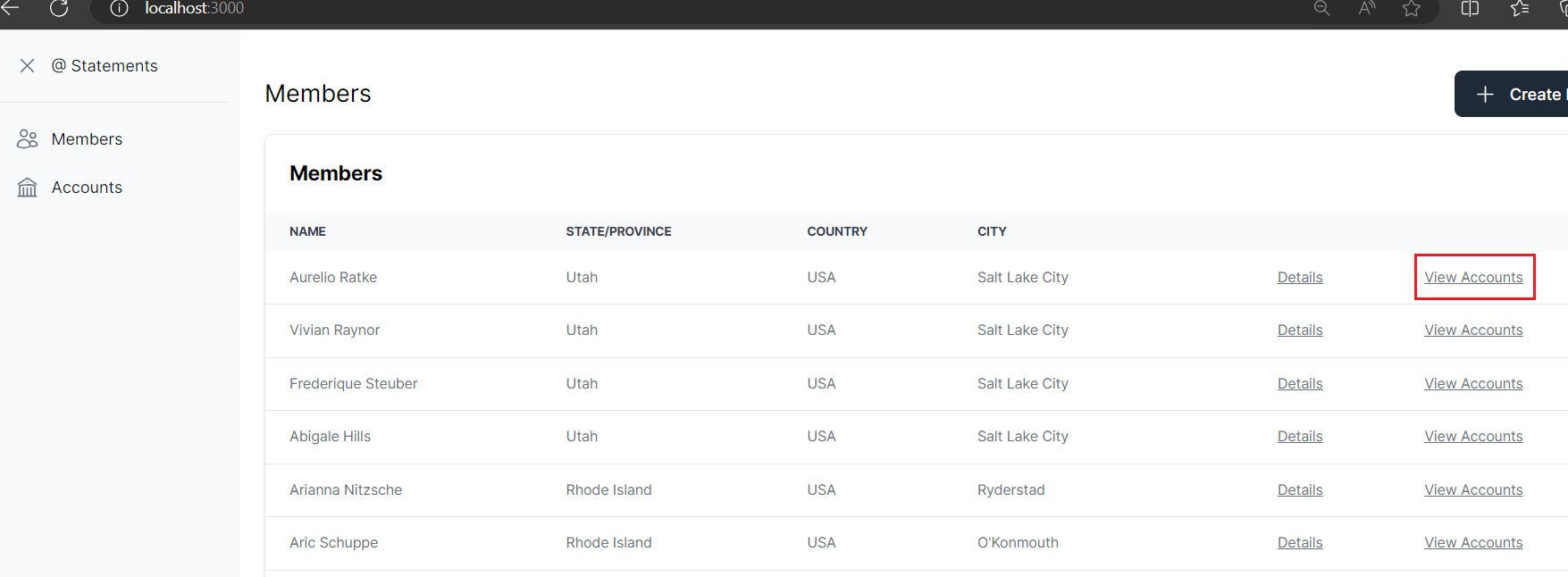
**Note:** Replace the **name** here within the double codes with the **name of the record that you updated** in step 4.



1. Note that the records have been updated.



1. Go back to the local host page and select **View Accounts.**

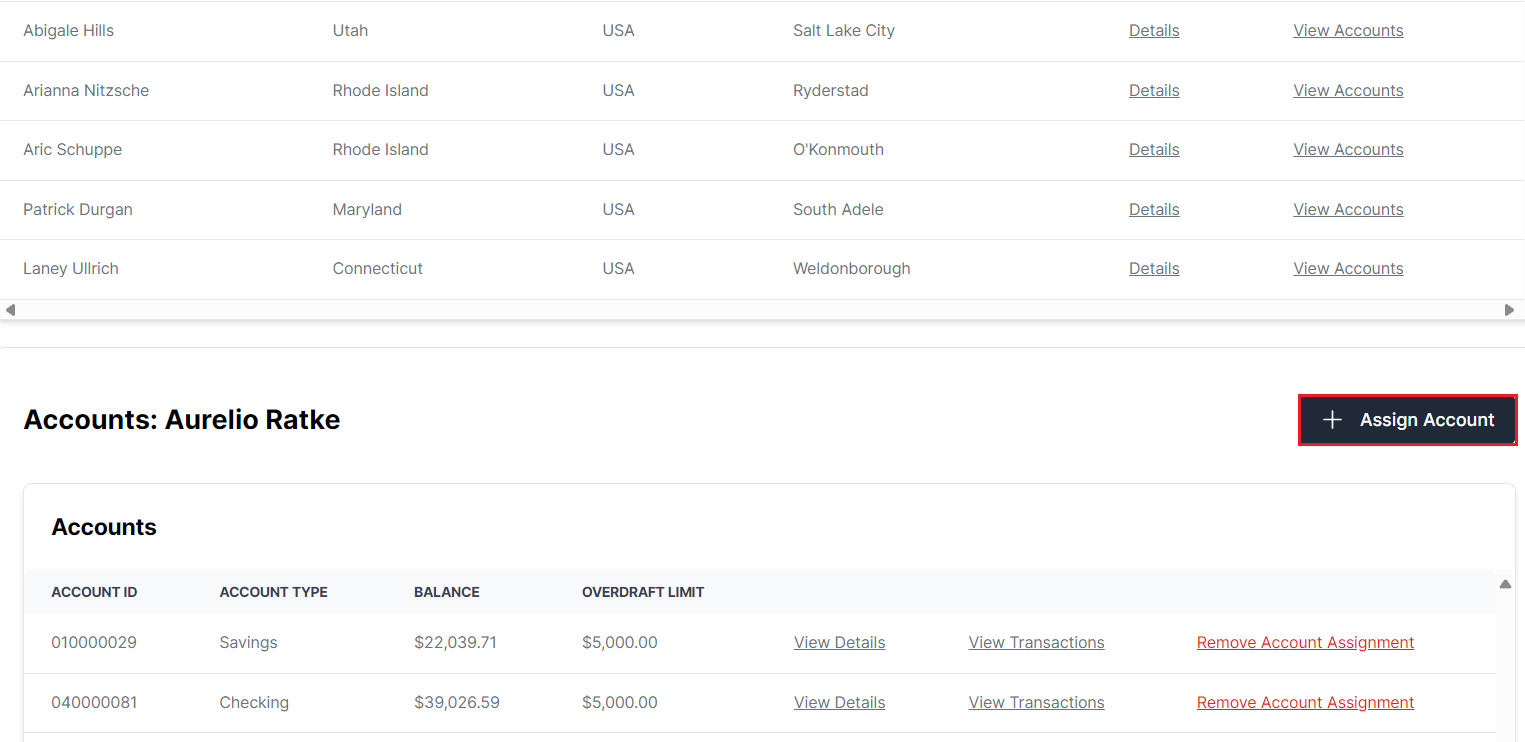
****

**A screenshot of a computer

Description automatically generated**

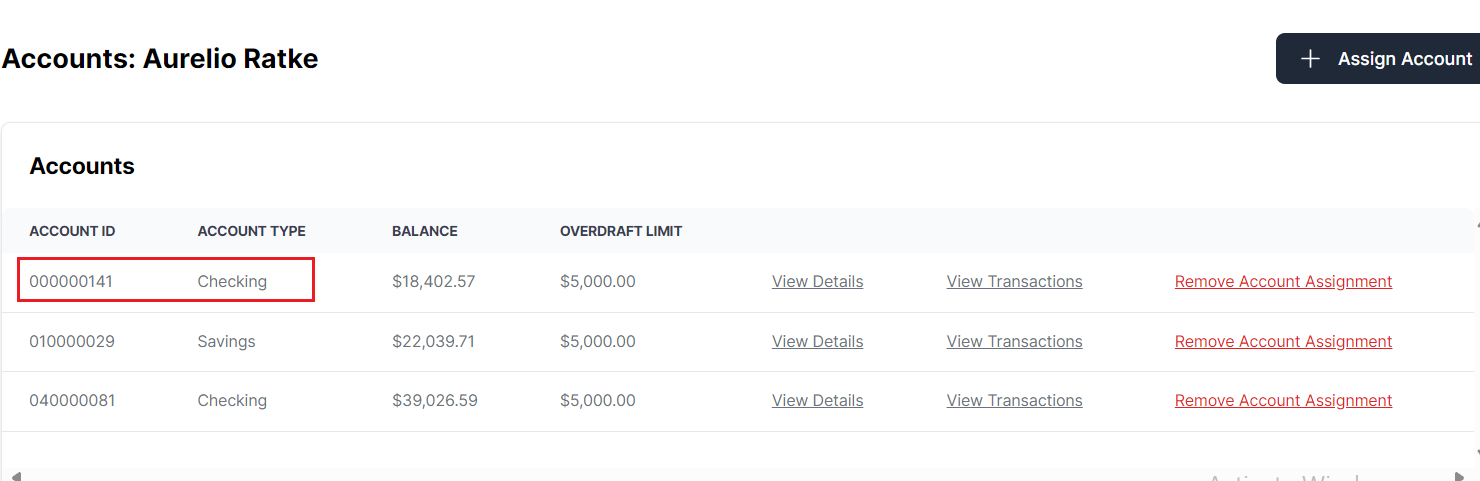
1. Click on **Assign Account**and search for an existing account number.

Explore how **global index** works (GlobalIndexRepository.cs) and how there is a **many-to-many relationship** between **members and accounts**. Think of family members in a household who may have access to a shared account and have their own accounts that only they can access. Or a business account managed by multiple users.

****

1. Select any account and click on the **Save** button. It will create a new **Global index** entry.

****

****

1. Click on **Remove Account Assignment.**

**A screenshot of a computer screen

Description automatically generated**

**A white background with black text

Description automatically generated**

1. Click on **View Details** to display a modal containing account details. This is a good place to mention the Overdraft Limit and how the overdraft rule applies to new transactions.

A screenshot of a computer

Description automatically generated

A screenshot of a account

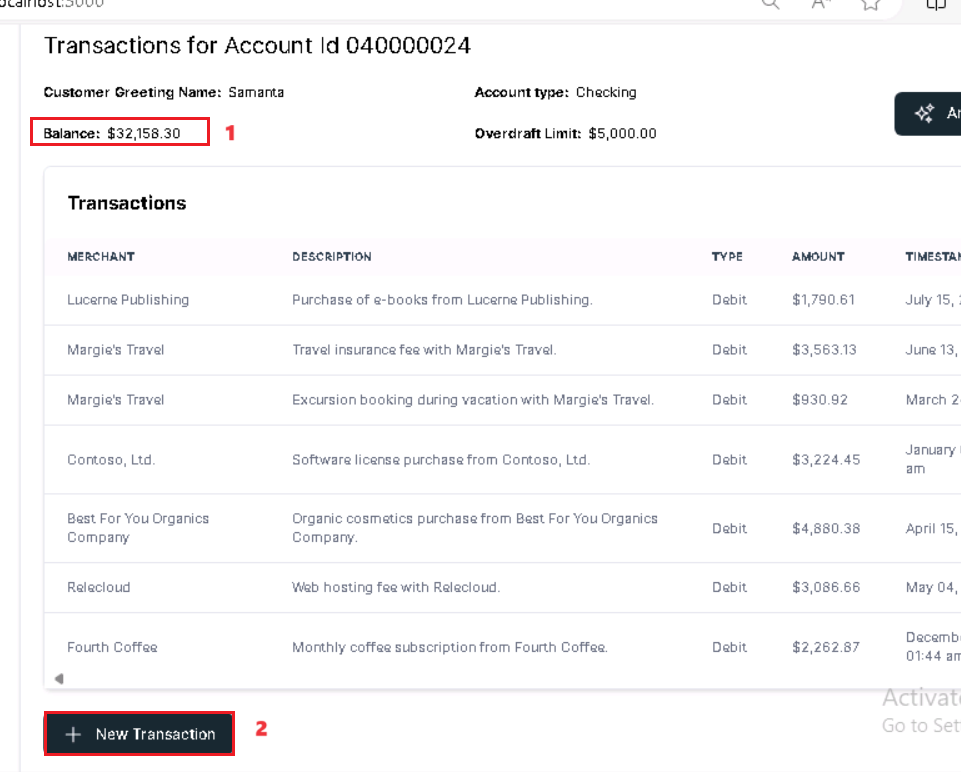
Description automatically generated

1. Click on **View Transactions** and observe the balance.

A screenshot of a computer

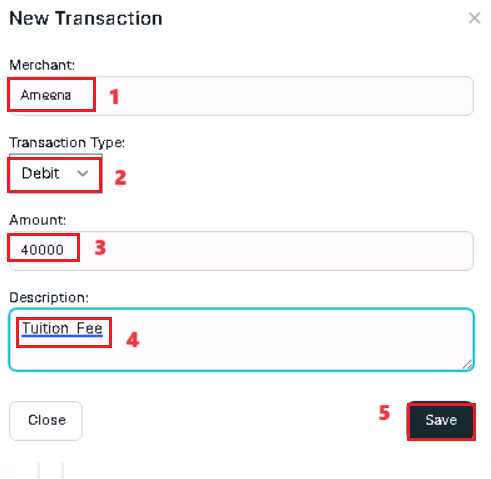
Description automatically generated

1. Now observe the balance and click on **+New Transaction**.

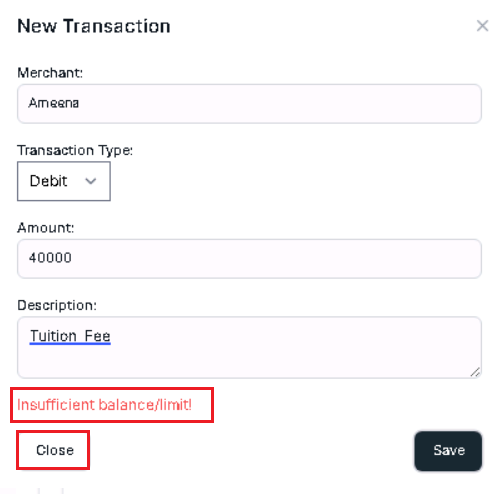


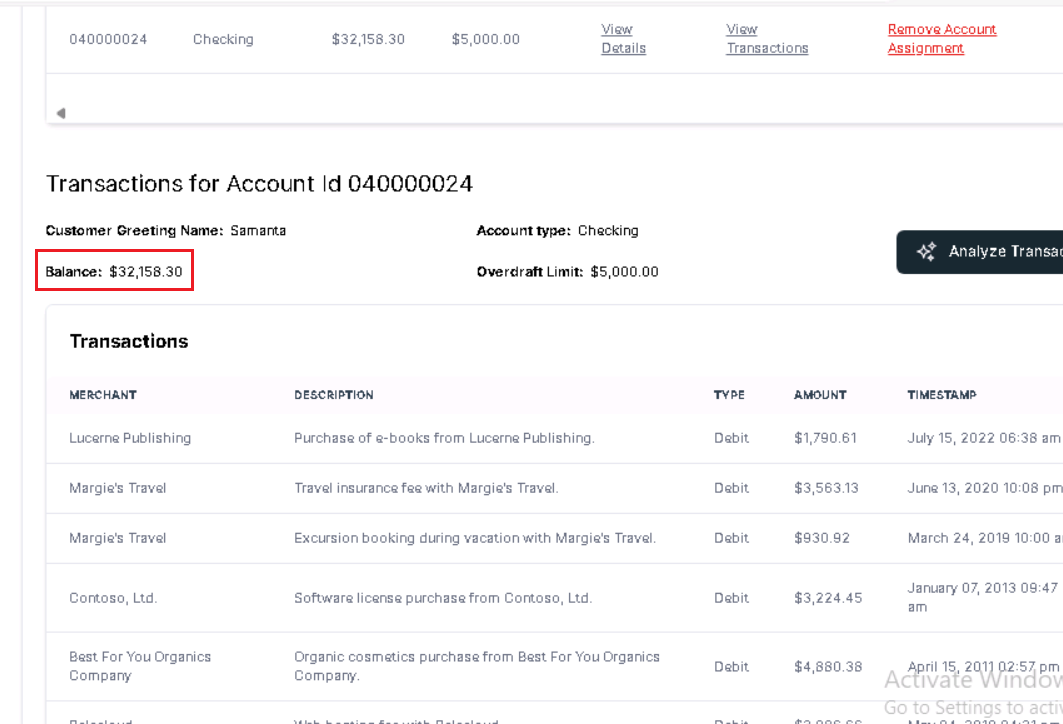
1. In the New Transaction tab, enter the below details.

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



1. Now you will get an error **Insufficient balance/limit**. Click on the **Close** button.





1. Click on the **Accounts** page and select **View Transactions** for one of the Accounts.

A screenshot of a computer

Description automatically generated

1. Click on **+ New Transaction** to display the modal and enter a new transaction.

A screenshot of a computer

Description automatically generated

1. Enter the new transaction details and click on the **Save** button.

Merchant – +++**Aimee Gibson**+++

Transaction Type – Debit

Amount – +++22000+++

A screenshot of a computer

Description automatically generated

1. Now the amount has transferred successfully.

A screenshot of a computer

Description automatically generated

**Task 6: Remove the global index, upsertion and business logic concepts from the code**

1. In the **CorePayments.Infrastructure**, select the **GlobalIndexRepository.cs** file from the **Repository** folder.

A computer screen with text

Description automatically generated

1. **Comment** out the lines **24** and **25** which creates the **global index document** for the **Member Account record**.
2. **Comment** out lines **31** and **32** which creates the **global index document** for the **Account Member record**.

A screenshot of a computer

Description automatically generated

1. **Comment** out the lines **67** to **71** that creates a query to retrieve the **global index documents** and add the below line as line no. 72

Return new List< GlobalIndex >();

**Remove** the **space** between **< and G** and **x and >** after pasting the above command.

A computer screen shot of a computer code

Description automatically generated

1. In the **CorePayments.Infrastructure**, select the **CosmosDbChangeFeedService.cs** file from the **Events** folder.

A screenshot of a computer

Description automatically generated

1. **Comment** out the line number **97** which takes invokes the method that takes care of upserting (insert/update) the document to the **customerTransactions** container via the **customer repository**.

A screenshot of a computer program

Description automatically generated

1. In the **CorePayments.Infrastructure**, select the **MemberRepository.cs** file from the **Repository** folder.

A screenshot of a computer

Description automatically generated

1. **Comment** the lines **51 to 54** to remove the logic that adds the patch operation to the list.

A screenshot of a computer

Description automatically generated

1. In the **CorePayments.Infrastructure**, select the **TransactionRepository.cs** file from the **Repository** folder.

A screenshot of a computer

Description automatically generated

1. **Comment** out the lines **41 to 48** to remove the logic to check if the transaction amount is greater than the account balance plus the overdraft limit.

A screenshot of a computer

Description automatically generated

1. Close the ui console window by pressing **Ctrl+C.**

A computer screen with white text

Description automatically generated

**Task 7: Redeploy the code**

**Note:** If you already have the **Windows PowerShell** window open 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 screenshot of a computer program

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 \ -subscription \ -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

Description automatically generated

**Task 8: Test the application**

1. Open the application using the Website url obtained from the above step.
2. From the Members page, click **Details** for a member and select **Edit** to display the form.

A screenshot of a computer

Description automatically generated

1. Click on **Edit**.

A close-up of a contact us

Description automatically generated

1. Enter the **City** as **Montreal** and click on **Save**.

A screenshot of a member account

Description automatically generated

1. It keeps processing. **Close** the dialog.

A screenshot of a member

Description automatically generated

1. The Edit operation does not work since we have removed the code to perform the **Upsert** from the **MemberRepository.cs**.

A close-up of a contact us

Description automatically generated

1. Click on **View Accounts** for any of the Member from the **Members** page. You can see that it does not retrieve any results.

A screenshot of a computer

Description automatically generated

1. This is because we have removed the relationship between the **Account** and the **Member** from **GlobalIndexRepository.cs**. Since we have delinked them, this operation does not work anymore.
2. 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 bank account

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**

**Important:** Please **uncomment** the lines of code that you commented in the **Task 6** and comment the line that you added in the **GlobalIndexRepository.cs**. After getting the actual code back, please **re-execute Task 7 - Redeploy the code** to have the application as it was earlier.