**QUICKKART – THE E-COMMERCE APPLICATION**

**---------------------------------------------------------------------------------------------------------**

**-: Quickkart** is an *E-Commerce Web Application*.

Users can buy products and commodities using this web app.

**Ultimate Requirement:**

This web app needs to be fully architected in Azure Cloud.

**Technology Stack:**

**Front-End:** Angular 13

**Backend**: Dotnet 3.1

**DB**: Azure SQL Server.

**Blob**: Azure Blob Container.

**TASK-1: SETTING UP THE ENVIRONMENT**

**Goal**:

* Set up the local environment
* Create SQL Database and Server
* Execute the Scripts
* Test the code locally first.
* Deploy it to Azure Paas Services.

**1.** Download Visual Studio Community 2022 & install it with Azure Development SDK and .Net 3.1.

**2**. Download **VS Code** from [Download Visual Studio Code - Mac, Linux, Windows](https://code.visualstudio.com/download)

**3**. install Git from [Git - Downloads (git-scm.com)](https://git-scm.com/downloads)

Check the version by **git -v**

**4.** Install **Node.js** from [Download | Node.js (nodejs.org)](https://nodejs.org/en/download/) and **angular CLI** by the below command

**npm install -g @angular/cli@13.3**

**5.** Check if the version is installed by ng v

Text

Description automatically generated

**6**. Go to the directory where you have cloned the front-end.

**Clone URL:** <https://siddharthdwivedi318@dev.azure.com/siddharthdwivedi318/Experiential%20Learning/_git/Quick-Cart-FrontEnd>

and Execute the npm install command.

This will create the **Node\_Modules**.

**7**. Start the application by ng s -o

**8**. Clone the backend Project

**Clone URL**: <https://siddharthdwivedi318@dev.azure.com/siddharthdwivedi318/Experiential%20Learning/_git/Quick-Cart-Backend>

**9**. Create a new **DB** and **Server**.

Download the DB Scripts from the below link and Execute it in

<https://quickcartstorage.blob.core.windows.net/data/QuickKart-DB.sql>

**10**. Change the connection string in the backend.

**11**. Run the Application locally and Test.

**12**. Deploy the front-end to Azure Static web App and Backend to Azure Web App.

**TASK-2: IMPLEMENT LOGIN FUNCTIONALITY USING AZURE FUNCTION**

**Goal**:

* Learn to build an Azure Function (C#) from IDE
* Deploy code to Azure function.

1. The login code is already provided to you in the form of an Artifact.

<https://quickcartstorage.blob.core.windows.net/data/loginFunction-Experiential.txt>

2. Decide the **Trigger** Type.

3. Develop the Azure function from an **IDE** as a best Practice.

**Note:** Recommended **IDE** is **Visual Studio 2022**.

4. Deploy the **login** code to an Azure function.

**5.** Make changes in the **home-page.service.ts**

Text

Description automatically generated

**6.** **Are you still facing an Error?** Check the Browser Console and fix the error from Azure Portal.

**TASK-3: INTEGRATING BACKEND WITH THE KEY-VAULT USING THE CONCEPT OF SERVICE PRINCIPLE (Client Credentials Flow)**

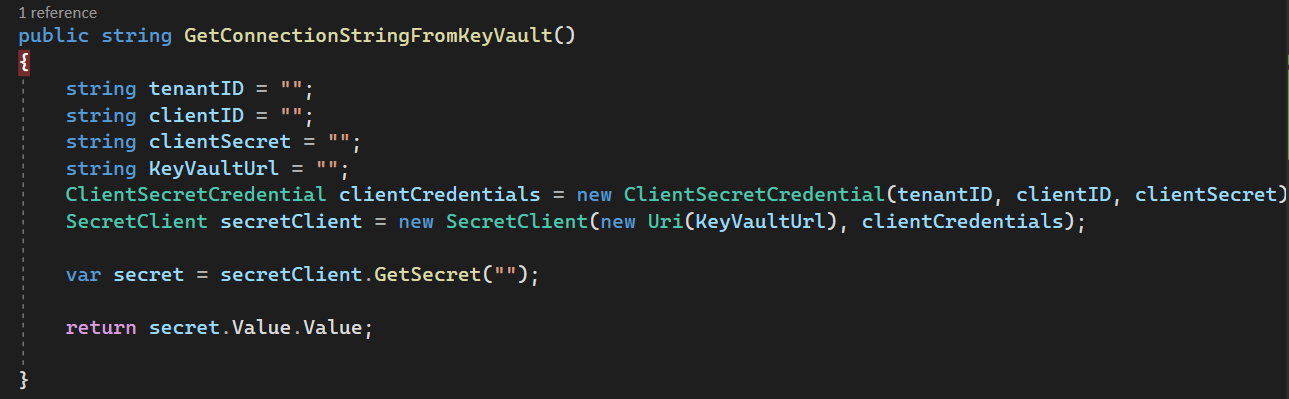
**Goal**:

* Learn to use Keyvault
* Use App Registrations and Permissions.

**1.** Create a new Folder and Clone the latest Code from

<https://siddharthdwivedi318@dev.azure.com/siddharthdwivedi318/Experiential%20Learning/_git/Quick-Cart-Backend>

**2.** Make changes in CustomerRepository.cs of DAL



**3.** Create **KeyVault**, Add the Conncetion String as a Secret and Integrate using Service Principle Concept.

**TASK-4: INTEGRATING BACKEND WITH THE STORAGE ACCOUNT**

**Goal**:

* Learn to use Azure Storage Account from the Backend.
* Learn to use Blob-Life Cycle feature to set up access tier rules.
* Learn to connect the Storage Account with a Logic App.

**1.** Create a new Folder and Clone the latest Code from

**Front-End:**

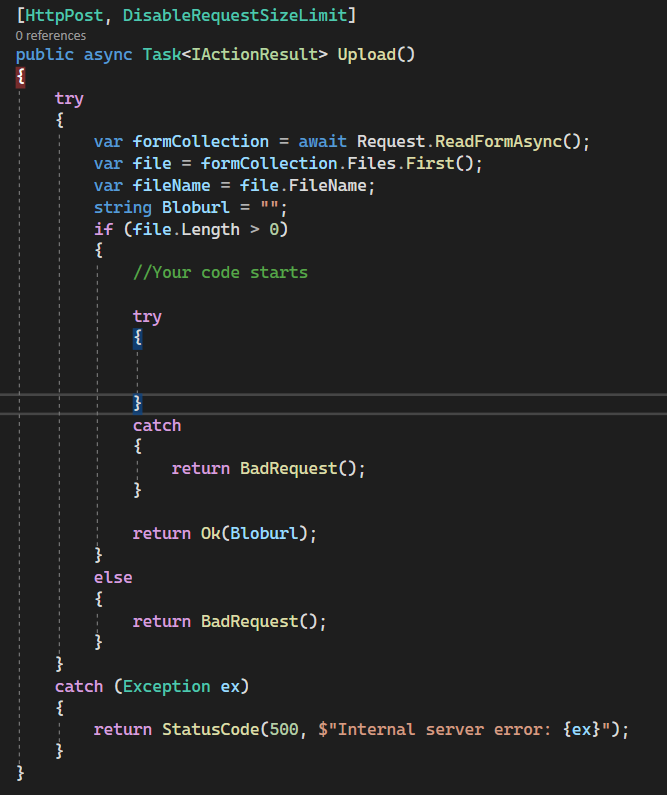
<https://siddharthdwivedi318@dev.azure.com/siddharthdwivedi318/Experiential%20Learning/_git/Quick-Cart-FrontEnd>

**Backend**:

<https://siddharthdwivedi318@dev.azure.com/siddharthdwivedi318/Experiential%20Learning/_git/Quick-Cart-Backend>

**2.** Write the Upload functionality code in **AdminController.cs**.

**Note:** Make use of Access Connection String/ SAS Connection String to establish the connection.



**3.** Test it locally first, whether the upload is working or not.

**4.** If the feature is working then deploy it to respective Azure resources.

**5.** Make sure to set up a Blob Lifecycle rule for that **respective container** only in which your blob is being uploaded.

**The rule should be**

* Upon the upload the Blob should be in Hot Tier.
* From Hot, Move the blob to Cool tier after 2 days,
* From cool, Move it to Archive after 200 days.
* From Archive, Delete the Blob after 600 days.

**6.** After setting up the Lifecycle!

Trigger an Outlook Email to the QuickCart Vendor Management Team to review the Product for Compliance purpose using the Concept of Logic App.

---------------------------------------------------------------------------------------------------------

**TASK-5: IMPLEMENTING CONTINUOUS INTEGRATION FOR BACKEND**

**Goal**:

* Learn to build CI Pipeline in Azure Devops using YAML.

**1.** Make sure the latest Dotnet Backend codebase is Present in Azure Repos.

**2.** Have the clarity of the Dotnet Build steps.

* Packages are Restored (with the help of Nuget Tool)
* Build Happens (With the help of Build tool)
* Publish happens (With the help of Build Tool)

**3.** Create a Build Pipeline using Yaml.

**4.** The Build Pipeline should be triggered whenever there is a modification in the master branch of the repository.

**5.** Create 1 Pipeline to run on Microsoft-Hosted agent.

**6.** Create a new VM.

**7.** Create a new Agent Pool in Azure Devops.

**8.** Add this VM as an agent by installing necessary Tools like **Git, Dotnet SDK 3.1**

**9.** Create a Build Pipeline to run on **Self-Hosted Agent**.

**TASK-6: IMPLEMENTING CONTINUOUS DEPLOYMENT PIPELINE**

**Goal**:

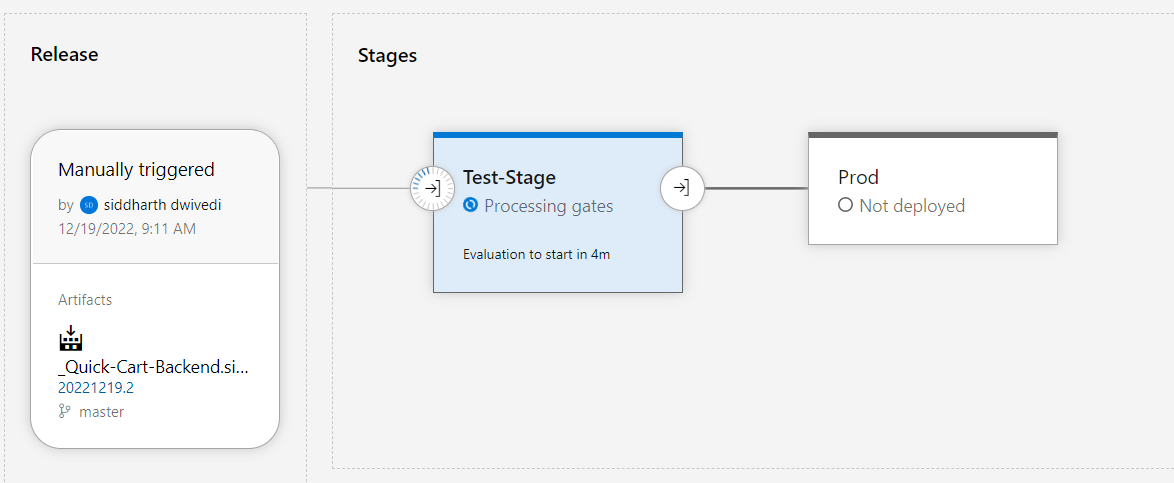
* Learn to build CD mechanism using Classic Interface.
* Understand Approval mechanism + Gates.

**1.** Make sure the Source code is present in the Azure Repos.

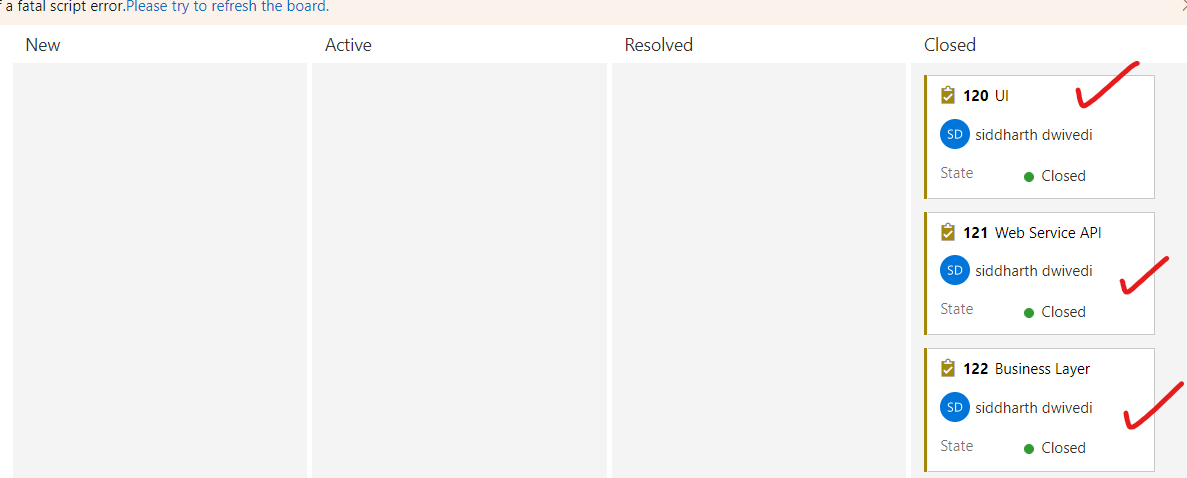
**2.** Make sure the CI Pipeline is building the solution and keeping the Binaries in Artifact.

**3**. Create a new Release Pipeline, which will deploy the changes to Azure App Service.

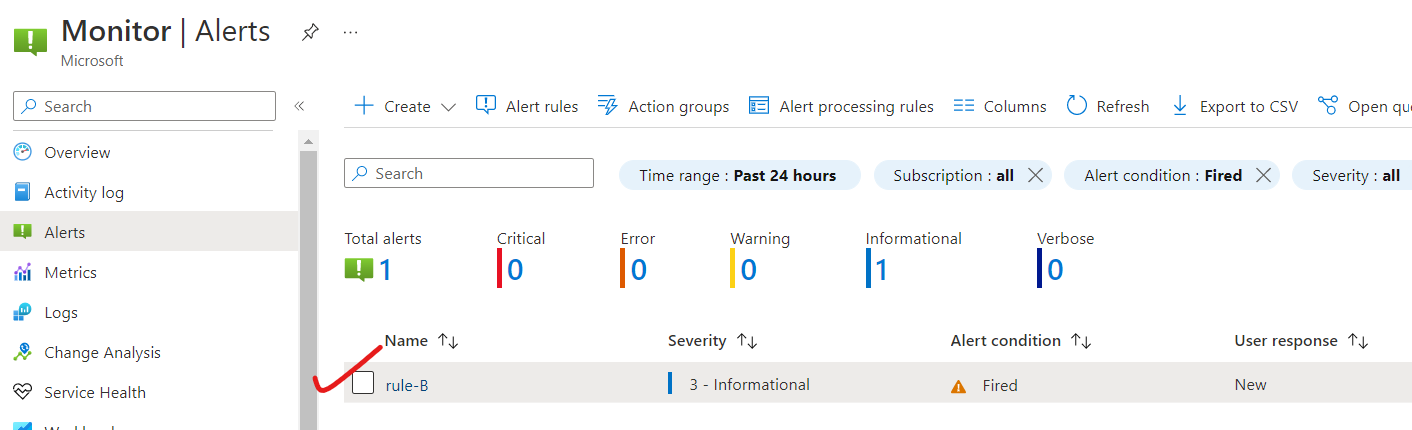
**4**. There should be 2 stages, Test and Prod.



**5**. Test stage should have a Pre-deployment Condition that All the Tasks should be in closed state from Azure Boards.



**6**. Test Stage should also have a Post-deployment Condition (GATE) that if there is any Azure Monitor Alerts raised on the Test-App Service (Average Working Memory Set > 1 Byte) then the further deployment should stop.



**7**. Prod Stage should have an Approval (Pre-deployment condition) in place which would always ask one person for the manual approval before the Prod deployment.

Keep the timeout to be 5 mins.

**8**. Now reference a different DB to this App Service for the Production Stage.

Data Source=quickcart-server.database.windows.net;Initial Catalog=test-dB;user id=demouser; password=Siddharth@1234

Make sure that the DB Value is coming from the Keyvault.

**TASK-7: IMPLEMENTING MESSAGES USING SERVICE BUS**

**Goal**:

* Create Service Bus Queue.
* Flow the messages from 1 Microservice to Another.

1. Clone the Frontend Project from <https://siddharthdwivedi318@dev.azure.com/siddharthdwivedi318/Experiential%20Learning/_git/Quick-Cart-FrontEnd>

2. Execute Orders Table

create table orders

(

orderid int identity,

custEmail varchar(50) references Customers(emailID),

prodID int,

prodCost int,

orderdate dateTime default getDate()

)

and Execute your own Procedure in the Database with this Format

create proc usp\_AddOrder\_<Your\_Name>

as

begin

begin try

insert into orders values ('Your\_email',1,2000,default)

return 1

end try

begin catch

return 0

end catch

end

Make sure in the customer’s table your emailID is present.

3. Create Payment Microservice Using Azure Function which would Process the Payment and then send the Payment Details to the Service Bus Queue.

Write the code yourself!

4. Create Order Microservice using Azure Function which would get triggered whenever there is a new message in the Queue. This service would insert a new row in the Order table.

**Reference Code:**

[FunctionName("order")]

public static async Task Run([ServiceBusTrigger("ffff", Connection = "con")]string myQueueItem, ILogger log)

{

dynamic data = JsonConvert.DeserializeObject(myQueueItem);

string prodCost = data.prodCost;

string prodID = data.prodId;

int result = 0;

//SQL connection

SqlConnection conObj = new SqlConnection("");

//command

SqlCommand cmdObj = new SqlCommand("usp\_AddOrder", conObj);

cmdObj.CommandType = CommandType.StoredProcedure;

try

{

SqlParameter prmReturnValue = new SqlParameter();

prmReturnValue.Direction = ParameterDirection.ReturnValue;

cmdObj.Parameters.Add(prmReturnValue);

conObj.Open();

cmdObj.ExecuteNonQuery();

int res = Convert.ToInt32(prmReturnValue.Value);

if (res == 1)

result = 1;//it means added

else

result = 0;//error

}

catch (Exception e)

{

result = -1;

}

finally

{

conObj.Close();

}

log.LogInformation(""+result);

}