AWS S3 BUCKET WITH ASP.NET CORE 8.0 TO STORE TRANSACTION DATA WITH SPECIFIC FOLDERS

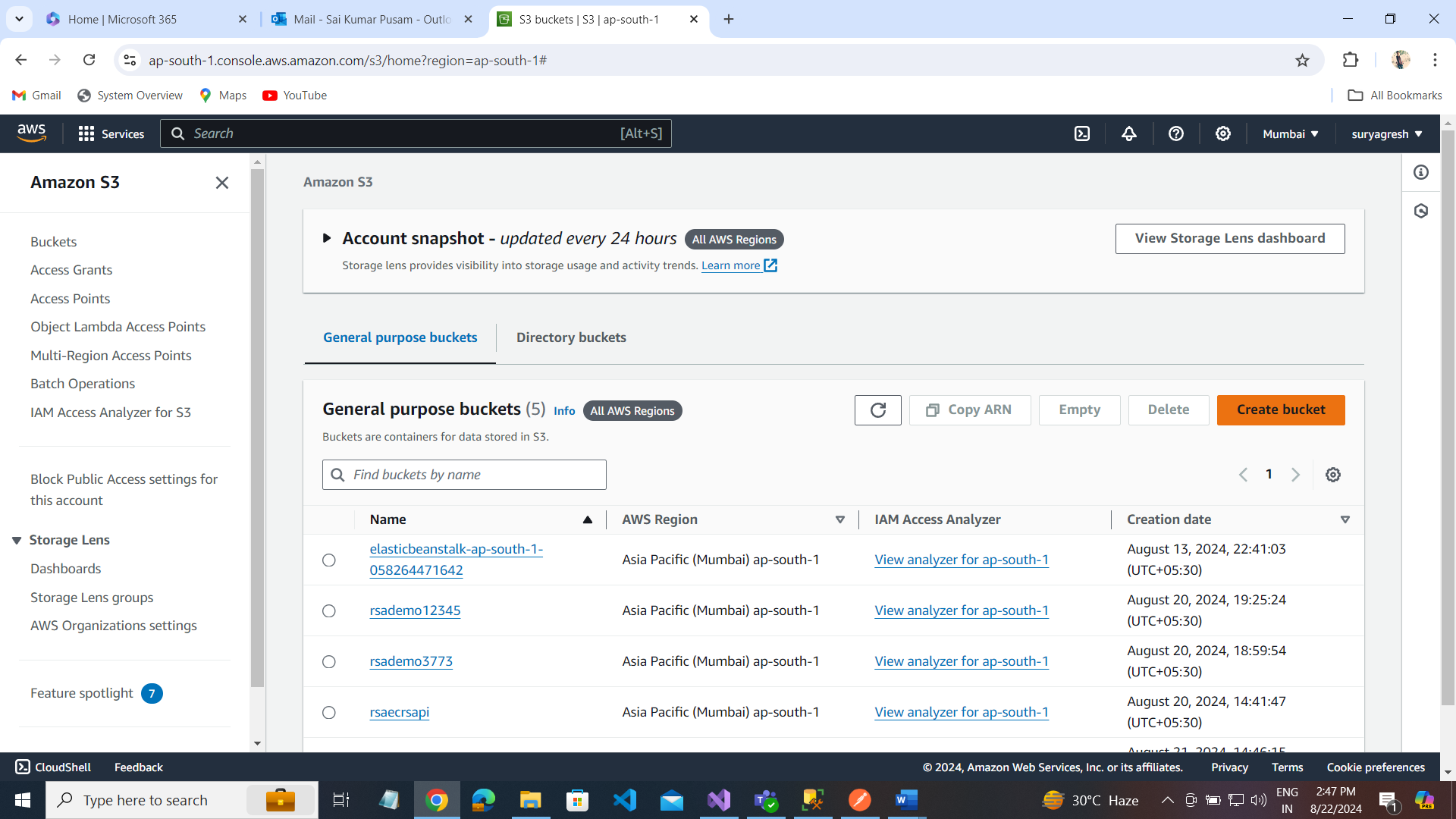
Pre-requisites:

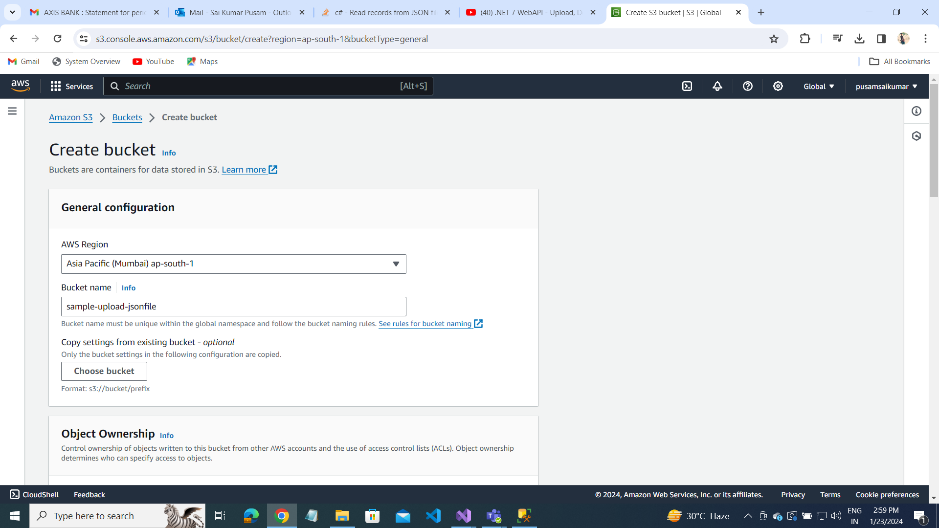
User accounts:

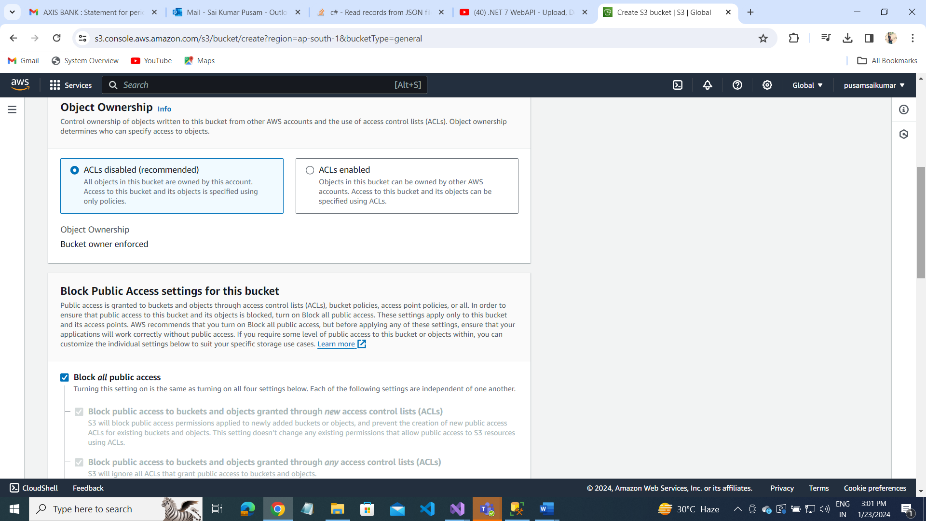
Username: surya.nupa@gmail.com

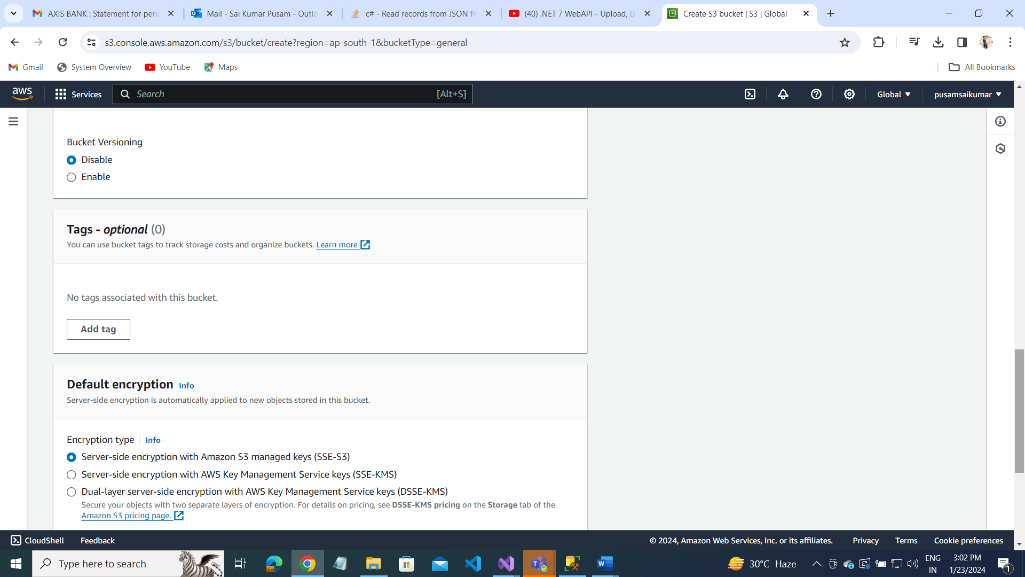
Password: (E9j:f.rfwBV67j

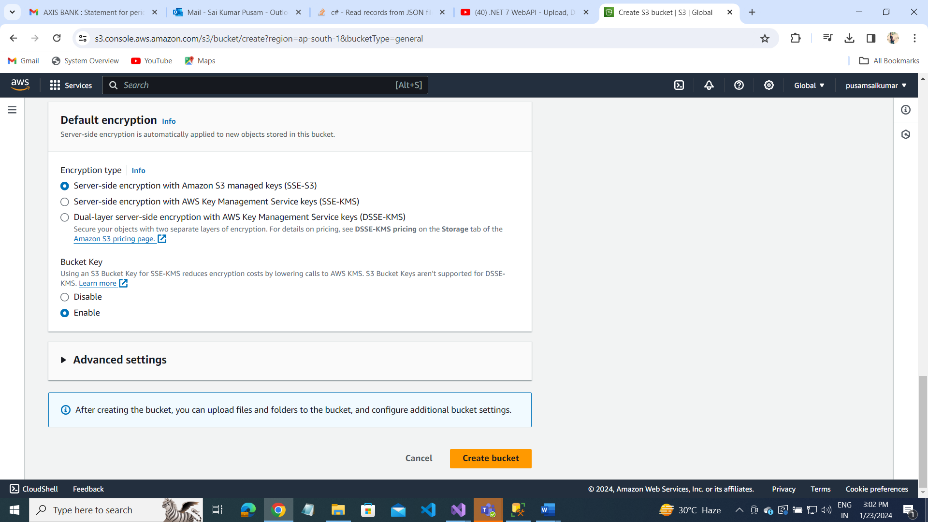
Go to S3 : create bucket

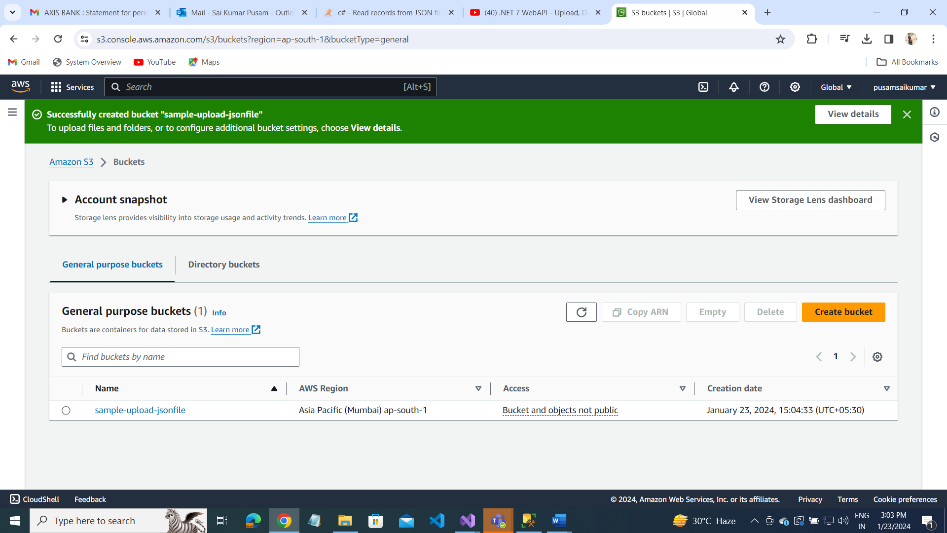


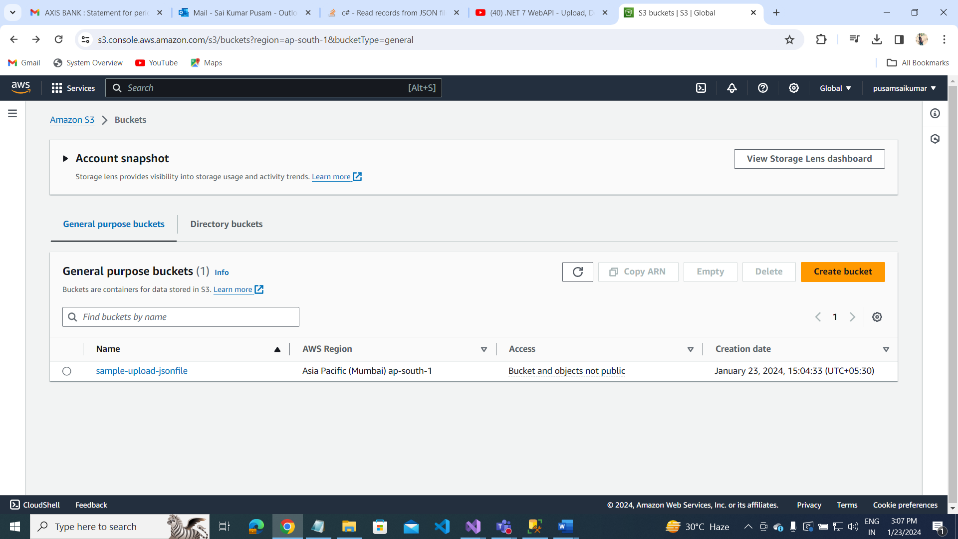








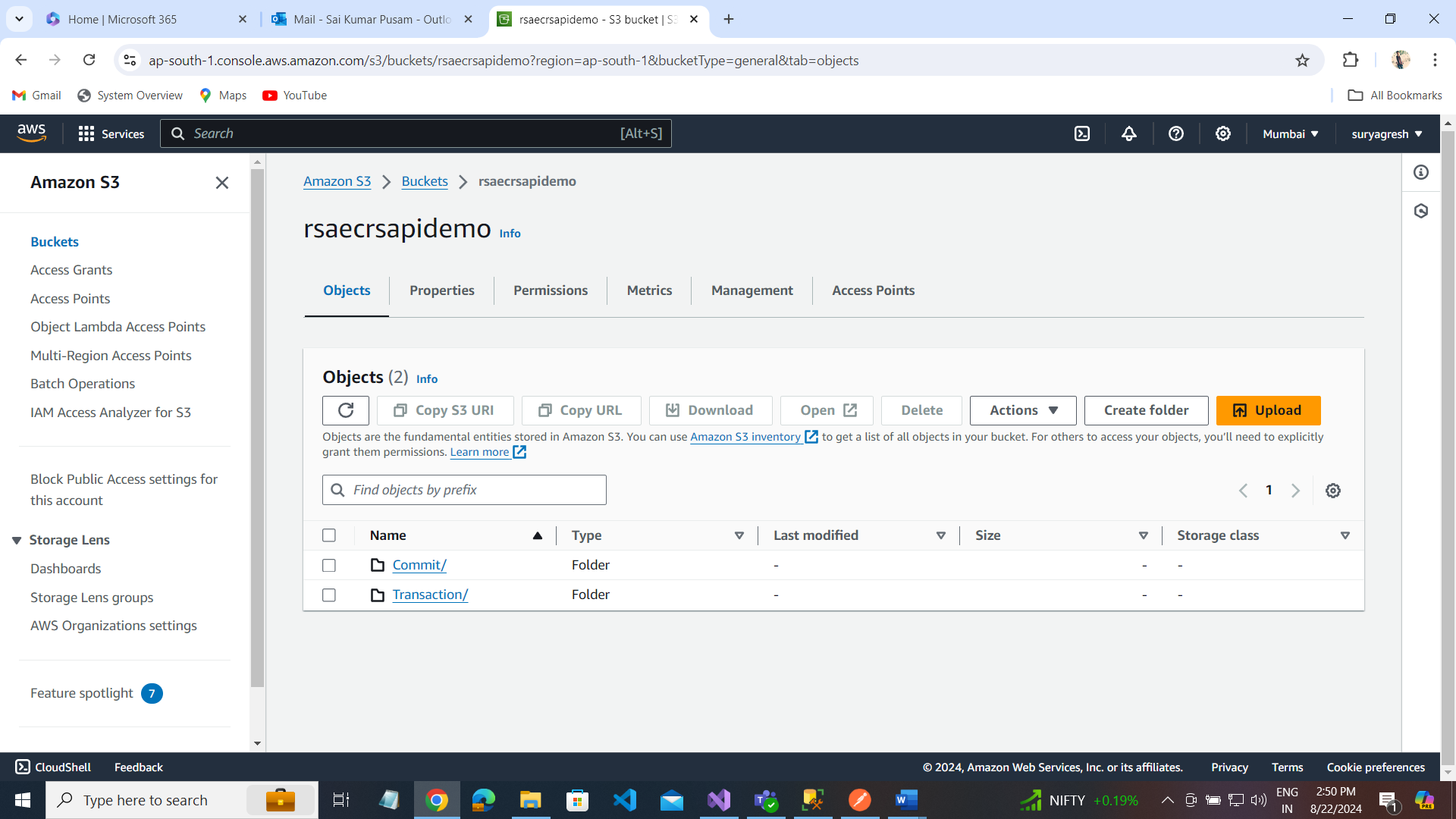


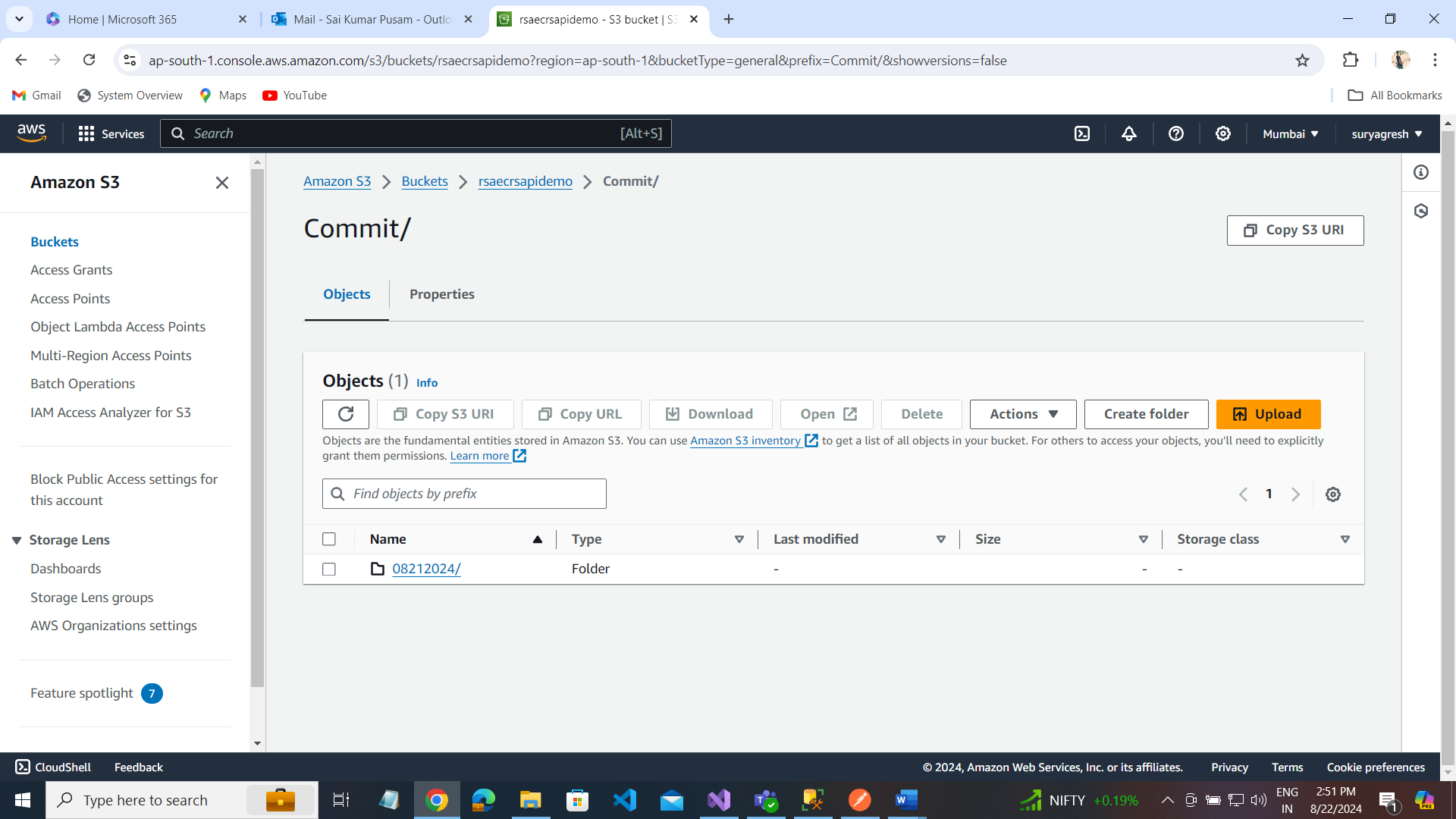


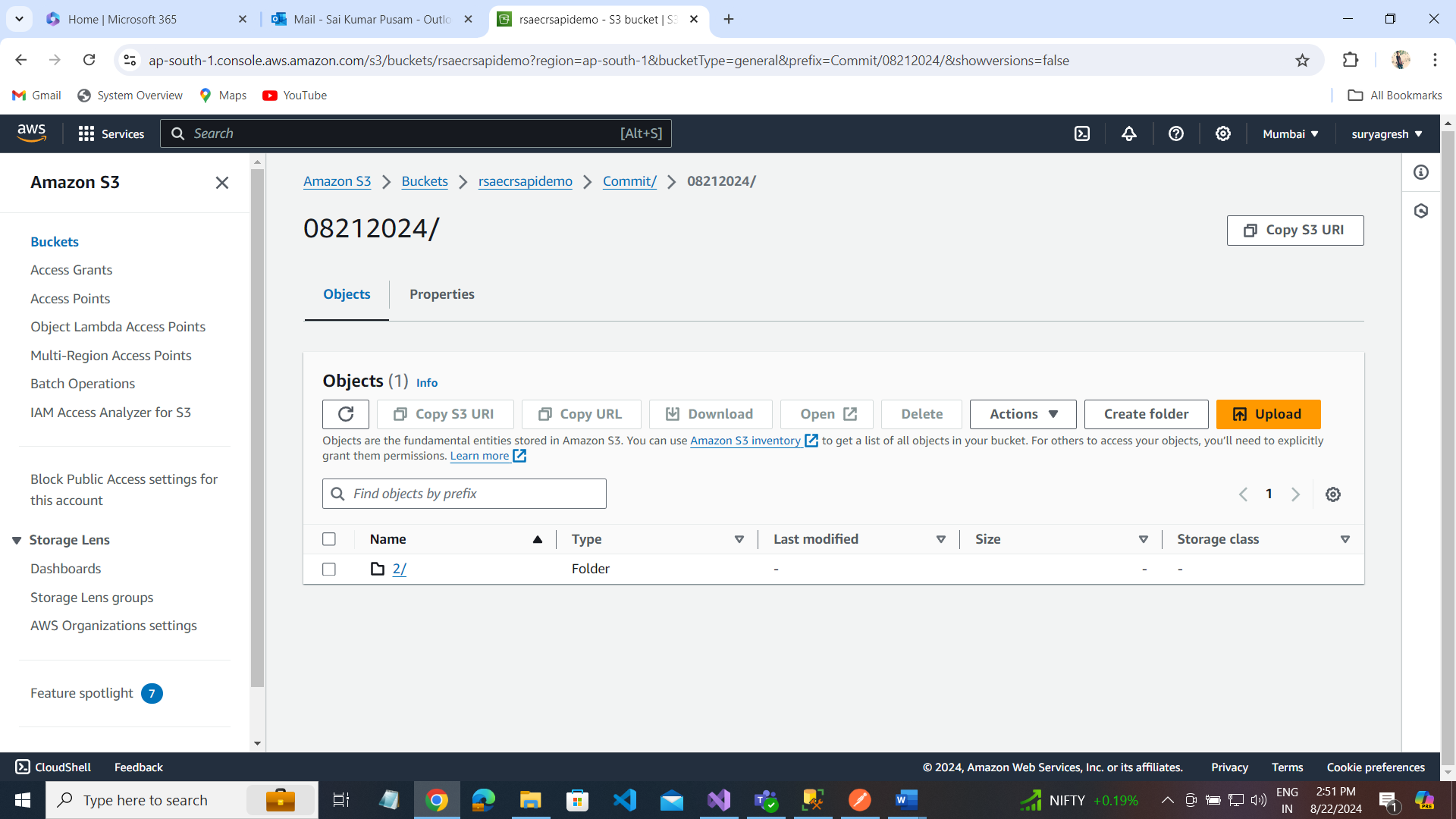
Create folder for buckets like following:

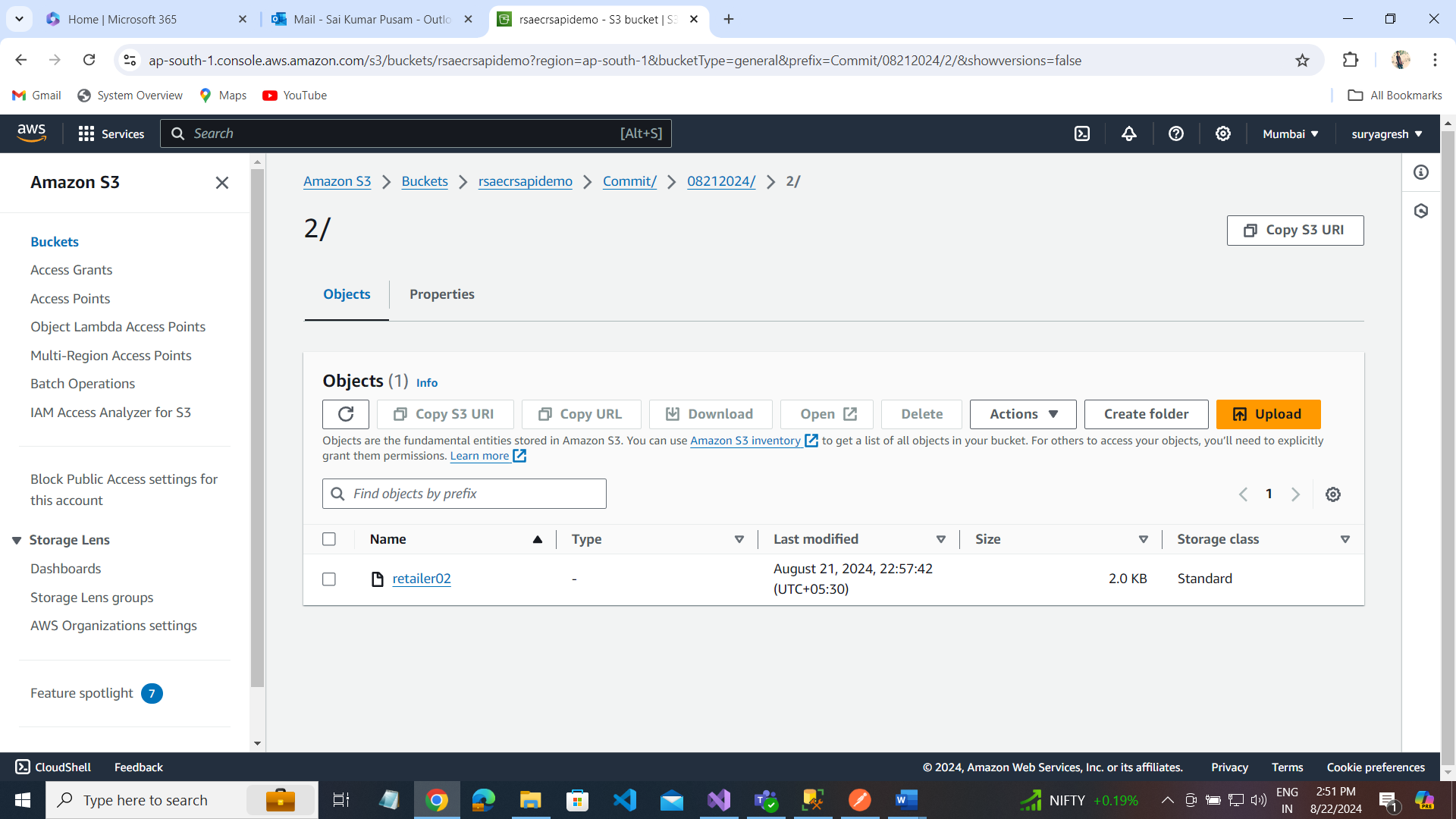
rsaecrsapidemo ------ BucketName

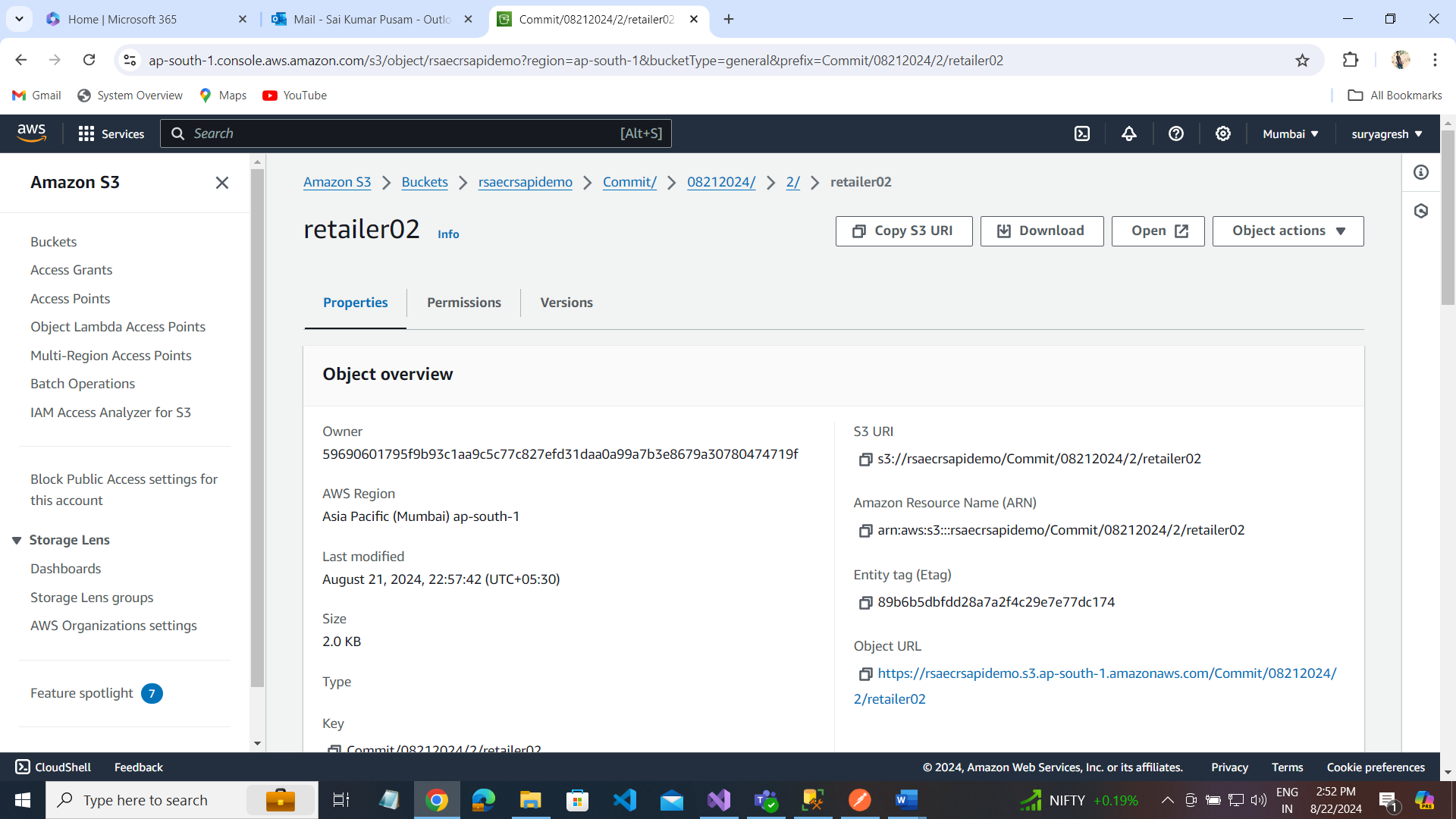
Commit/08212024/2/retailer02 ----- Folder path (commit/Mmddyyyy/storeId/fileName)











<https://rsaecrsapidemo.s3.ap-south-1.amazonaws.com/Commit/08212024/2/retailer02>

Go to your Asp.Net Core web api project:

Install packages:

AWSSDK.S3

AWSSDK.Extensions.NETCore.Setup

StartUp.cs:

public class StartUp

{

//private readonly IConfiguration \_configuration;

public IConfiguration configRoot { get; set; }

// constructor

public StartUp(IConfiguration configuration)

{

configRoot = configuration;

}

// configure services

public void ConfigureServices(IServiceCollection services

{  
 // connect aws s3 bucket

services.AddDefaultAWSOptions(configRoot.GetAWSOptions());

services.AddAWSService<IAmazonS3>();   
 }

}

appsettings.json:

"AWS": {

"Profile": "default",

"Region": "ap-south-1"

}

S3BucketHelpers.cs:

using Amazon.S3;

using Amazon.S3.Model;

namespace RSAECRSAPI.Models

{

public class S3BucketHelpers

{

private readonly IAmazonS3 \_amazonS3;

public S3BucketHelpers(IAmazonS3 amazonS3)

{

\_amazonS3 = amazonS3;

}

public async Task<string> GetS3BucketObjectDetails(string bucketName, string key)

{

var response = await \_amazonS3.GetObjectAsync(bucketName, key);

using (var reader = new StreamReader(response.ResponseStream)) {

return await reader.ReadToEndAsync();

}

}

public async Task<BucketResponse> SaveS3BucketObject(string bucketName, string key, string jsonData)

{

BucketResponse bucketResponse = new BucketResponse();

try

{

var putObjectRequest = new PutObjectRequest

{

BucketName = bucketName,

Key = key,

ContentBody = jsonData,

ContentType = "application/json",

};

await \_amazonS3.PutObjectAsync(putObjectRequest);

bucketResponse.ErrorCode = "200";

bucketResponse.ErrorDesc = "Successfully saved bucket details";

bucketResponse.Status = "Success";

}

catch (Exception ex) {

bucketResponse.ErrorCode = "500";

bucketResponse.ErrorDesc = ex.Message;

bucketResponse.Status = "Internel server error";

}

return bucketResponse;

}

public async Task<BucketResponse> DeleteBucketObject(string bucketName,string key)

{

BucketResponse bucketResponse = new BucketResponse();

try

{

var deleteObjectRequest = new DeleteObjectRequest

{

BucketName = bucketName,

Key = key

};

await \_amazonS3.DeleteObjectAsync(deleteObjectRequest);

bucketResponse.ErrorCode = "200";

bucketResponse.ErrorDesc = "Deleted bucket object successfully.";

bucketResponse.Status = "Success";

}

catch (Exception ex) {

bucketResponse.ErrorCode = "500";

bucketResponse.ErrorDesc = ex.Message;

bucketResponse.Status = "Internel server error";

}

return bucketResponse;

}

public async Task<bool> ValidateKeyPath(string bucketName,string folderName, string transactionDate,int storeId,string fileName)

{

var request = new ListObjectsV2Request

{

BucketName = bucketName,

Prefix = "",

};

var s3BucketObjects = await \_amazonS3.ListObjectsV2Async(request);

var validatePath = s3BucketObjects.S3Objects.FirstOrDefault(x =>

{

return x.Key.Split("/")[0] == folderName &&

x.Key.Split("/")[1].Length > 1 && x.Key.Split("/")[1] == transactionDate &&

x.Key.Split("/")[2].Length > 2 && x.Key.Split("/")[2] == (storeId).ToString() &&

x.Key.Split("/")[3].Length > 3 && x.Key.Split("/")[3] == fileName;

});

if (validatePath != null) {

return true;

}

return false;

}

public async Task<string> ValidateKeyPathAsync(string bucketName, string folderName, string transactionDate, int storeId, string fileName)

{

string key = string.Empty;

var request = new ListObjectsV2Request

{

BucketName = bucketName,

Prefix = ""

};

var result = await \_amazonS3.ListObjectsV2Async(request);

var validPath = result.S3Objects.FirstOrDefault(x =>

{

return x.Key.Split("/")[0] == folderName &&

x.Key.Split("/")[1] == transactionDate &&

x.Key.Split("/")[2] == storeId.ToString() &&

x.Key.Split("/")[3] == fileName;

});

if (validPath != null)

{

key = validPath.Key;

}

return key;

}

public async Task<bool> ExistFolderPath(string bucketName, string folderPath)

{

var request = new ListObjectsV2Request

{

BucketName = bucketName,

Prefix = folderPath,

};

var result = await \_amazonS3.ListObjectsV2Async(request);

if (result.S3Objects.Count > 0 || result.CommonPrefixes.Count > 0) {

return true;

}

return false;

}

}

}

S3Models:

public class S3Models

{

}

public class S3Model

{

public string Name { get; set; }

public string PresignedUrl { get; set; }

}

public class BucketResponse

{

public string ErrorCode { get; set; }

public string ErrorDesc { get; set; }

public string Status { get; set; }

}

S3Repo.cs:

using Amazon.Lambda.RuntimeSupport.Helpers;

using Amazon.S3;

using Amazon.S3.Model;

using Microsoft.AspNetCore.DataProtection;

using Newtonsoft.Json;

using RSAECRSAPI.Models;

using System.Transactions;

namespace RSAECRSAPI.ECRSDAL

{

public class S3Repo : IS3Repo

{

private readonly IAmazonS3 \_amazonS3;

private readonly List<AppConfigurations> \_appConfigurations;

private readonly S3BucketHelpers \_s3BucketHelpers;

public S3Repo(

IAmazonS3 amazonS3,

List<AppConfigurations> appConfigurations,

S3BucketHelpers s3BucketHelpers

)

{

\_amazonS3 = amazonS3;

\_appConfigurations = appConfigurations;

\_s3BucketHelpers = s3BucketHelpers;

}

public async Task<BucketResponse> CreateBucket(string bucketName)

{

BucketResponse response = new BucketResponse();

try

{

//var bucketExists = await Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(\_amazonS3, bucketName);

var listOfBuckets = await \_amazonS3.ListBucketsAsync();

var exist = listOfBuckets.Buckets.Any(b => b.BucketName == bucketName);

if (exist) {

response.ErrorCode = "400";

response.ErrorDesc = $"Bucket {bucketName} already existed.";

response.Status = "Failed";

return response;

}

var putBucketRequest = new PutBucketRequest

{

BucketName = bucketName,

UseClientRegion = true,

};

await \_amazonS3.PutBucketAsync(putBucketRequest);

response.ErrorCode = "200";

response.ErrorDesc = $"Bucket {bucketName} created successfully.";

response.Status = "Success";

}

catch (Exception ex) {

response.ErrorCode = "500";

response.ErrorDesc = ex.Message;

response.Status = "Internel server error";

}

return response;

}

public async Task<BucketResponse> DeleteBucket(string bucketName)

{

BucketResponse response = new BucketResponse();

try

{

await \_amazonS3.DeleteBucketAsync(bucketName);

response.ErrorCode = "200";

response.ErrorDesc = $"Bucket {bucketName} deleted successfully.";

response.Status = "Success";

}

catch (Exception ex) {

response.ErrorCode = "500";

response.ErrorDesc = "";

response.Status = "Internel server error";

}

return response;

}

}

}

public async Task<UpdateTransactionResponse> transaction(string site, string customer, string transaction, UpdateTransactionRequest TranRequest)

{

UpdateTransactionResponse response = new UpdateTransactionResponse();

try

{

var appConfigurationData = \_appConfigurations.ToArray();

AppConfigurations appConfigurations = new AppConfigurations();

// appConfigurations = appConfigurationData.FirstOrDefault(x => x.SharedKey == TranRequest.sharedKey && x.SecretKey == TranRequest.Secret);

var jsonData = \_appConfigurations.ToArray().FirstOrDefault(x => x.SharedKey == TranRequest.sharedKey && x.SecretKey == TranRequest.Secret);

TransactionData transactionData = null;

if (jsonData != null)

{

transactionData = new TransactionData

{

SharedKey = jsonData.SharedKey,

SecretKey = jsonData.SecretKey,

Retailer = jsonData.RetailerName,

Customer = TranRequest.Customer,

StoreId = 5,

TransactionId = TranRequest.Transaction,

Cashier = TranRequest.Cashier,

Terminal = TranRequest.Terminal,

TransactionDate = DateTime.Now,

TransactionTime = new TimeSpan(),

Items = new List<TransactionItem>()

{

new TransactionItem

{

Id = 1,

IdType = "TypeB",

UPC = "098765432109",

Amount = 5,

StandardPrice = 6,

AmountPaid = 6,

Qty = 2,

QtyType = "Piece",

ItemWeight = 0,

ItemType = "Grocery",

DeptId = 102,

FamilyCode1 = "FC3",

FamilyCode2 = "FC4",

SaleType = "Retail",

CoPrefix = "PrefixB",

LineID = "Line2",

IsDiscountable = false,

Weightunit = "kg"

}

},

TransactionTotalAmount = (int)TranRequest.SubTotal,

TransactionTaxAmount = (int)TranRequest.TaxTotal,

TransactionGrossTotal = (int)TranRequest.GrossTotal,

PhoneNumber = TranRequest.PhoneNumber,

Tenders = new List<TransactionTender>()

{

new TransactionTender

{

Type = "CreditCard",

Amount = 0,

}

},

TenderType = "CreditCard",

PosType = jsonData.POSType,

DBInstanceName = jsonData.DBInstanceName,

DBName = jsonData.DBName,

IsMemberTransaction = false,

Coupons = new List<AppliedCoupon>()

{

new AppliedCoupon

{

PromotionId = "promo123",

IdType = "Coupon",

Title = "10% Off",

Items = new List<CouponLineItem>()

{

new CouponLineItem

{

LineId = 1,

DiscountAmount = 2,

Quantity = 1

}

},

TotalDiscount = 2,

CouponTypeId = 1,

AppliedCount = 1,

CouponType = "Discount",

CouponSource = "POS",

QualifiedLineIds = "1",

QualifiedUPCs = "123456789012",

IsAppliedAtPOS = true

}

},

Promotions = new List<AppliedPromotion>()

{

new AppliedPromotion

{

PromotionId = "promo123",

IdType = "Promotion",

Title = "10% Off",

Items = new List<CouponLineItem>()

{

new CouponLineItem

{

LineId = 1,

DiscountAmount = 2,

Quantity = 1

}

},

TotalDiscount = 2,

CouponTypeId = 1,

AppliedCount = 1,

CouponType = "Discount",

CouponSource = "POS",

QualifiedLineIds = "1",

QualifiedUPCs = "123456789012",

IsAppliedAtPOS = true

}

},

//CommitRequest = null,

CommitRequest = new CommitData(),

CloudRetailerName = "rsaecrsapidemo",

CloudFileName = "retailer05",

CloudFolderName = "Transaction"

};

}

var bucketName = transactionData?.CloudRetailerName;

var listOfBuckets = await \_amazonS3.ListBucketsAsync();

var existBucketName = listOfBuckets.Buckets.Any(b => b.BucketName == bucketName);

// var exist = listOfBuckets.Buckets.Select(b => b.BucketName == bucketName);

// var createBucket = await CreateBucket(bucketName);

if (existBucketName && bucketName != null && transactionData != null)

{

string json = JsonConvert.SerializeObject(transactionData);

var storeId = transactionData.StoreId;

var transactionDate = transactionData.TransactionDate.ToString("MMddyyyy");

var folderName = transactionData.CloudFolderName;

var fileName = transactionData.CloudFileName;

//string key = $"Transaction/{dateFolder}/{storeId}/filename.json";

string key = $"{folderName}/{transactionDate}/{storeId}/{fileName}";

var putObjectRequest = new PutObjectRequest

{

BucketName = bucketName,

Key = key,

ContentBody = json,

ContentType = "application/json",

};

var validPath = await \_s3BucketHelpers.ValidateKeyPathAsync(bucketName, folderName, transactionDate, storeId, fileName);

if (!string.IsNullOrEmpty(validPath))

{

response.ErrorCode = "400";

response.ErrorDesc = "Transaction data has been already existed.";

response.Status = "Failed";

return response;

}

var saveS3Obj = await \_s3BucketHelpers.SaveS3BucketObject(bucketName, key, json);

// PutObjectResponse putObjectResponse = await \_amazonS3.PutObjectAsync(putObjectRequest);

if (saveS3Obj.Status == "Success")

{

response.ErrorCode = saveS3Obj.ErrorCode;

response.ErrorDesc = saveS3Obj.ErrorDesc;

response.Status = saveS3Obj.Status;

return response;

}

response.ErrorCode = "400";

response.ErrorDesc = saveS3Obj.ErrorDesc;

response.Status = "Failed";

return response;

}

response.ErrorCode = "400";

response.ErrorDesc = "";

response.Status = "Failed";

}

catch (Exception ex)

{

response.ErrorCode = "500";

response.ErrorDesc = ex.Message;

response.Status = "Internel server error";

}

return response;

}

public async Task<CommitResponse> commit(string site, string customer, string transaction, string sharedkey, string secret, CommitTransactionRequest commitTranRequest)

{

CommitResponse commitResponse = new CommitResponse();

try

{

var jsonData = \_appConfigurations?.FirstOrDefault(x => x.SharedKey == commitTranRequest.SharedKey && x.SecretKey == commitTranRequest.Secret);

var bucketName = "rsaecrsapidemo";

var folderName = "Transaction";

var storeId = 5;

var transactionDate = DateTime.Now.ToString("MMddyyyy");

var fileName = "retailer5";

var key = $"{folderName}/{transactionDate}/{storeId}/{fileName}";

// validate BucketName:

// var existBucket = await \_amazonS3.DoesS3BucketExistAsync(bucketName);

//var listbuckets = await \_amazonS3.ListBucketsAsync();

//var existBucket = listbuckets.Buckets.Any(b => b.BucketName == bucketName);

var existBucket = await Amazon.S3.Util.AmazonS3Util.DoesS3BucketExistV2Async(\_amazonS3, bucketName);

var transactiondata = new TransactionData();

if (existBucket)

{

var validPath = await \_s3BucketHelpers.ValidateKeyPathAsync(bucketName, folderName, transactionDate, storeId, fileName);

if (!string.IsNullOrEmpty(validPath))

{

//string key = Transaction/08212024/2/retailer02;

var getS3TransactionData = await \_s3BucketHelpers.GetS3BucketObjectDetails(bucketName, validPath);

transactiondata = JsonConvert.DeserializeObject<TransactionData>(getS3TransactionData);

if (transactiondata != null)

{

var commitData = new CommitData

{

SharedKey = transactiondata.SharedKey,

SecretKey = transactiondata.SecretKey,

Customer = transactiondata.Customer,

StoreId = transactiondata.StoreId,

TransactionId = transactiondata.TransactionId,

Cashier = transactiondata.Cashier,

Terminal = transactiondata.Terminal,

DateTime = transactiondata.TransactionDate,

Coupons = new List<string> { "COUPON123", "COUPON456" },

Tenders = new List<TransactionTender>

{

new TransactionTender {Type = "Credit",Amount=90},

new TransactionTender {Type = "Cash",Amount=90},

},

TenderType = "Mixed",

IsMemberTransaction = transactiondata.IsMemberTransaction,

CloudRetailerName = transactiondata.CloudRetailerName,

CloudFileName = transactiondata.CloudFileName,

CloudFolderName = transactiondata.CloudFolderName,

};

transactiondata.Coupons = transactiondata.Coupons;

transactiondata.Promotions =transactiondata.Promotions;

transactiondata.CommitRequest = commitData;

}

}

else

{

if (jsonData != null )

{

var commitData = new CommitData

{

SharedKey = jsonData.SharedKey,

SecretKey = jsonData.SecretKey,

Customer = commitTranRequest.Customer,

StoreId = 4,

TransactionId = commitTranRequest.Transaction,

Cashier = commitTranRequest.Cashier,

Terminal = commitTranRequest.Terminal,

DateTime = DateTime.Now,

Coupons = commitTranRequest.coupons,

Tenders = commitTranRequest.tenders.Select(x =>

{

return new TransactionTender { Type = x.Type, Amount = (decimal)x.Amount };

}).ToList(),

TenderType = "Mixed",

IsMemberTransaction = true,

CloudRetailerName = "rsaecrsapidemo",

CloudFileName = "retailer4",

CloudFolderName = "Commit"

};

transactiondata = new TransactionData

{

SharedKey = jsonData.SharedKey,

SecretKey = jsonData.SecretKey,

Retailer = jsonData.RetailerName,

Customer = "saikumar",

StoreId = 5,

TransactionId = "349857",

Cashier = 49,

Terminal = 99,

TransactionDate = DateTime.Now,

TransactionTime = new TimeSpan(),

Items = new List<TransactionItem>()

{

new TransactionItem

{

Id = 1,

IdType = "TypeB",

UPC = "098765432109",

Amount = 5,

StandardPrice = 6,

AmountPaid = 6,

Qty = 2,

QtyType = "Piece",

ItemWeight = 0,

ItemType = "Grocery",

DeptId = 102,

FamilyCode1 = "FC3",

FamilyCode2 = "FC4",

SaleType = "Retail",

CoPrefix = "PrefixB",

LineID = "Line2",

IsDiscountable = false,

Weightunit = "kg"

}

},

TransactionTotalAmount = 29,

TransactionTaxAmount = 49,

TransactionGrossTotal = 49,

PhoneNumber = "9959608677",

Tenders = new List<TransactionTender>()

{

new TransactionTender

{

Type = "CreditCard",

Amount = 0,

}

},

TenderType = "CreditCard",

PosType = jsonData.POSType,

DBInstanceName = jsonData.DBInstanceName,

DBName = jsonData.DBName,

IsMemberTransaction = false,

Coupons = new List<AppliedCoupon>()

{

new AppliedCoupon

{

PromotionId = "promo123",

IdType = "Coupon",

Title = "10% Off",

Items = new List<CouponLineItem>()

{

new CouponLineItem

{

LineId = 1,

DiscountAmount = 2,

Quantity = 1

}

},

TotalDiscount = 2,

CouponTypeId = 1,

AppliedCount = 1,

CouponType = "Discount",

CouponSource = "POS",

QualifiedLineIds = "1",

QualifiedUPCs = "123456789012",

IsAppliedAtPOS = true

}

},

Promotions = new List<AppliedPromotion>()

{

new AppliedPromotion

{

PromotionId = "promo123",

IdType = "Promotion",

Title = "10% Off",

Items = new List<CouponLineItem>()

{

new CouponLineItem

{

LineId = 1,

DiscountAmount = 2,

Quantity = 1

}

},

TotalDiscount = 2,

CouponTypeId = 1,

AppliedCount = 1,

CouponType = "Discount",

CouponSource = "POS",

QualifiedLineIds = "1",

QualifiedUPCs = "123456789012",

IsAppliedAtPOS = true

}

},

CommitRequest = commitData,

CloudRetailerName = "rsaecrsapidemo",

CloudFileName = "retailer5",

CloudFolderName = "Commit"

};

}

}

var jsonTransactiondata = JsonConvert.SerializeObject(transactiondata);

var BucketName = transactiondata?.CloudRetailerName;

var FileName = transactiondata?.CloudFileName;

// var FolderName = transactiondata.CloudFolderName;

var FolderName = "Commit";

var StoreId = transactiondata?.StoreId;

var TransactionDate = transactiondata?.TransactionDate.ToString("MMddyyyy");

var Key = $"{FolderName}/{TransactionDate}/{StoreId}/{FileName}";

var checkTransaction = await \_s3BucketHelpers.ExistFolderPath(bucketName, key);

var savedCommitData = await \_s3BucketHelpers.SaveS3BucketObject(BucketName, Key, jsonTransactiondata);

if(savedCommitData.Status == "Success")

{

if (checkTransaction)

{

await \_s3BucketHelpers.DeleteBucketObject(BucketName, key);

}

else if (!checkTransaction)

{

commitResponse.ErrorCode = "200";

commitResponse.ErrorDesc = "Transaction not found for this commit.";

commitResponse.Status = "Success";

return commitResponse;

}

commitResponse.ErrorCode = savedCommitData.ErrorCode;

commitResponse.ErrorDesc = savedCommitData.ErrorDesc;

commitResponse.Status = savedCommitData.Status;

return commitResponse;

}

commitResponse.ErrorCode = savedCommitData.ErrorCode;

commitResponse.ErrorDesc = savedCommitData.ErrorDesc;

commitResponse.Status = "Failed.";

return commitResponse;

}

commitResponse.ErrorCode = "400";

commitResponse.ErrorDesc = "Invalid bucket name.";

commitResponse.Status = "Failed";

return commitResponse;

}

catch (Exception ex)

{

commitResponse.ErrorCode = "500";

commitResponse.ErrorDesc = ex.Message;

commitResponse.Status = "Internel server error";

}

return commitResponse;

}

Dal Layer:

public interface IS3Repo

{

Task<BucketResponse> CreateBucket(string bucketName);

Task<BucketResponse> DeleteBucket(string bucketName);

Task<UpdateTransactionResponse> transaction(string site, string customer, string transaction, UpdateTransactionRequest TranRequest);

Task<CommitResponse> commit(string site, string customer, string transaction, string sharedkey, string secret, CommitTransactionRequest commitTranRequest);

}

Business Logic Layer:

IS3Service.cs:

namespace RSAECRSAPI.ECRSBLL

{

public interface IS3Service

{

Task<BucketResponse> CreateBucket(string bucketName);

Task<BucketResponse> DeleteBucket(string bucketName);

Task<UpdateTransactionResponse> transaction(string site, string customer, string transaction, UpdateTransactionRequest TranRequest);

Task<CommitResponse> commit(string site, string customer, string transaction, string sharedkey, string secret, CommitTransactionRequest commitTranRequest);

}

}

S3Service.cs:

using RSAECRSAPI.ECRSDAL;

using RSAECRSAPI.Models;

namespace RSAECRSAPI.ECRSBLL

{

public class S3Service : IS3Service

{

private readonly IS3Repo \_s3Repo;

public S3Service(IS3Repo s3Repo)

{

\_s3Repo = s3Repo;

}

public async Task<CommitResponse> commit(string site, string customer, string transaction, string sharedkey, string secret, CommitTransactionRequest commitTranRequest)

{

var result = await \_s3Repo.commit(site, customer, transaction, sharedkey, secret, commitTranRequest);

return result;

}

public async Task<BucketResponse> CreateBucket(string bucketName)

{

var result = await \_s3Repo.CreateBucket(bucketName);

return result;

}

public async Task<BucketResponse> DeleteBucket(string bucketName)

{

var result = await \_s3Repo.DeleteBucket(bucketName);

return result;

}

public async Task<UpdateTransactionResponse> transaction(string site, string customer, string transaction, UpdateTransactionRequest TranRequest)

{

var result = await \_s3Repo.transaction(site, customer, transaction, TranRequest);

return result;

}

}

}

Save to json data into S3 Buket like below:

======================================== TRANSACTION ============

{

"SharedKey": "rsaecrsapi",

"SecretKey": "rsaecrsapi123",

"Retailer": "",

"Customer": "RSA AMERICA",

"StoreId": 3,

"TransactionId": "4928354",

"Cashier": 27,

"Terminal": 7,

"TransactionDate": "2024-08-22T15:25:43.1009517+05:30",

"TransactionTime": "00:00:00",

"Items": [

{

"Id": 1,

"IdType": "TypeB",

"UPC": "098765432109",

"Amount": 5.0,

"StandardPrice": 6.0,

"AmountPaid": 6.0,

"Qty": 2,

"QtyType": "Piece",

"ItemWeight": 0.0,

"ItemType": "Grocery",

"DeptId": 102,

"FamilyCode1": "FC3",

"FamilyCode2": "FC4",

"SaleType": "Retail",

"CoPrefix": "PrefixB",

"LineID": "Line2",

"IsDiscountable": false,

"Weightunit": "kg"

}

],

"TransactionTotalAmount": 170.0,

"TransactionTaxAmount": 2.0,

"TransactionGrossTotal": 0.0,

"PhoneNumber": "9959608677",

"Tenders": [

{

"Type": "CreditCard",

"Amount": 0.0

}

],

"TenderType": "CreditCard",

"PosType": "ECRS",

"DBInstanceName": "",

"DBName": "",

"IsMemberTransaction": false,

"Coupons": [

{

"PromotionId": "promo123",

"IdType": "Coupon",

"Title": "10% Off",

"Items": [

{

"LineId": 1,

"DiscountAmount": 2.0,

"Quantity": 1

}

],

"TotalDiscount": 2.0,

"CouponTypeId": 1,

"AppliedCount": 1,

"CouponType": "Discount",

"CouponSource": "POS",

"QualifiedLineIds": "1",

"QualifiedUPCs": "123456789012",

"IsAppliedAtPOS": true

}

],

"Promotions": [

{

"PromotionId": "promo123",

"IdType": "Promotion",

"Title": "10% Off",

"Items": [

{

"LineId": 1,

"DiscountAmount": 2.0,

"Quantity": 1

}

],

"TotalDiscount": 2.0,

"CouponTypeId": 1,

"AppliedCount": 1,

"CouponType": "Discount",

"CouponSource": "POS",

"QualifiedLineIds": "1",

"QualifiedUPCs": "123456789012",

"IsAppliedAtPOS": true

}

],

"CommitRequest": { // passing null object

"SharedKey": null,

"SecretKey": null,

"Customer": null,

"StoreId": 0,

"TransactionId": null,

"Cashier": 0,

"Terminal": 0,

"DateTime": "0001-01-01T00:00:00",

"Coupons": null,

"Tenders": null,

"TenderType": null,

"IsMemberTransaction": false,

"CloudRetailerName": null,

"CloudFileName": null,

"CloudFolderName": null

},

"IsPOSDiscountApplied": false,

"CloudRetailerName": "rsaecrsapidemo",

"CloudFileName": "retailer03",

"CloudFolderName": "Transaction"

}

============== COMMIT DATA ===============

{

"SharedKey": "rsaecrsapi",

"SecretKey": "rsaecrsapi123",

"Retailer": "",

"Customer": "RSA AMERICA",

"StoreId": 3,

"TransactionId": "4928354",

"Cashier": 27,

"Terminal": 7,

"TransactionDate": "2024-08-22T15:25:43.1009517+05:30",

"TransactionTime": "00:00:00",

"Items": [

{

"Id": 1,

"IdType": "TypeB",

"UPC": "098765432109",

"Amount": 5.0,

"StandardPrice": 6.0,

"AmountPaid": 6.0,

"Qty": 2,

"QtyType": "Piece",

"ItemWeight": 0.0,

"ItemType": "Grocery",

"DeptId": 102,

"FamilyCode1": "FC3",

"FamilyCode2": "FC4",

"SaleType": "Retail",

"CoPrefix": "PrefixB",

"LineID": "Line2",

"IsDiscountable": false,

"Weightunit": "kg"

}

],

"TransactionTotalAmount": 170.0,

"TransactionTaxAmount": 2.0,

"TransactionGrossTotal": 0.0,

"PhoneNumber": "9959608677",

"Tenders": [

{

"Type": "CreditCard",

"Amount": 0.0

}

],

"TenderType": "CreditCard",

"PosType": "ECRS",

"DBInstanceName": "",

"DBName": "",

"IsMemberTransaction": false,

"Coupons": [

{

"PromotionId": "promo123",

"IdType": "Coupon",

"Title": "10% Off",

"Items": [

{

"LineId": 1,

"DiscountAmount": 2.0,

"Quantity": 1

}

],

"TotalDiscount": 2.0,

"CouponTypeId": 1,

"AppliedCount": 1,

"CouponType": "Discount",

"CouponSource": "POS",

"QualifiedLineIds": "1",

"QualifiedUPCs": "123456789012",

"IsAppliedAtPOS": true

}

],

"Promotions": [

{

"PromotionId": "promo123",

"IdType": "Promotion",

"Title": "10% Off",

"Items": [

{

"LineId": 1,

"DiscountAmount": 2.0,

"Quantity": 1

}

],

"TotalDiscount": 2.0,

"CouponTypeId": 1,

"AppliedCount": 1,

"CouponType": "Discount",

"CouponSource": "POS",

"QualifiedLineIds": "1",

"QualifiedUPCs": "123456789012",

"IsAppliedAtPOS": true

}

],

"CommitRequest": { //update commit data

"SharedKey": "rsaecrsapi",

"SecretKey": "rsaecrsapi123",

"Customer": "RSA AMERICA",

"StoreId": 3,

"TransactionId": "4928354",

"Cashier": 27,

"Terminal": 7,

"DateTime": "2024-08-22T15:25:43.1009517+05:30",

"Coupons": [

"COUPON123",

"COUPON456"

],

"Tenders": [

{

"Type": "Credit",

"Amount": 90.0

},

{

"Type": "Cash",

"Amount": 90.0

}

],

"TenderType": "Mixed",

"IsMemberTransaction": false,

"CloudRetailerName": "rsaecrsapidemo",

"CloudFileName": "retailer03",

"CloudFolderName": "Transaction"

},

"IsPOSDiscountApplied": false,

"CloudRetailerName": "rsaecrsapidemo",

"CloudFileName": "retailer03",

"CloudFolderName": "Transaction"

}