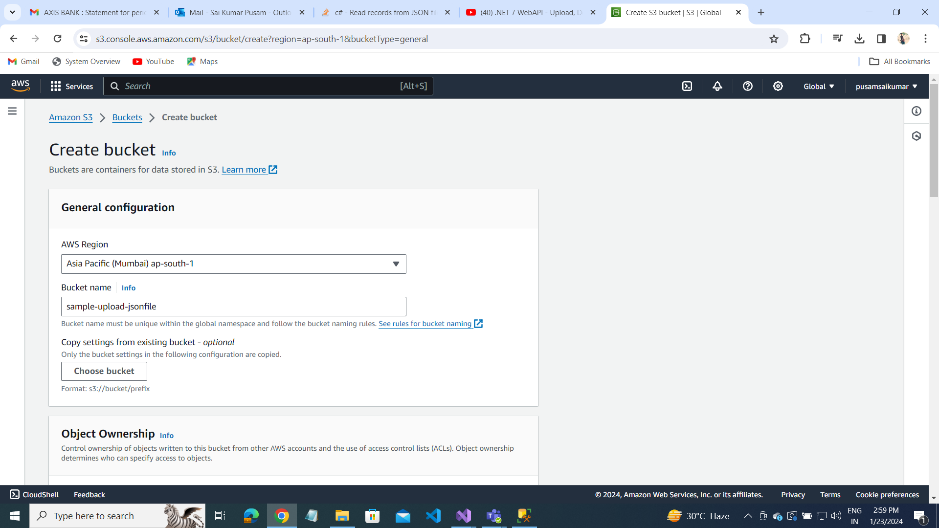
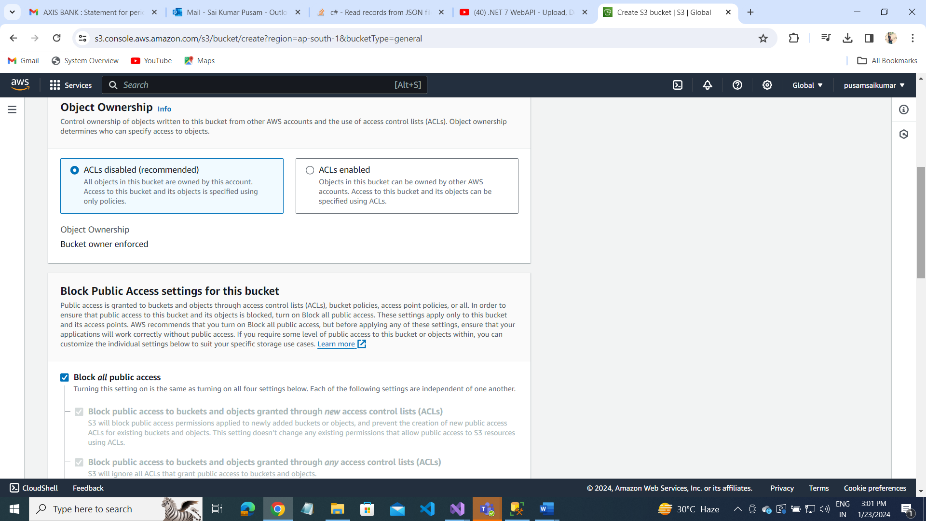
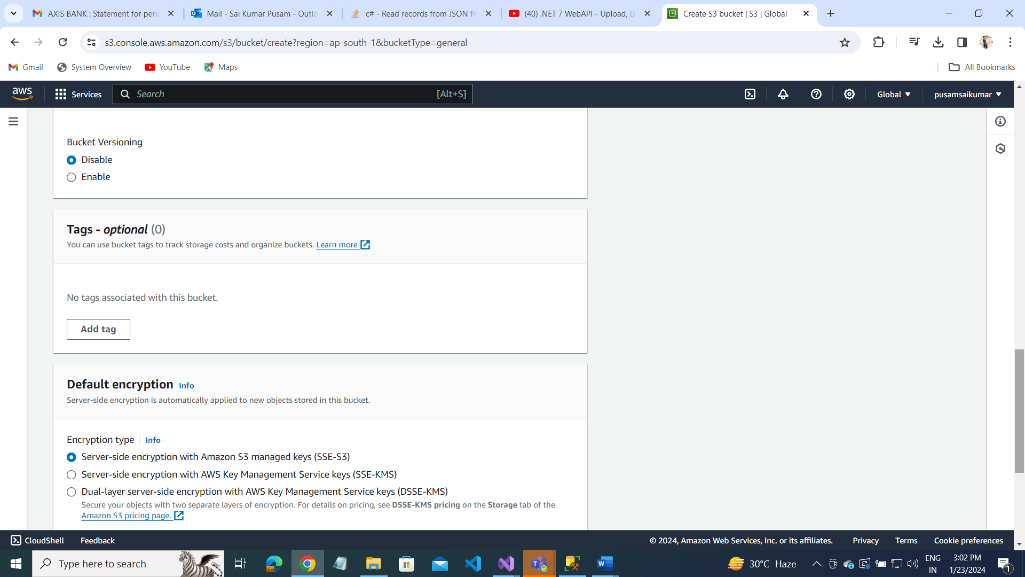
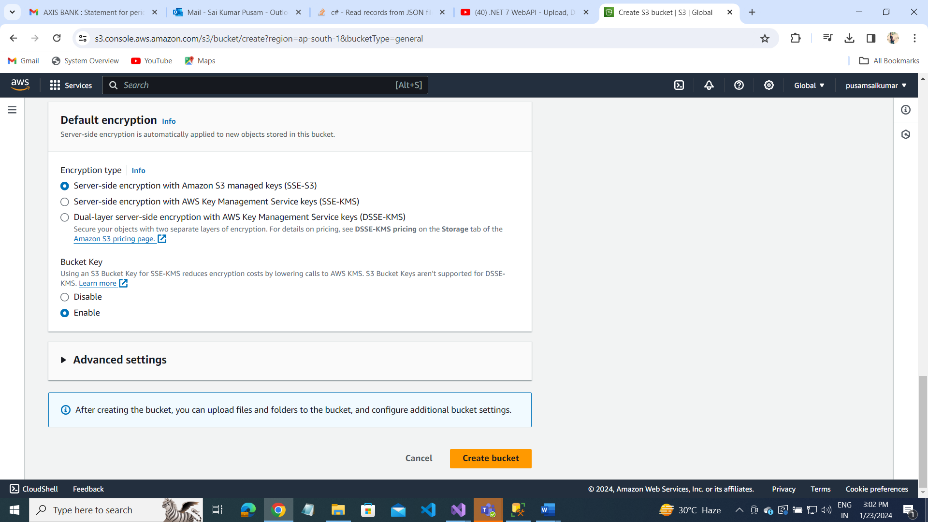
AWS S3 BUCKET

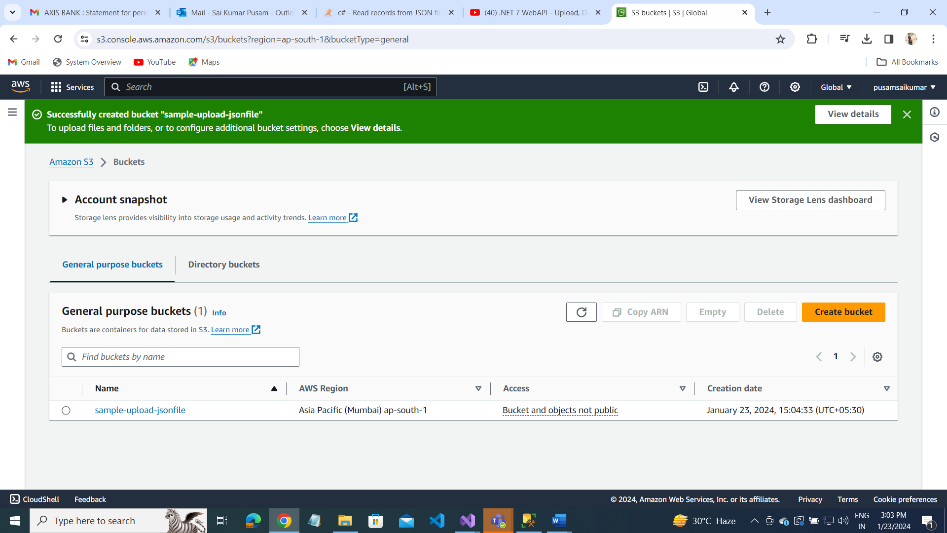
Create aws s3 bucket:

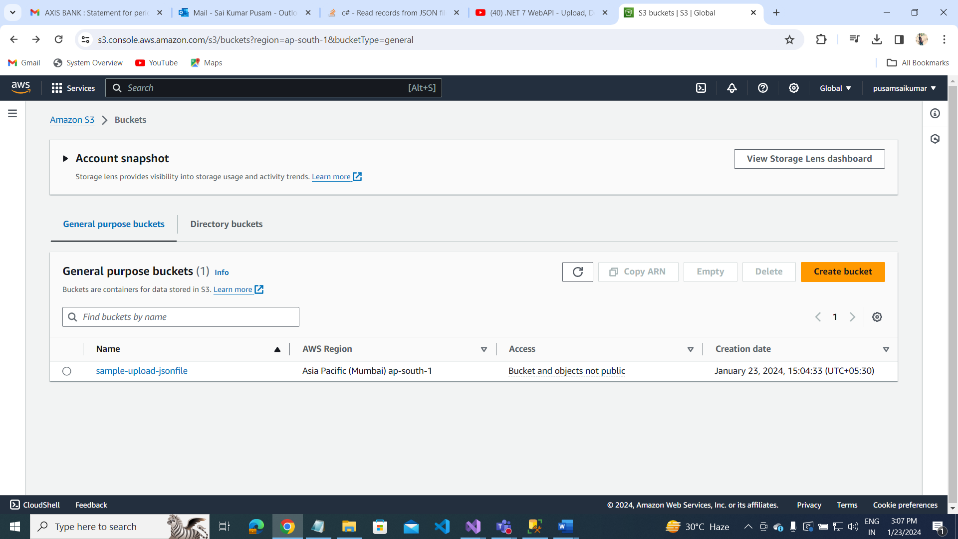


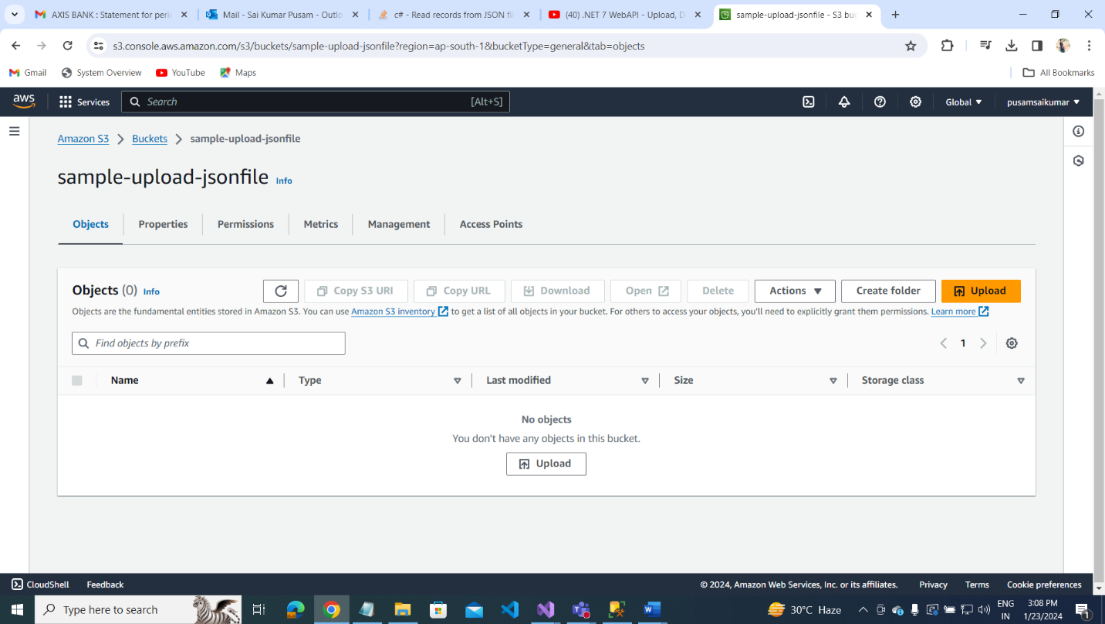






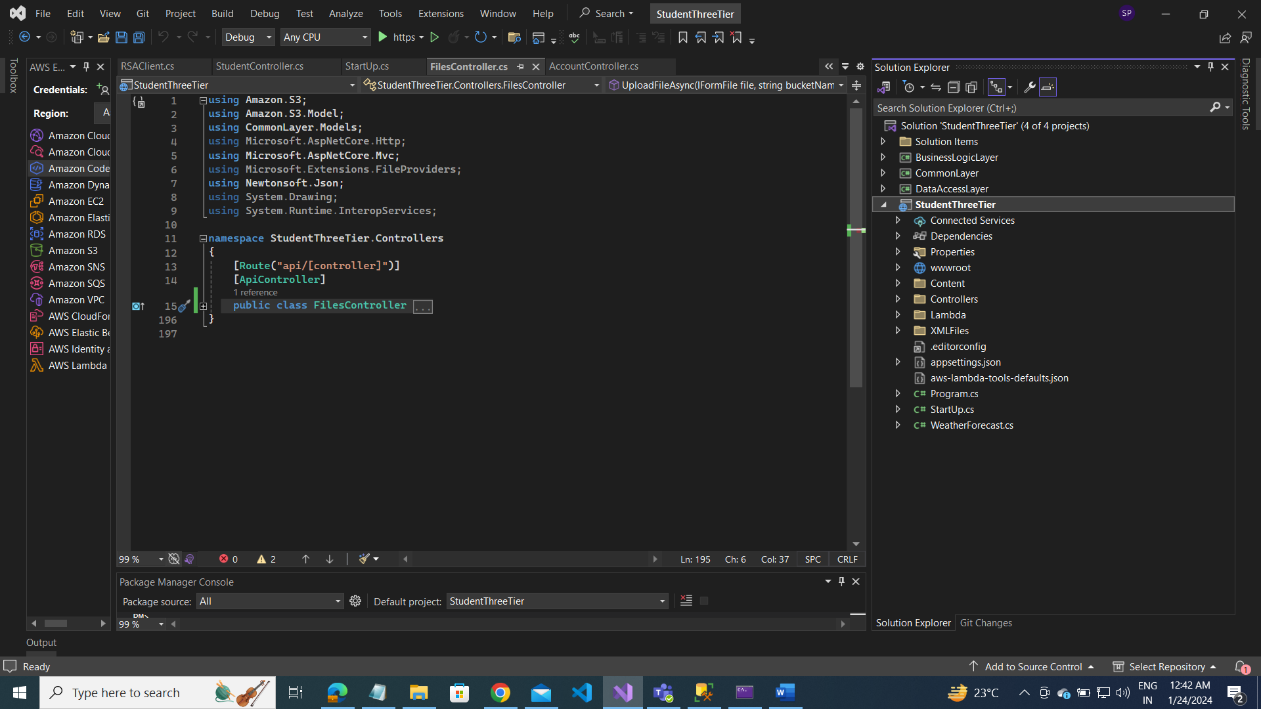






JSON FILE UPLOAD AND READ IT

Create Folder: Right click of Project(StudentThreeTier)– Add-NewFolder – wwwroot(folder name )



S3BucketController

using Amazon.S3;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace StudentThreeTier.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class S3BucketController : ControllerBase

{

private readonly IAmazonS3 \_amazonS3;

public S3BucketController(IAmazonS3 amazonS3)

{

\_amazonS3 = amazonS3;

}

// create bucket:

[HttpPost]

[Route("CreateBucketAsync")]

public async Task<IActionResult> CreateBucketAsync(string bucketName)

{

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

if (bucketExists)

{

return BadRequest($"Bucket {bucketName} alread existed.");

}

await \_amazonS3.PutBucketAsync(bucketName);

return Created("Created" ,$"Bucket {bucketName} created.");

}

[HttpGet]

[Route("GelAllBucketsAsync")]

public async Task<IActionResult> GelAllBucketsAsync()

{

var data = await \_amazonS3.ListBucketsAsync();

var buckets = data.Buckets.Select(b => b.BucketName);

return Ok(buckets);

}

[HttpDelete]

[Route(" DeleteBucketAsyn")]

public async Task<IActionResult> DeleteBucketAsyn(string bucketName)

{

await \_amazonS3.DeleteBucketAsync(bucketName);

return NoContent();

}

}

}

Files Controller

using Amazon.S3;

using Amazon.S3.Model;

using CommonLayer.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.FileProviders;

using Newtonsoft.Json;

using System.Drawing;

using System.Runtime.InteropServices;

namespace StudentThreeTier.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class FilesController : ControllerBase

{

private readonly IAmazonS3 \_amazonS3;

private readonly IWebHostEnvironment \_webHostEnvironment;

public FilesController(

IAmazonS3 amazonS3,

IWebHostEnvironment webHostEnvironment

)

{

\_amazonS3 = amazonS3;

\_webHostEnvironment = webHostEnvironment;

}

[HttpPost]

[Route("UploadFileAsync")]

public async Task<IActionResult> UploadFileAsync(IFormFile file, string bucketName, string? prefix)

{

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

//if (bucketExists)

//{

// return BadRequest($"Bucket {bucketName} alread existed.");

if (!bucketExists)

{

return NotFound($"Bucket {bucketName} does not existed.");

}

var request = new PutObjectRequest()

{

BucketName = bucketName,

Key = string.IsNullOrEmpty(prefix) ? file.FileName : $"{prefix?.TrimEnd('/')}/{file.FileName}",

InputStream = file.OpenReadStream(),

};

request.Metadata.Add("Content-Type", file.ContentType);

await \_amazonS3.PutObjectAsync(request);

return Ok($"File {prefix}/{file.FileName} uploaded to s3 successfully");

}

[HttpGet]

[Route("GetAllFilesAsync")]

public async Task<IActionResult> GetAllFilesAsync(string bucketName, string? prefix)

{

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

if (!bucketExists)

{

return NotFound($"Bucket {bucketName} does not existed.");

}

var request = new ListObjectsV2Request()

{

BucketName = bucketName,

Prefix = prefix

};

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

var s3Objects = result.S3Objects.Select(s =>

{

var urlRequest = new GetPreSignedUrlRequest()

{

BucketName = bucketName,

Key = s.Key,

Expires = DateTime.UtcNow.AddMinutes(30)

};

return new S3Model

{

Name = s.Key.ToString(),

PresignedUrl = \_amazonS3.GetPreSignedURL(urlRequest),

};

});

return Ok(s3Objects);

}

// geyby name and key get json file content data

[HttpGet]

[Route("GetFileByKeyAsync")]

public async Task<IActionResult> GetFileByKeyAsync(string bucketName, string key)

{

// var cachemem = \_distributedCache.GetString("s3BucketData");

try

{

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

if (!bucketExists)

{

return NotFound($"Bucket {bucketName} does not existed.");

}

GetObjectRequest request = new GetObjectRequest()

{

BucketName = bucketName,

Key = key

};

// var s3object = await \_amazonS3.GetObjectAsync(bucketName, key);

//var data = File(s3object.ResponseStream, s3object.Headers.ContentType);

using (GetObjectResponse response = await \_amazonS3.GetObjectAsync(request))

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

{

var conttent = await reader.ReadToEndAsync();

// Imemory cache:

\_memoryCache.Set("JsonFile", conttent, TimeSpan.FromSeconds(80));

// Distributed cache timeout option

var options = new DistributedCacheEntryOptions

{

AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(80),

};

// distribution cache setstring value

\_distributedCache.SetString("JsonFile", conttent, options);

return Ok(conttent);

}

}

catch (AmazonS3Exception ex)

{

throw new Exception(ex.Message);

}

}

[HttpDelete]

[Route("DeleteFileByKeyAsync")]

public async Task<IActionResult> DeleteFileByKeyAsync(string bucketName, string key)

{

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

if (!bucketExists)

{

return NotFound($"Bucket {bucketName} does not existed.");

}

await \_amazonS3.DeleteObjectAsync(bucketName, key);

return NoContent();

}

}

}

UPLOAD JSON FILES AND READ IN DIFFERENT WAYS

FileController.cs:

private readonly IAmazonS3 \_amazonS3;

private readonly IWebHostEnvironment \_webHostEnvironment;

public FilesController(

IAmazonS3 amazonS3,

IWebHostEnvironment webHostEnvironment

)

{

\_amazonS3 = amazonS3;

\_webHostEnvironment = webHostEnvironment;

}

[HttpGet]

[Route("ReadJsonFile")]

public IActionResult ReadJsonFile(int id)

{

//var uploadpath = \_webHostEnvironment.WebRootPath;

//var destinationPath = Path.Combine(uploadpath, "D:\\csharpprojects\\StudentThreeTier\\StudentThreeTier\\Content\\");

//if (!Directory.Exists(destinationPath))

//{

// Directory.CreateDirectory(destinationPath);

//}

//var sourcePath = Path.GetFileName(formFile.Name);

//var fullpath = Path.Combine(destinationPath, sourcePath);

//using(FileStream stream = new FileStream(fullpath, FileMode.Create))

//{

// formFile.CopyTo(stream);

//}

var json = System.IO.File.ReadAllText("D:\\csharpprojects\\StudentThreeTier\\StudentThreeTier\\Content\\Uploads\\client.json");

// var json = System.IO.File.ReadAllText(fullpath);

var jsonData = JsonConvert.DeserializeObject<List<RSAClient>>(json);

var data = jsonData.FirstOrDefault(j => j.RSAClientId == id);

if (data == null)

{

return NotFound($" id {id} is not found.");

}

else

{

var clientModel = new RSAClient

{

RSAClientId = data.RSAClientId,

RSAClientName = data.RSAClientName,

Stores = data.Stores,

};

return Ok(clientModel);

}

}

[HttpPost]

[Route(" UplaodAndReadJsonFile")]

public IActionResult UplaodAndReadJsonFile(IFormFile formFile,int id)

{

var uploadpath = \_webHostEnvironment.WebRootPath;

var destinationPath = Path.Combine(uploadpath, "JSONFILES");

if (!Directory.Exists(destinationPath))

{

Directory.CreateDirectory(destinationPath);

}

var sourcePath = Path.GetFileName(formFile.FileName);

var fullpath = Path.Combine(destinationPath, sourcePath);

using (FileStream stream = new FileStream(fullpath, FileMode.Create))

{

formFile.CopyTo(stream);

}

Console.WriteLine(fullpath);

// read json file

var json = System.IO.File.ReadAllText(fullpath);

var jsonData = JsonConvert.DeserializeObject<List<RSAClient>>(json);

var data = jsonData.FirstOrDefault(item => item.RSAClientId == id);

if(data == null)

{

return NotFound($" Id {id} is invalid");

}

// return Ok(jsonData);

RSAClient clientModel = new RSAClient();

clientModel.RSAClientId = data.RSAClientId;

clientModel.RSAClientName = data.RSAClientName;

clientModel.Stores = data.Stores;

return Ok(clientModel);

}

[HttpGet]

[Route("ReadJsonFile")]

public IActionResult ReadJsonFile(int id)

{

//var uploadpath = \_webHostEnvironment.WebRootPath;

//var destinationPath = Path.Combine(uploadpath, "D:\\csharpprojects\\StudentThreeTier\\StudentThreeTier\\Content\\");

//if (!Directory.Exists(destinationPath))

//{

// Directory.CreateDirectory(destinationPath);

//}

//var sourcePath = Path.GetFileName(formFile.Name);

//var fullpath = Path.Combine(destinationPath, sourcePath);

//using(FileStream stream = new FileStream(fullpath, FileMode.Create))

//{

// formFile.CopyTo(stream);

//}

var json = System.IO.File.ReadAllText("D:\\csharpprojects\\StudentThreeTier\\StudentThreeTier\\Content\\Uploads\\client.json");

// var json = System.IO.File.ReadAllText(fullpath);

var jsonData = JsonConvert.DeserializeObject<List<RSAClient>>(json);

var data = jsonData.FirstOrDefault(j => j.RSAClientId == id);

if (data == null)

{

return NotFound($" id {id} is not found.");

}

else

{

var clientModel = new RSAClient

{

RSAClientId = data.RSAClientId,

RSAClientName = data.RSAClientName,

Stores = data.Stores,

};

return Ok(clientModel);

}

}

[HttpPost]

[Route(" UplaodAndReadJsonFile")]

public IActionResult UplaodAndReadJsonFile(IFormFile formFile,int id)

{

var uploadpath = \_webHostEnvironment.WebRootPath;

var destinationPath = Path.Combine(uploadpath, "JSONFILES");

if (!Directory.Exists(destinationPath))

{

Directory.CreateDirectory(destinationPath);

}

var sourcePath = Path.GetFileName(formFile.FileName);

var fullpath = Path.Combine(destinationPath, sourcePath);

using (FileStream stream = new FileStream(fullpath, FileMode.Create))

{

formFile.CopyTo(stream);

}

Console.WriteLine(fullpath);

// read json file

var json = System.IO.File.ReadAllText(fullpath);

var jsonData = JsonConvert.DeserializeObject<List<RSAClient>>(json);

var data = jsonData?.FirstOrDefault(item => item.RSAClientId == id);

if(data == null)

{

return NotFound($" Id {id} is invalid");

}

// return Ok(jsonData);

RSAClient clientModel = new RSAClient();

clientModel.RSAClientId = data.RSAClientId;

clientModel.RSAClientName = data.RSAClientName;

clientModel.Stores = data.Stores;

return Ok(clientModel);

}

[HttpGet]

[Route("ReadJsonFromFolder")]

public IActionResult ReadJsonFromFolder(int Id)

{

// var basepath = Path.Combine(Environment.CurrentDirectory, @"XMLFiles\");

// var basePath = Path.Combine(\_webHostEnvironment.ContentRootPath, @"JSONFiles\client.json");

// string fileName = @"client.json";

string fileName = @"RSAClient.json";

string currentDirectory = Directory.GetCurrentDirectory();

// get full path with folders names dynamically

string[] fullFilePath = Directory.GetFiles(currentDirectory,fileName, SearchOption.AllDirectories);

var jsonData = System.IO.File.ReadAllText(fullFilePath[0]);

var dataobj = JsonConvert.DeserializeObject<List<RSAClient>>(jsonData);

var data = dataobj?.FirstOrDefault(d => d.RSAClientId == Id);

if(data == null)

{

return NotFound("Invalid id ");

}

return Ok(data);

}

Session Storage

Program.cs:

// CONFIGURE THE SESSION STORAGE SERVICES

/ session storage:

// services.AddMvc().AddSessionStateTempDataProvider();

// Builder.Services.AddDistributedMemoryCache();

Builder.Services. .AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromSeconds(10);

options.Cookie.HttpOnly = true;

options.Cookie.IsEssential = true;

});

services.AddDistributedMemoryCache();

services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromSeconds(10);

options.Cookie.HttpOnly = true;

options.Cookie.IsEssential = true;

});

// Use in application

app.UseSession();

Controller.cs:

[HttpGet]

[Route("GetSessionData")]

public IActionResult GetSessionData()

{

// get session value

var data = HttpContext.Session.GetString("Student");

if(string.IsNullOrEmpty(data))

{

// return BadRequest("Expired session");

var studentlist = \_studentService.GetAllStudent();

return Ok(studentlist);

}

return Ok(data);

}

[HttpGet]

[Route("SetSession")]

public IActionResult SetSessionn()

{

var studentData = \_studentService.GetAllStudent();

// set session value

HttpContext.Session.SetString("Student", JsonConvert.SerializeObject(studentData));

return Ok(studentData);

}

Imemory Cache

Configure AddMemoryCache()

Program.cs:

// add service add memory cache

Builder.Service.AddMemoryCache();

services.AddMemoryCache();

Controller.cs: add ImemoryCache interface inside a constructor class of controller

private readonly IMemoryCache \_memoryCache;

public StudentController(

IMemoryCache memoryCache )

{

\_memoryCache = memoryCache;

}

// set Memory Cache value:

[HttpGet]

[Route("SetSession")]

public IActionResult SetSessionn()

{

// individual:

var studentData = \_studentService.GetAllStudent();

// set memory cache

\_memoryCache.Set("Student", JsonConvert.SerializeObject(studentData), TimeSpan.FromSeconds(80));

// get

var data = \_memoryCache.Get("Student");

if (data != null)

{

return Ok(data);

}

else

{

return Ok(studentData);

}

}

[HttpGet]

[Route("GetSession")]

public IActionResult GetSession()

{

// get memory cache valuea

var getStudents = \_memoryCache.Get("Student");

if (getStudents == null)

{

return BadRequest("No data found");

}

return Ok(getStudents);

}

[Get object from AWS S3 as a File stream](https://stackoverflow.com/questions/42145696/get-object-from-aws-s3-as-a-stream)

public FileStream GetFile(string keyName)

{

using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast2))

{

GetObjectRequest request = new GetObjectRequest

{

BucketName = bucketName,

Key = keyName

};

using (GetObjectResponse response = client.GetObject(request))

using (Stream responseStream = response.ResponseStream)

using (StreamReader reader = new StreamReader(responseStream))

{

// The following outputs the content of my text file:

Console.WriteLine(reader.ReadToEnd());

// Do some magic to return content as a stream

}

}

IDistributedCache

private readonly IDistributedCache \_distributedCache;

public FilesController(

IAmazonS3 amazonS3,

IWebHostEnvironment webHostEnvironment,

IMemoryCache memoryCache,

IDistributedCache distributedCache

)

{

\_amazonS3 = amazonS3;

\_webHostEnvironment = webHostEnvironment;

\_memoryCache = memoryCache;

\_distributedCache = distributedCache;

}

[HttpGet]

[Route("GetFileByKeyAsync")]

public async Task<IActionResult> GetFileByKeyAsync(string bucketName, string key)

{

// var cachemem = \_distributedCache.GetString("s3BucketData");

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

if (!bucketExists)

{

return NotFound($"Bucket {bucketName} does not existed.");

}

var s3object = await \_amazonS3.GetObjectAsync(bucketName, key);

//var data = File(s3object.ResponseStream, s3object.Headers.ContentType);

GetObjectRequest request = new GetObjectRequest()

{

BucketName = bucketName,

Key = key

};

using(GetObjectResponse response = await \_amazonS3.GetObjectAsync(request))

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

{

var conttent = await reader.ReadToEndAsync();

// Distributed cache timeout option

var options = new DistributedCacheEntryOptions

{

AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(80),

};

// distribution cache setstring value

\_distributedCache.SetString("JsonFile", conttent, options);

return Ok(conttent);

}

}

[HttpGet]

[Route("GetJsonFileData")]

public IActionResult GetJsonFileData()

{

var data = \_distributedCache.GetString("JsonFile");

if (data == null)

{

return NotFound("No data found.");

}

return Ok(data);

}