ASP .NET Core Assessment

Sasikumar S 23-04-2022

1. Create a web api provides CRUD facility for

Courses (CourseName, Duration, Trainer, Fee, Mode (online or Offline))

a) Enable CORS,

b) Json Serialization,

c) Enable JWT Authentication

**Database First Approach:**

Create database CourseDB

use CourseDB

Create Table Course(

CourseID Int Identity(1,1) Primary Key,

CourseName Varchar(100) Not Null,

Duration Varchar(10),

Trainer Varchar(20),

Fee Decimal Not Null,

Mode Varchar(20) Not Null)

Create Table UserInfo(

UserId Int Identity(1,1) Not null Primary Key,

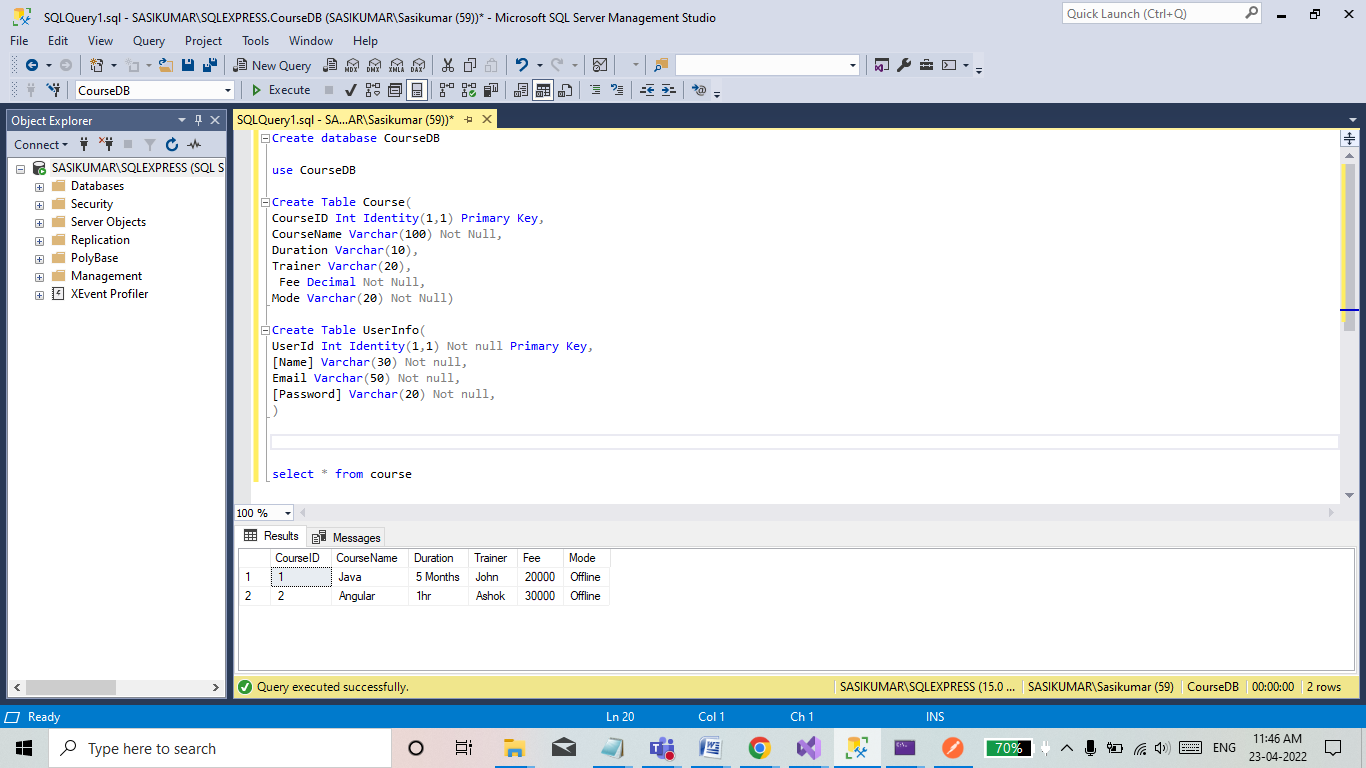
[Name] Varchar(30) Not null,

Email Varchar(50) Not null,

[Password] Varchar(20) Not null,

)

select \* from course



**Auto Created Codes:**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace CourseAPI.Models

{

public partial class Course

{

[Key]

public int CourseId { get; set; }

public string CourseName { get; set; } = null!;

public string? Duration { get; set; }

public string? Trainer { get; set; }

public decimal Fee { get; set; }

public string Mode { get; set; } = null!;

}

}

**UserInfo Code**

using System;

using System.Collections.Generic;

namespace CourseAPI.Models

{

public partial class UserInfo

{

public int UserId { get; set; }

public string Name { get; set; } = null!;

public string Email { get; set; } = null!;

public string Password { get; set; } = null!;

}

}

**Course DB Context Code**

using System;

using System.Collections.Generic;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata;

namespace CourseAPI.Models

{

public partial class CourseDBContext : DbContext

{

public CourseDBContext()

{

}

public CourseDBContext(DbContextOptions<CourseDBContext> options)

: base(options)

{

}

public virtual DbSet<Course> Courses { get; set; } = null!;

public virtual DbSet<UserInfo> UserInfos { get; set; } = null!;

// protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

// {

// if (!optionsBuilder.IsConfigured)

// {

//#warning To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723263.

// optionsBuilder.UseSqlServer("Server=SASIKUMAR\\SQLEXPRESS;Database=CourseDB;Integrated Security=True");

// }

// }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Course>(entity =>

{

entity.ToTable("Course");

entity.Property(e => e.CourseId).HasColumnName("CourseID");

entity.Property(e => e.CourseName)

.HasMaxLength(100)

.IsUnicode(false);

entity.Property(e => e.Duration)

.HasMaxLength(10)

.IsUnicode(false);

entity.Property(e => e.Fee).HasColumnType("decimal(18, 0)");

entity.Property(e => e.Mode)

.HasMaxLength(20)

.IsUnicode(false);

entity.Property(e => e.Trainer)

.HasMaxLength(20)

.IsUnicode(false);

});

modelBuilder.Entity<UserInfo>(entity =>

{

entity.HasKey(e => e.UserId)

.HasName("PK\_\_UserInfo\_\_1788CC4C3140B6BB");

entity.ToTable("UserInfo");

entity.Property(e => e.Email)

.HasMaxLength(50)

.IsUnicode(false);

entity.Property(e => e.Name)

.HasMaxLength(30)

.IsUnicode(false);

entity.Property(e => e.Password)

.HasMaxLength(20)

.IsUnicode(false);

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

}

**AppSetting.json**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"CourseStr": "Server=SASIKUMAR\\SQLEXPRESS;Database=CourseDB;Integrated Security=True"

},

"Jwt": {

"Key": "SomeUserSecretKey",

"Issuer": "InventoryServiceServer",

"Audience": "AuthenticatedClients",

"Subject": "InventoryServiceAccessToken"

}

}

**Program.cs**

using CourseAPI.Models;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

string ConStr = builder.Configuration.GetConnectionString("CourseStr");

builder.Services.AddDbContext<CourseDBContext>(options => options.UseSqlServer(ConStr));

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(Options =>

{

Options.RequireHttpsMetadata = false;

Options.SaveToken = true;

Options.TokenValidationParameters = new TokenValidationParameters()

{

ValidateIssuer = true,

ValidateAudience = true,

ValidAudience = builder.Configuration["Jwt:Audience"],

ValidIssuer = builder.Configuration["Jwt:Issuer"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))

};

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Token Controller**

using CourseAPI.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace CourseAPIAPI.Controllers

{

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

[ApiController]

public class TokenController : ControllerBase

{

private readonly IConfiguration \_configuration;

private readonly CourseDBContext \_context;

public TokenController(IConfiguration configuration,CourseDBContext context)

{

\_configuration = configuration;

\_context = context;

}

[HttpPost]

public async Task<IActionResult> Post(UserInfo \_userData)

{

if (\_userData != null && \_userData.Email != null && \_userData.Password != null)

{

var user = await GetUser(\_userData.Email, \_userData.Password);

if (user != null)

{

//

var claims = new[] {

new Claim(JwtRegisteredClaimNames.Sub, \_configuration["Jwt:Subject"]),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),

new Claim(JwtRegisteredClaimNames.Iat, DateTime.UtcNow.ToString()),

new Claim("Id", user.UserId.ToString()),

new Claim("Name", user.Name),

new Claim("Email", user.Email)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]));

var signIn = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(\_configuration["Jwt:Issuer"], \_configuration["Jwt:Audience"], claims, expires: DateTime.UtcNow.AddDays(1), signingCredentials: signIn);

return Ok(new JwtSecurityTokenHandler().WriteToken(token));

}

else

{

return BadRequest("Invalid credentials");

}

}

else

{

return BadRequest();

}

}

private async Task<UserInfo> GetUser(string email, string password)

{

return await \_context.UserInfos.FirstOrDefaultAsync(u => u.Email == email && u.Password == password);

}

}

}

**Course Controller**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using CourseAPI.Models;

using Microsoft.AspNetCore.Authorization;

namespace CourseAPI.Controllers

{

[Authorize]

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

[ApiController]

public class CoursesController : ControllerBase

{

private readonly CourseDBContext \_context;

public CoursesController(CourseDBContext context)

{

\_context = context;

}

// GET: api/Courses

[HttpGet]

public async Task<ActionResult<IEnumerable<Course>>> GetCourses()

{

return await \_context.Courses.ToListAsync();

}

// GET: api/Courses/5

[HttpGet("{id}")]

public async Task<ActionResult<Course>> GetCourse(int id)

{

var course = await \_context.Courses.FindAsync(id);

if (course == null)

{

return NotFound();

}

return course;

}

// PUT: api/Courses/5

// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPut("{id}")]

public async Task<IActionResult> PutCourse(int id, Course course)

{

if (id != course.CourseId)

{

return BadRequest();

}

\_context.Entry(course).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CourseExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/Courses

// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPost]

public async Task<ActionResult<Course>> PostCourse(Course course)

{

\_context.Courses.Add(course);

Console.WriteLine(course.CourseName);

await \_context.SaveChangesAsync();

Console.WriteLine("Saved");

return CreatedAtAction("GetCourse", new { id = course.CourseId }, course);

}

// DELETE: api/Courses/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteCourse(int id)

{

var course = await \_context.Courses.FindAsync(id);

if (course == null)

{

return NotFound();

}

\_context.Courses.Remove(course);

await \_context.SaveChangesAsync();

return NoContent();

}

private bool CourseExists(int id)

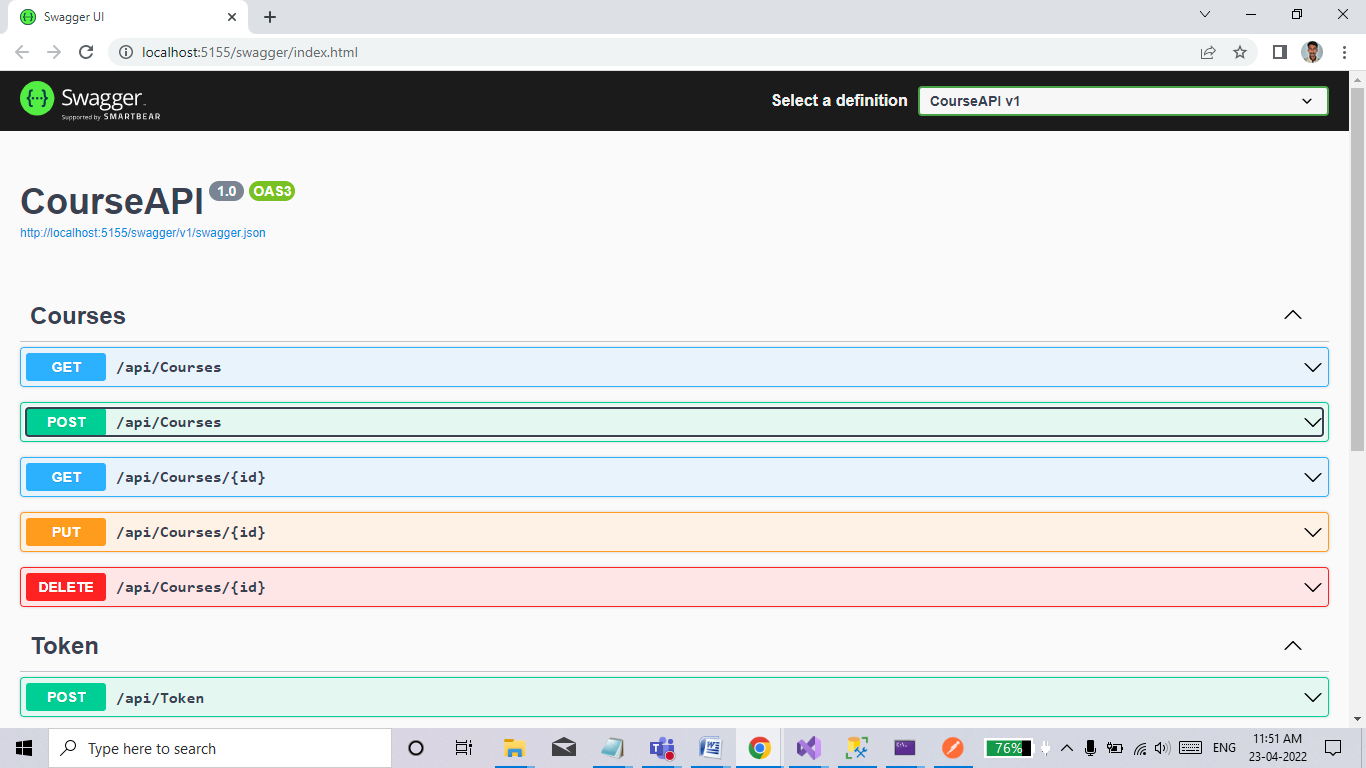
{

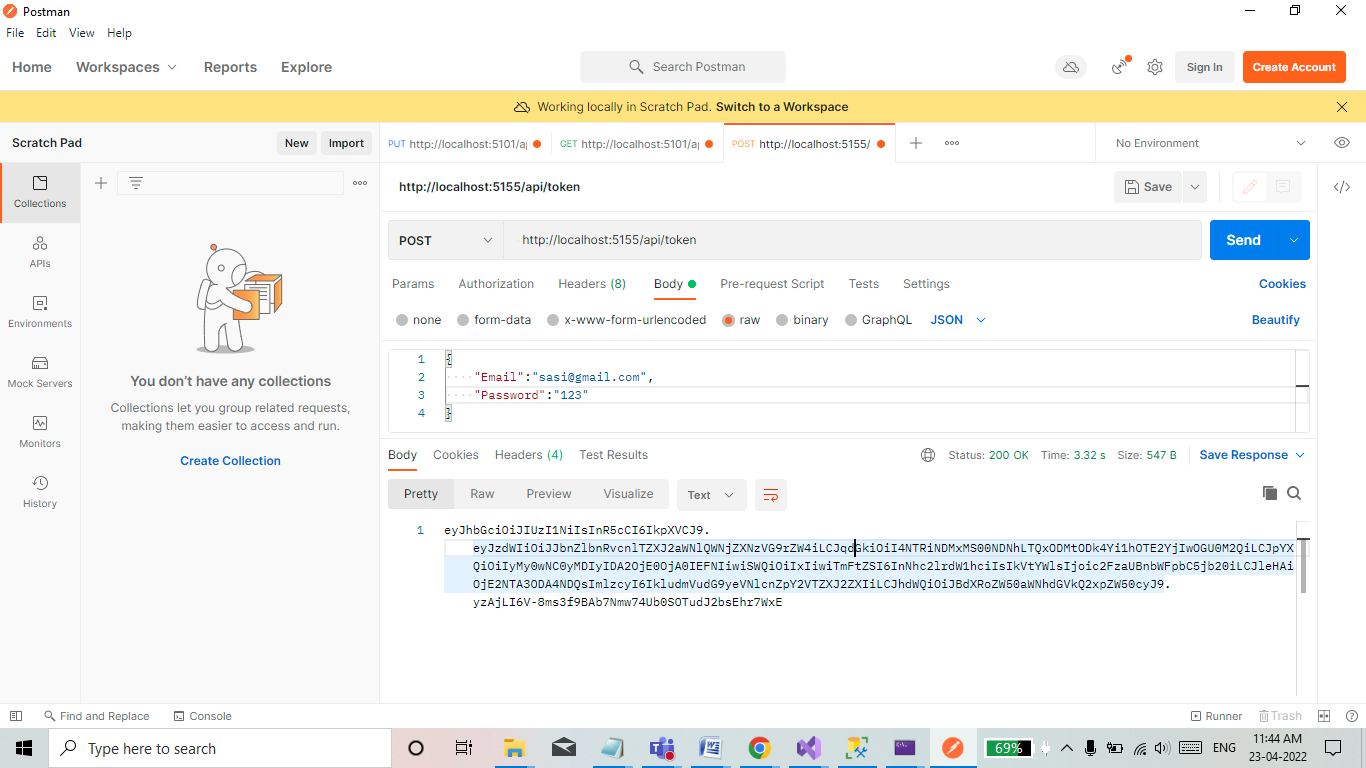
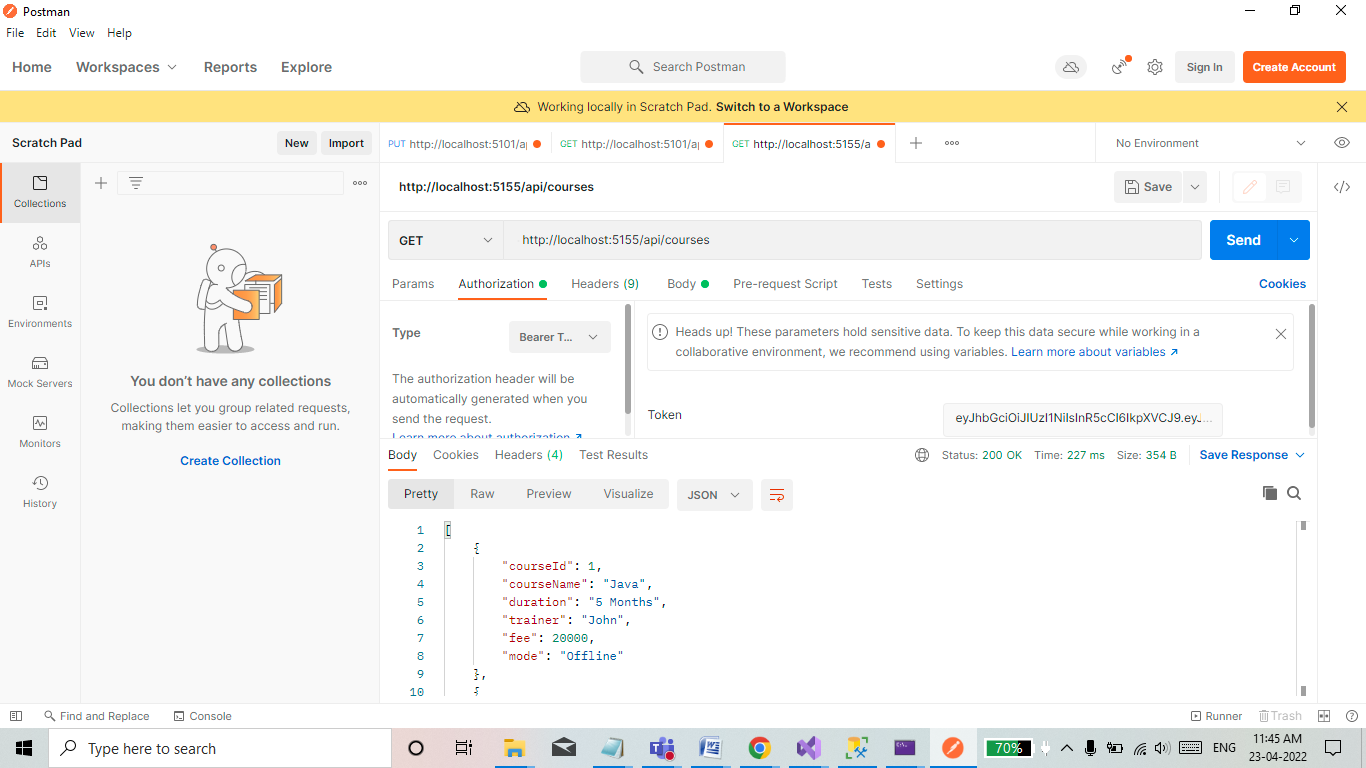
return \_context.Courses.Any(e => e.CourseId == id);

}

}

}



**2. Consume above WebAPI from CORE MVC**

a) Implement Login Module to generate token

b) Create session to store JWT Token

c) Create controller to display courses information for valid user.

**Model Class**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace CourseClient.Models

{

public partial class Course

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int CourseId { get; set; }

public string CourseName { get; set; } = null!;

public string? Duration { get; set; }

public string? Trainer { get; set; }

public decimal Fee { get; set; }

public string Mode { get; set; } = null!;

}

}

**UserInfo**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace CourseClient.Models

{

public class UserInfo

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int UserId { get; set; }

public string Email { get; set; }

public string Password { get; set; }

}

}

**AppSetting.json**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"BaseURL": "http://localhost:5155"

}

**Program.cs**

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

builder.Services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromMinutes(60);

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseSession();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

**Home Controller**

using CourseClient.Models;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using System.Diagnostics;

using System.Text;

namespace CourseClient.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

IConfiguration \_configuration;

string BaseURL;

public HomeController(ILogger<HomeController> logger, IConfiguration configuration)

{

\_logger = logger;

\_configuration = configuration;

BaseURL = \_configuration.GetValue<string>("BaseURL");

}

public IActionResult Index()

{

return View();

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

[HttpPost]

public async Task<IActionResult> LoginUser(UserInfo user)

{

HttpClientHandler clientHandler = new HttpClientHandler();

clientHandler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) => { return true; };

using (var httpClient = new HttpClient(clientHandler))

{

StringContent stringContent = new StringContent(JsonConvert.SerializeObject(user), Encoding.UTF8, "application/json");

using (var response = await httpClient.PostAsync(BaseURL + "/api/token", stringContent))

{

string token = await response.Content.ReadAsStringAsync();

if (token == "Invalid credentials")

{

ViewBag.Message = "Incorrect Email and Password";

return Redirect("~/Home/Index");

}

HttpContext.Session.SetString("JWToken", token);

}

return Redirect("~/Courses/Index");

}

}

public IActionResult Logout()

{

HttpContext.Session.Clear();

return RedirectToAction("Index");

}

}

}

**Course Controller**

using CourseClient.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using System.Net.Http.Headers;

using System.Text;

namespace CourseClient.Controllers

{

public class CoursesController : Controller

{

IConfiguration \_configuration;

private readonly string BaseUrl;

public CoursesController(IConfiguration configuration)

{

\_configuration = configuration;

BaseUrl = \_configuration.GetValue<string>("BaseURl");

}

public async Task<IActionResult> Index()

{

return View(await GetCourses());

}

[HttpGet]

public async Task<List<Course>> GetCourses()

{

var accessToken = HttpContext.Session.GetString("JWToken");

HttpClientHandler clientHandler = new HttpClientHandler();

clientHandler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyError) =>

{

return true;

};

HttpClient client = new HttpClient(clientHandler);

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

string JsonStr = await client.GetStringAsync(BaseUrl + "/api/courses");

var result = JsonConvert.DeserializeObject<List<Course>>(JsonStr);

return result;

}

public async Task<IActionResult> Details(int id)

{

List<Course> courses = await GetCourses();

return View(courses.FirstOrDefault(p => p.CourseId == id));

}

[HttpGet]

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Course course)

{

var accessToken = HttpContext.Session.GetString("JWToken");

Course receivedCourse = new Course();

HttpClientHandler clientHandler = new HttpClientHandler();

clientHandler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) => { return true; };

using (var httpClient = new HttpClient(clientHandler))

{

httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

StringContent content = new StringContent(JsonConvert.SerializeObject(course), Encoding.UTF8, "application/json");

using (var response = await httpClient.PostAsync(BaseUrl + "/api/Courses", content))

{

string apiResponse = await response.Content.ReadAsStringAsync();

receivedCourse = JsonConvert.DeserializeObject<Course>(apiResponse);

if (receivedCourse != null)

{

ViewBag.Message = "Course creation Successfully";

return View(receivedCourse);

}

}

}

ViewBag.Message = "Course creation Failed";

return View();

}

public async Task<Course> GetCourses(int id)

{

var accessToken = HttpContext.Session.GetString("JWToken");

Course receivedCourses = new Course();

HttpClientHandler clientHandler = new HttpClientHandler();

clientHandler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) => { return true; };

using (var httpClient = new HttpClient(clientHandler))

{

httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

using (var response = await httpClient.GetAsync(BaseUrl + "/api/Courses/" + id))

{

Console.WriteLine("Inside using");

if (response.StatusCode == System.Net.HttpStatusCode.OK)

{

string apiResponse = await response.Content.ReadAsStringAsync();

receivedCourses = JsonConvert.DeserializeObject<Course>(apiResponse);

Console.WriteLine(receivedCourses.CourseName);

}

else

ViewBag.StatusCode = response.StatusCode;

}

}

return receivedCourses;

}

public async Task<ActionResult> Edit(int id)

{

Course course = await GetCourses(id);

return View(course);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Edit(int id, Course updatedCourses)

{

updatedCourses.CourseId = id;

var accessToken = HttpContext.Session.GetString("JWToken");

HttpClientHandler clientHandler = new HttpClientHandler();

clientHandler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) => { return true; };

using (var httpClient = new HttpClient(clientHandler))

{

httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

StringContent contents = new StringContent(JsonConvert.SerializeObject(updatedCourses), Encoding.UTF8, "application/json");

using (var response = await httpClient.PutAsync(BaseUrl + "/api/courses/" + id, contents))

{

string apiResponse = await response.Content.ReadAsStringAsync();

if (apiResponse != null)

ViewBag.Message = "Course Updated Successfully";

else

ViewBag.Message = "Course updation Failed";

}

}

return View();

}

public async Task<ActionResult> Delete(int id)

{

Course course = await GetCourses(id);

return View(course);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Delete(int id, IFormCollection collection)

{

var accessToken = HttpContext.Session.GetString("JWToken");

HttpClientHandler clientHandler = new HttpClientHandler();

clientHandler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) => { return true; };

using (var httpClient = new HttpClient(clientHandler))

{

httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

using (var response = await httpClient.DeleteAsync(BaseUrl + "/api/courses/" + id))

{

string apiResponse = await response.Content.ReadAsStringAsync();

}

}

return RedirectToAction("Index");

}

}

}

**Index-HomeController**

@model CourseClient.Models.UserInfo

@{

ViewData["Title"] = "Home Page";

}

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="LoginUser" **asp-controller**="Home">

<div class="form-group">

<**label** **asp-for**="Email" class="control-label">Email</**label**>

<**input** **asp-for**="Email" class="form-control" />

<**span** **asp-validation-for**="Email" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Password" class="control-label">Password</**label**>

<**input** **asp-for**="Password" class="form-control" />

<**span** **asp-validation-for**="Password" class="text-danger"></**span**>

</div>

<div class="form-group">

<input type="submit" value="login" class="btn btn-primary" />

</div>

<**div** **asp-validation-summary**="ModelOnly" class="text-danger">

</**div**>

<label>@Html.Raw(ViewBag.Message)</label>

</**form**>

</div>

</div>

Index.cshtml

@model IEnumerable<CourseClient.Models.Course>

@{

ViewData["Title"] = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Index</h1>

<p>

<a asp-action="Create">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.CourseId)

</th>

<th>

@Html.DisplayNameFor(model => model.CourseName)

</th>

<th>

@Html.DisplayNameFor(model => model.Duration)

</th>

<th>

@Html.DisplayNameFor(model => model.Trainer)

</th>

<th>

@Html.DisplayNameFor(model => model.Fee)

</th>

<th>

@Html.DisplayNameFor(model => model.Mode)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.CourseId)

</td>

<td>

@Html.DisplayFor(modelItem => item.CourseName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Duration)

</td>

<td>

@Html.DisplayFor(modelItem => item.Trainer)

</td>

<td>

@Html.DisplayFor(modelItem => item.Fee)

</td>

<td>

@Html.DisplayFor(modelItem => item.Mode)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.CourseId }) |

@Html.ActionLink("Details", "Details", new { id=item.CourseId}) |

@Html.ActionLink("Delete", "Delete", new { id=item.CourseId})

</td>

</tr>

}

</tbody>

</table>

Create.cshtml

@model CourseClient.Models.Course

@{

ViewData["Title"] = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Create</h1>

<h4>Course</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Create" >

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="CourseName" class="control-label"></**label**>

<**input** **asp-for**="CourseName" class="form-control" />

<**span** **asp-validation-for**="CourseName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Duration" class="control-label"></**label**>

<**input** **asp-for**="Duration" class="form-control" />

<**span** **asp-validation-for**="Duration" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Trainer" class="control-label"></**label**>

<**input** **asp-for**="Trainer" class="form-control" />

<**span** **asp-validation-for**="Trainer" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Fee" class="control-label"></**label**>

<**input** **asp-for**="Fee" class="form-control" />

<**span** **asp-validation-for**="Fee" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Mode" class="control-label"></**label**>

<**input** **asp-for**="Mode" class="form-control" />

<**span** **asp-validation-for**="Mode" class="text-danger"></**span**>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Edit.cshtml

@model CourseClient.Models.Course

@{

ViewData["Title"] = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Edit</h1>

<h4>Course</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label asp-for="CourseId" class="control-label"></label>

<input asp-for="CourseId" class="form-control" />

<span asp-validation-for="CourseId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="CourseName" class="control-label"></label>

<input asp-for="CourseName" class="form-control" />

<span asp-validation-for="CourseName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Duration" class="control-label"></label>

<input asp-for="Duration" class="form-control" />

<span asp-validation-for="Duration" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Trainer" class="control-label"></label>

<input asp-for="Trainer" class="form-control" />

<span asp-validation-for="Trainer" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Fee" class="control-label"></label>

<input asp-for="Fee" class="form-control" />

<span asp-validation-for="Fee" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Mode" class="control-label"></label>

<input asp-for="Mode" class="form-control" />

<span asp-validation-for="Mode" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Details.cshtml

@model CourseClient.Models.Course

@{

ViewData["Title"] = "Details";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Details</h1>

<div>

<h4>Course</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.CourseId)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.CourseId)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.CourseName)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.CourseName)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Duration)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Duration)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Trainer)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Trainer)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Fee)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Fee)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Mode)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Mode)

</dd>

</dl>

</div>

<div>

@Html.ActionLink("Edit", "Edit", new { /\* id = Model.PrimaryKey \*/ }) |

<a asp-action="Index">Back to List</a>

</div>

Delete.cshtml

@model CourseClient.Models.Course

@{

ViewData["Title"] = "Delete";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Course</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.CourseId)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.CourseId)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.CourseName)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.CourseName)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Duration)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Duration)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Trainer)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Trainer)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Fee)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Fee)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Mode)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Mode)

</dd>

</dl>

<form asp-action="Delete">

<input type="submit" value="Delete" class="btn btn-danger" /> |

<a asp-action="Index">Back to List</a>

</form>

</div>

**Output**

