EmployeeManagment

# Controllers (folder)

## HomeController.cs

using EmployeeManagement.Models;

using EmployeeManagement.ViewModels;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.Controllers

{

// Blocks access to all pages exept the login and register, Controller

// level authentication

// [Authorize]

public class HomeController : Controller

{

private readonly IEmployeeRepository \_employeeRepository;

private readonly IHostingEnvironment hostingEnvironment;

private readonly ILogger logger;

private readonly UserManager<ApplicationUser> userManager;

public HomeController(IEmployeeRepository employeeRepository,

IHostingEnvironment hostingEnvironment,

ILogger<HomeController> logger,

UserManager<ApplicationUser> userManager)

{

\_employeeRepository = employeeRepository;

this.hostingEnvironment = hostingEnvironment;

this.logger = logger;

this.userManager = userManager;

}

// Allows access to anonymous user with authentication is implemented

// at the Controller or at the configeration level (Startup.cs)

// [AllowAnonymous]

public ViewResult Index()

{

// Retrieves all employee list

var model = \_employeeRepository.GetAllEmployee();

return View(model);

}

// Allows access to anonymous user with authentication is implemented

// at the Controller or at the configeration level (Startup.cs)

// [AllowAnonymous]

public ViewResult Details(int? id)

{

// throws and exception automatically

// throw new Exception("Error in Details View");

// Check if the employee exists

Employee employee = \_employeeRepository.GetEmployee(id.Value);

if (employee == null)

{

Response.StatusCode = 404;

return View("EmployeeNotFound", id.Value);

}

// ViewModel create a class

HomeDetailsViewModel homeDetailsViewModel = new HomeDetailsViewModel()

{

// id??1 if ID is given it will be displayed, if not default vaule is set to 1

Employee = employee,

PageTitle = "Employee Details"

};

// Sending Strongly Typed View

return View(homeDetailsViewModel);

}

[HttpGet]

// Limits access to the CREATE attribute

// [Authorize]

public ViewResult Create()

{

return View();

}

[HttpGet]

// Limits access to the EDIT attribute

// [Authorize]

public ViewResult Edit(int id)

{

// Inject the Employee List (IEmployeeRepository)

Employee employee = \_employeeRepository.GetEmployee(id);

// Logging types

logger.LogTrace("Trace Log");

logger.LogDebug("Debug Log");

logger.LogInformation("Information Log");

logger.LogWarning("Warning Log");

logger.LogError("Error Log");

logger.LogCritical("Critical Log");

// Populate the variable

EmployeeEditViewModel employeeEditViewModel = new EmployeeEditViewModel

{

Id = employee.Id,

Name = employee.Name,

Email = employee.Email,

Department = employee.Department,

ExistingPhotoPath = employee.PhotoPath

};

return View(employeeEditViewModel);

}

[HttpPost]

// Limits access to the CREATE attribute

// [Authorize]

public IActionResult Create(EmployeeCreateViewModel model)

{

// Verify if fields are valid

if (ModelState.IsValid)

{

// Create a string to capture the fileName of the image file

string uniqueFileName = ProcessUploadedFile(model);

// Creates the new employee and fills the CLASS

Employee newEmployee = new Employee

{

Name = model.Name,

Email = model.Email,

Department = model.Department,

PhotoPath = uniqueFileName

};

// Adds new employee to the Employee Repository

\_employeeRepository.Add(newEmployee);

// Redirects the information back at detail with new employee ID

return RedirectToAction("Details", new { id = newEmployee.Id });

}

return View();

}

[HttpPost]

// Limits access to the EDIT attribute

// [Authorize]

public IActionResult Edit(EmployeeEditViewModel model)

{

// Verify if fields are valid

if (ModelState.IsValid)

{

// Get the injected model employee to update

Employee employee = \_employeeRepository.GetEmployee(model.Id);

// Load the variables

employee.Name = model.Name;

employee.Email = model.Email;

employee.Department = model.Department;

// Debug Logs

logger.LogDebug("ID: " + employee.Id);

logger.LogDebug("Name: " + employee.Name);

logger.LogDebug("Email: " + employee.Email);

logger.LogDebug("Department: " + employee.Department);

logger.LogDebug("Existing Path: " + model.ExistingPhotoPath);

// Check to see if a new img file is being uploaded

if (model.Photo != null)

{

// Check to see if an img exists

if(model.ExistingPhotoPath != null)

{

// Build path to existing img

string filePath = Path.Combine(hostingEnvironment.WebRootPath, "images",

Path.GetFileName(model.ExistingPhotoPath));

// Delete the img file

System.IO.File.Delete(filePath);

}

// Returns a string of the new file name

employee.PhotoPath = ProcessUploadedFile(model);

}

// Adds new employee to the Employee Repository

\_employeeRepository.Update(employee);

// Debug Logs

logger.LogDebug("Updated ID: " + employee.Id);

logger.LogDebug("Updated Name: " + employee.Name);

logger.LogDebug("Updated Email: " + employee.Email);

logger.LogDebug("Updated Department: " + employee.Department);

logger.LogDebug("Updated Path: " + employee.PhotoPath);

// Redirects the information back at detail with new employee ID

return RedirectToAction("Index");

}

return View(model);

}

[HttpPost]

[Authorize(Policy = "DeleteRolePolicy")]

public async Task<IActionResult> Delete(string id)

{

var user = await userManager.FindByIdAsync(id);

if (user == null)

{

ViewBag.ErrorMessage = $"User with ID: {id} cannot be found";

return View("NotFound");

}

else

{

var result = await userManager.DeleteAsync(user);

if (result.Succeeded)

{

return RedirectToAction("ListUsers");

}

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

return View("ListUsers");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ProcessUploadedFile()

\* This method takes the parameter EMPLOYEECREATEVIEWMODEL from the

\* ViewModel folder and prepares the path, in this instance an IMAGE

\* file and copies it to PHOTO column of the USER.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private string ProcessUploadedFile(EmployeeCreateViewModel model)

{

// Create a string to capture the fileName of the image file

string uniqueFileName = null;

// Check to see if the Photo object (model.Photo) is not null

if (model.Photo != null)

{

// Generates the path to the images folder

string uploadsFolder = Path.Combine(hostingEnvironment.WebRootPath, "images");

// Generates a unique fileName for each image file

// string photoFileName = model.Photo.FileName;

string photoFileName = Path.GetFileName(model.Photo.FileName);

uniqueFileName = Guid.NewGuid().ToString() + "\_" + photoFileName;

// Combining both the path and the fileName to be stored on the DATABASE

string filePath = Path.Combine(uploadsFolder, uniqueFileName);

// Creates the image at the specified path

using (var fileStream = new FileStream(filePath, FileMode.Create))

{

model.Photo.CopyTo(fileStream);

}

}

return uniqueFileName;

}

}

}

# Models (folder)

## AppDbContext.cs

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.Models

{

public class AppDbContext : IdentityDbContext<ApplicationUser>

{

public AppDbContext(DbContextOptions<AppDbContext> options): base(options)

{

}

public DbSet<Employee> Employees { get; set; }

// Adding content to the database without going through the web-interface

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Seed();

// Does not all Roles with Users assigned to it to be deleted

foreach (var foreignKey in modelBuilder.Model.GetEntityTypes()

.SelectMany(e => e.GetForeignKeys()))

{

foreignKey.DeleteBehavior = DeleteBehavior.Restrict;

}

}

}

}

## ApplicationUser.cs

using Microsoft.AspNetCore.Identity;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.Models

{

// Extending the IdentityUser Class by adding City.

public class ApplicationUser : IdentityUser

{

public string City { get; set; }

}

}

## Employee.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.Models

{

public class Employee

{

public int Id { get; set; }

[Required]

[MaxLength(50, ErrorMessage = "Name cannot exceed 50 characters")]

public string Name { get; set; }

[Required]

[RegularExpression(@"^[a-zA-z0-9\_.+-]+@[a-sA-Z0-9-]+\.[a-zA-Z0-9-.]+$", ErrorMessage = "Invalid Email Format")]

[Display(Name = "Office Email")]

public string Email { get; set; }

[Required]

public Dept? Department { get; set; }

public string PhotoPath { get; set; }

}

}

## IEmployeeRepository.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.Models

{

public interface IEmployeeRepository

{

Employee GetEmployee(int Id);

IEnumerable<Employee> GetAllEmployee();

Employee Add(Employee employee);

Employee Update(Employee employeeChanges);

Employee Delete(int id);

}

}

## MockEmployeeRepository.cs

## ModelBuilderExtension.cs

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.Models

{

public static class ModelBuilderExtension

{

public static void Seed(this ModelBuilder modelBuilder)

{

modelBuilder.Entity<Employee>().HasData(

new Employee

{

Id = 1,

Name = "Jean",

Department = Dept.HR,

Email = "jyyChristensen@gmail.com"

},

new Employee

{

Id = 2,

Name = "Mark",

Department = Dept.Facilities,

Email = "markjc@gmail.com"

},

new Employee

{

Id = 3,

Name = "Dustin",

Department = Dept.SE,

Email = "dustinsc1977@gmail.com"

}

);

}

}

}

## SQLEntryRepository.cs

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.Models

{

public class SQLEmployeeRepository : IEmployeeRepository

{

private readonly AppDbContext context;

private readonly ILogger<SQLEmployeeRepository> logger;

// Constructor to include, inject AppDbContext Class

public SQLEmployeeRepository(AppDbContext context,

ILogger<SQLEmployeeRepository> logger)

{

this.context = context;

this.logger = logger;

}

public Employee Add(Employee employee)

{

// Adds new employee to Employee Context

context.Employees.Add(employee);

// Saves changes to Employee Context

context.SaveChanges();

// RETURN add employee object

return employee;

}

public Employee Delete(int id)

{

// Locate the Employee (via ID) to be deleted

Employee employee = context.Employees.Find(id);

// Verify that employee information is present

if(employee != null)

{

// Remove the employee information

context.Employees.Remove(employee);

// Save changes made to the context

context.SaveChanges();

}

// RETURN deleted employee object

return employee;

}

public IEnumerable<Employee> GetAllEmployee()

{

// RETURN all employees

return context.Employees;

}

public Employee GetEmployee(int Id)

{

// Logging types

logger.LogTrace("Trace Log");

logger.LogDebug("Debug Log");

logger.LogInformation("Information Log");

logger.LogWarning("Warning Log");

logger.LogError("Error Log");

logger.LogCritical("Critical Log");

// Find and Return the indicated employee (ID)

return context.Employees.Find(Id);

}

public Employee Update(Employee employeeChanges)

{

// Save changes to a variable

var employee = context.Employees.Attach(employeeChanges);

// Indicate that employee information has been modified

employee.State = Microsoft.EntityFrameworkCore.EntityState.Modified;

// Save changes to the Employee Context

context.SaveChanges();

// Return the updated employee object

return employeeChanges;

}

}

}

# Utilities (folder)

# ViewModels (folder)

## EmployeeCreateViewModel.cs

using EmployeeManagement.Models;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.ViewModels

{

public class EmployeeCreateViewModel

{

[Required]

[MaxLength(50, ErrorMessage = "Name cannot exceed 50 characters")]

[Remote(action: "IsUserNameInUse", controller: "Account")]

public string Name { get; set; }

[Required]

[RegularExpression(@"^[a-zA-z0-9\_.+-]+@[a-sA-Z0-9-]+\.[a-zA-Z0-9-.]+$", ErrorMessage = "Invalid Email Format")]

[Remote(action: "IsEmailInUse", controller: "Account")]

[Display(Name = "Office Email")]

public string Email { get; set; }

[Required]

public Dept? Department { get; set; }

public IFormFile Photo { get; set; }

}

}

## EmployeeEditViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace EmployeeManagement.ViewModels

{

public class EmployeeEditViewModel : EmployeeCreateViewModel

{

public int Id { get; set; }

public string ExistingPhotoPath { get; set; }

}

}

## HomeDetailsViewModel.cs

# Views (folder)

# Views-Account (folder)

# Views-Administration (folder)

# Views-Home (folder)

## Create.cshtml

@model EmployeeCreateViewModel

@{

ViewBag.Title = "Create Employee";

}

<h1>Create Employee</h1>

<!--Form Tag Helpers-->

<form enctype="multipart/form-data" **asp-controller**="Home" **asp-action**="Create" method="post" class="mt-3">

<div class="form-group row">

<label **asp-for**="Name" class="col-sm-2 col-form-label"></label>

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

<input **asp-for**="Name" class="form-control" placeholder="Name">

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

</div>

</div>

<div class="form-group row">

<label **asp-for**="Email" class="col-sm-2 col-form-label"></label>

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

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

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

</div>

</div>

<div class="form-group row">

<label **asp-for**="Department" class="col-sm-2 col-form-label"></label>

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

<select **asp-for**="Department" class="custom-select mr-sm-2"

**asp-items**="Html.GetEnumSelectList<Dept>()">

<option **value**="">Please Select</option>

</select>

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

</div>

</div>

<div class="form-group row">

<label **asp-for**="Photo" class="col-sm-2 col-form-label"></label>

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

<div class="custom-file">

<input **asp-for**="Photo" class="form-control custom-file-input" />

<label class="custom-file-label">Choose File...</label>

</div>

</div>

</div>

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

<div class="form-group row">

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

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

</div>

</div>

@section Scripts {

<script>

$(document).ready(function () {

$('.custom-file-input').on("change", function () {

var fileName = $(this).val().split("\\").pop();

$(this).next('.custom-file-label').html(fileName);

});

});

</script>

}

</form>

## Details.cshtml

<!--ViewModel type view, Remove both asterist and on at symbol to activate-->

<!--@\*\*@model EmployeeManagement.ViewModels.HomeDetailsViewModel-->

<!--\_ViewImports.cshtml notation, namespace designated there-->

@model HomeDetailsViewModel

@{

// Layout View

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

// Set up for the title of the webpage

ViewBag.Title = "Employee Details";

// To post a provided image or a no-image

var photoPath = "~/images/" + (Model.Employee.PhotoPath ?? "NoImage.jpg");

}

<!--Strongly Type View-->

<!--@\*\*\*@model EmployeeManagement.Models.Employee-->

<div class="row justify-content-center m-3">

<div class="col-sm-8">

<div class="card">

<div class="card-header">

<h1>@Model.Employee.Name</h1>

</div>

<div class="card-body text-center">

<img class="card-img-top imageThumbNail" **src**="@photoPath" **asp-append-version**="true"/>

<h4>Employee ID: @Model.Employee.Id</h4>

<h4>Email: @Model.Employee.Email</h4>

<h4>Department: @Model.Employee.Department</h4>

</div>

<div class="card-footer text-center">

<a **asp-controller**="home" **asp-action**="index" class="btn btn-primary">Back</a>

<a **asp-controller**="home" **asp-action**="edit" **asp-route-id**="@Model.Employee.Id" class="btn btn-primary">Edit</a>

<a href="#" class="btn btn-danger">Delete</a>

</div>

</div>

</div>

</div>

@section Scripts

{

<script src="~/js/CustomScript.js"></script>

}

# Views-Shared (folder)