Database Connection

using DrugVerizone.Entities;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using DrugVerizone.Models;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace DrugVerizone.DbContexts

{

public class DrugVerifyContext : IdentityDbContext<ApplicationUser>

{

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

{

}

public virtual DbSet<Admin> Admins { get; set; }

//public virtual DbSet<Roles> Roles { get; set; }

public virtual DbSet<Drugs> Drugs { get; set; }

public virtual DbSet<Manufacturer> Manufacturers { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace DrugVerizone.Entities

{

public class Drugs

{

public Guid Id { get; set; }

public string Name { get; set; }

public string NafdacNo { get; set; }

public DateTime ManFactureDate { get; set; }

public DateTime ExpiryDate { get; set; }

public Guid ManufacturerId { get; set; }

[RegularExpression(@"^[A-Za-z][A-Za-z][0-9][0-9]$")]

[Required]

[StringLength(6)]

public string UniqueCode { get; set; }

public DateTime RegisteredDate { get; set; }

public virtual Manufacturer Manufacturer { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace DrugVerizone.Entities

{

public class Manufacturer

{

public Manufacturer()

{

Drugs = new HashSet<Drugs>();

}

public Guid Id { get; set; }

public string Name { get; set; }

public string Address { get; set; }

public string PhoneNo { get; set; }

public string email { get; set; }

public string certificateOfReg { get; set; }

public DateTime RegistrationDate { get; set; }

public virtual ICollection<Drugs> Drugs { get; set; }

}

}

using DrugVerizone.Entities;

using DrugVerizone.Models;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace DrugVerizone.Services

{

public interface IDrugsRepository

{

Task<DrugsViewDto> Create(DrugCreateDto drugs);

public Task<IEnumerable<ManufacturerViewDto>> GetMan();

Task<DrugsViewDto> Delete(Guid drugID);

Task<bool> DrugExist(Guid drugId);

public Task<IEnumerable<DrugsViewDto>> Get();

Task<DrugsViewDto> ListById(Guid id);

Task<bool> Save();

Task<DrugsViewDto> Update(Guid drugID, DrugUpdateDto drugUpdate);

}

}

using AutoMapper;

using DrugVerizone.DbContexts;

using DrugVerizone.Entities;

using DrugVerizone.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace DrugVerizone.Services

{

public class cManufacturerRepository : IManufacturerRepository

{

private readonly DrugVerifyContext \_context;

private readonly IMapper \_mapper;

public cManufacturerRepository(DrugVerifyContext context, IMapper mapper)

{

\_context = context;

\_mapper = mapper;

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (disposing)

{

// dispose resources when needed

}

}

public async Task<bool> Save()

{

return await Task.Run(async () =>

{

return (await \_context.SaveChangesAsync() >= 0);

});

}

public async Task<IEnumerable<ManufacturerViewDto>> Get()

{

return await Task.Run(async () =>

{

var result = await \_context.Manufacturers.Include(m => m.Drugs).ToListAsync<Manufacturer>();

return \_mapper.Map<IEnumerable<ManufacturerViewDto>>(result);

});

}

public async Task<ManufacturerViewDto> ListById(Guid id)

{

return await Task.Run(async () =>

{

if (id == Guid.Empty)

{

throw new ArgumentNullException(nameof(id));

}

var result = await \_context.Manufacturers.Where(x => x.Id == id).FirstOrDefaultAsync();

return \_mapper.Map<ManufacturerViewDto>(result);

});

}

public async Task<ManufacturerViewDto> Create(ManufacturerCreateDto man)

{

return await Task.Run(async () =>

{

if (man == null)

{

throw new ArgumentNullException(nameof(man));

}

var drugEntity = \_mapper.Map<Manufacturer>(man);

drugEntity.Id = Guid.NewGuid();

drugEntity.RegistrationDate = DateTime.Now;

\_context.Manufacturers.Add(drugEntity);

bool saveDrug = await Save();

return \_mapper.Map<ManufacturerViewDto>(drugEntity);

});

}

public async Task<ManufacturerViewDto> Update(Guid manID, ManufacturerUpdateDto drugUpdate)

{

return await Task.Run(async () =>

{

if (drugUpdate == null)

{

throw new ArgumentNullException(nameof(drugUpdate));

}

var drugEntity = await \_context.Manufacturers.FirstOrDefaultAsync(c => c.Id == manID);

if (drugEntity == null)

{

throw new ArgumentNullException(nameof(drugEntity));

}

var result = \_mapper.Map(drugUpdate, drugEntity);

bool save = await Save();

return \_mapper.Map<ManufacturerViewDto>(result);

});

}

public async Task<ManufacturerViewDto> Delete(Guid manID)

{

return await Task.Run(async () =>

{

var value = await \_context.Manufacturers.FirstOrDefaultAsync(d => d.Id == manID);

if (value == null)

{

throw new ArgumentNullException(nameof(manID));

}

\_context.Manufacturers.Remove(value);

await Save();

return \_mapper.Map<ManufacturerViewDto>(value);

});

}

public async Task<bool> ManufacturerExist(Guid manId)

{

return await Task.Run(async () =>

{

if (manId == Guid.Empty)

{

throw new ArgumentNullException(nameof(manId));

}

return await \_context.Manufacturers.AnyAsync(d => d.Id == manId);

});

}

}

}

@model IEnumerable<DrugsViewDto>;

@{

ViewData["Title"] = "Available Drugs";

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

}

<!-- Content Header (Page header) -->

<section class="content-header">

<div class="container-fluid">

<div class="row mb-2">

<div class="col-sm-6">

@\*<h4>@ViewData["Title"]</h4>\*@

</div>

<div class="col-sm-6">

<ol class="breadcrumb float-sm-right">

<li class="breadcrumb-item"><a **asp-area**="" **asp-controller**="Home" **asp-action**="Index">Dashboard</a></li>

<li class="breadcrumb-item active">@ViewData["Title"]</li>

</ol>

</div>

</div>

</div><!-- /.container-fluid -->

</section>

<section class="content">

<div class="row">

<div class="col-12">

<!-- /.card -->

<div class="card">

<div class="card-header">

<h5 class="card-title">List of @ViewData["Title"]</h5>

</div>

<!-- /.card-header -->

<div class="card-body">

<div class="clearfix mb-3">

<div class="float-right">

<a **asp-area**="" **asp-action**="Create" class="btn btn-light">

<span><i class="fa fa-plus"></i> New Drug</span>

</a>

</div>

</div>

<table class="table table-bordered table-striped dataTable" role="grid" aria-describedby="example1\_info">

<thead>

<tr role="row">

<td>Name</td>

<td>Manufactured By</td>

<td>NAFDAC No</td>

<td>Man. Date</td>

<td>Exp. Date</td>

<td>Unique Code</td>

<td>Actions</td>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr role="row">

<td>@(item.Name != null ? item.Name : "NOT STATED")</td>

<td>

<span>@(item.Manufacturers.Name != null ? item.Manufacturers.Name : "NOT STATED")</span>

</td>

<td>

<span>@(item.NafdacNo != null ? item.NafdacNo : "NOT STATED")</span>

</td>

<td>

<span>@(item.ManFactureDate.ToString() != null ? item.ManFactureDate.ToString() : "NOT STATED")</span>

</td>

<td>

<span>@(item.ExpiryDate.ToString() != null ? item.ExpiryDate.ToString() : "NOT STATED")</span>

</td>

<td>

<span>@(item.UniqueCode != null ? item.UniqueCode: "NOT STATED")</span>

</td>

<td class="text-center p-1">

<a **asp-action**="Details" **asp-controller**="Drugs" **asp-route-id**="@item.Id" class="btn btn-light btn-sm " data-toggle="link" data-placement="top" title="Details">

<i class="fas fa-desktop"></i>

</a>

<a **asp-action**="Delete" **asp-controller**="Drugs" **asp-route-id**="@item.Id " class="btn btn-danger btn-sm" data-toggle="link" data-placement="top" title="Delete">

<i class="fas fa-trash"></i>

</a>

</td>

</tr>

}

</tbody>

</table>

</div>

<!-- /.card-body -->

</div>

<!-- /.card -->

</div>

<!-- /.col -->

</div>

</section>

<!-- /.content -->