MVC Core

**Add new Folders:**

Data

public class ApplicationContext:DbContext

{

public DbSet<Job> Jobs { get; set; }

public DbSet<Source> Sources { get; set; }

public DbSet<Status> Statuses { get; set; }

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

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

builder.Seed();

}

}

public static void Seed(this ModelBuilder modelBuilder)

{

modelBuilder.Entity<Goal>().HasData(

new Goal

{

Id = 1,

Description = "Goal 1",

DateCompleted = DateTime.Now,

Comments="Comments 1"

},

new Goal

{

Id = 2,

Description = "Goal 2",

Comments = "Comments 2"

}

);

}

}

**Entities**

public class Job

{

public int Id { get; set; }

public string JobTitle { get; set; }

public int SourceId { get; set; }

public string CompanyAppliedTo { get; set; }

public bool HaveTheyContactedMe { get; set; }

public int StatusId { get; set; }

}

**Services**

public interface IRepository<T> where T : class

{

IEnumerable<T> GetAll();

T Get(Func<T, bool> predicate);

IEnumerable<T> FindBy(Expression<Func<T, bool>> predicate);

void Add(T entity);

void Update(T entity);

void Delete(T entity);

void Attach(T entity);

}

public interface IUnitOfWork : IDisposable

{

//ApplicationContext Context { get; }

void SaveChanges();

IRepository<T> Repository<T>() where T : class;

}

public class Repository<T> : IRepository<T> where T : class

{

private readonly ApplicationContext context;

private DbSet<T> entities;

string errorMessage = string.Empty;

public Repository(ApplicationContext context)

{

this.context = context;

entities = context.Set<T>();

}

public IEnumerable<T> GetAll()

{

return entities.AsEnumerable();

}

public T Get(Func<T, bool> predicate)

{

return entities.SingleOrDefault(predicate);

}

public void Add(T entity)

{

if (entity == null)

{

throw new ArgumentNullException("entity");

}

entities.Add(entity);

}

public void Update(T entity)

{

if (entity == null)

{

throw new ArgumentNullException("entity");

}

}

public void Delete(T entity)

{

if (entity == null)

{

throw new ArgumentNullException("entity");

}

entities.Remove(entity);

}

public IEnumerable<T> FindBy(System.Linq.Expressions.Expression<Func<T, bool>> predicate)

{

IEnumerable<T> query = entities.Where(predicate).AsEnumerable();

return query;

}

public void Attach(T entity)

{

entities.Attach(entity);

}

}

public class UnitOfWork : IUnitOfWork

{

private ApplicationContext entities = null;

public UnitOfWork(ApplicationContext entities)

{

this.entities=entities;

}

public Dictionary<Type, object> repositories = new Dictionary<Type, object>();

public IRepository<T> Repository<T>() where T : class

{

if (repositories.Keys.Contains(typeof(T)) == true)

{

return repositories[typeof(T)] as IRepository<T>;

}

IRepository<T> repo = new Repository<T>(entities);

repositories.Add(typeof(T), repo);

return repo;

}

public void SaveChanges()

{

entities.SaveChanges();

}

private bool disposed = false;

public ApplicationContext Context => throw new NotImplementedException();

protected virtual void Dispose(bool disposing)

{

if (!this.disposed)

{

if (disposing)

{

entities.Dispose();

}

}

this.disposed = true;

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

**ViewModels**

**Startup**

public Startup(IConfiguration configuration)

{

Configuration = configuration;

var builder = new ConfigurationBuilder().SetBasePath(Directory.GetCurrentDirectory()).AddJsonFile("appsettings.json", optional: true);

Configuration = builder.Build();

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

var conn = Configuration.GetConnectionString("DefaultConnection");

services.AddDbContext<ApplicationContext>(options => options.UseSqlServer(conn));

services.AddSingleton(provider => Configuration);

services.AddScoped<IUnitOfWork, UnitOfWork>();

services.AddScoped(typeof(IRepository<>), typeof(Repository<>));

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

**Appsettings.json**

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

//"AllowedHosts": "\*",

//"ConnectionStrings": {

// "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=Goals;Trusted\_Connection=True;MultipleActiveResultSets=true"

//}

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=Goals;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

}

**Package Manager Console commands**

Add-migration Initial

Update-database

remove-database (if needed…)

**Home Controller (and other controllers for dependency injection)**

public class HomeController : Controller

{

private IUnitOfWork unitOfWork;

public HomeController(IUnitOfWork unitOfWork)

{

this.unitOfWork = unitOfWork;

}

public IActionResult Index()

{

List<GoalsListViewModel> model = new List<GoalsListViewModel>();

var goals = unitOfWork.Repository<Goal>().GetAll().ToList();

string strDateCompleted=string.Empty;

foreach(var goal in goals)

{

if(goal.DateCompleted!=null)

{

strDateCompleted = Convert.ToDateTime(goal.DateCompleted).ToShortDateString();

}

else

{

strDateCompleted = string.Empty;

}

model.Add(new GoalsListViewModel { Id = goal.Id, Description = goal.Description, DateCompleted = strDateCompleted, Comments=goal.Comments });

}

return View(model);

}

public IActionResult Details(int id)

{

var goal=unitOfWork.Repository<Goal>().Get(c=>c.Id==id);

string strDateCompleted = string.Empty;

if (goal.DateCompleted != null)

{

strDateCompleted = Convert.ToDateTime(goal.DateCompleted).ToShortDateString();

}

return View(new GoalsListViewModel { Id = goal.Id, Description = goal.Description, DateCompleted = strDateCompleted, Comments = goal.Comments });

}

public ActionResult Create(Goal goal)

{

if (ModelState.IsValid)

{

if (ModelState.IsValid)

{

unitOfWork.Repository<Goal>().Add(goal);

unitOfWork.SaveChanges();

return RedirectToAction("Index");

}

return View(goal);

}

return View(goal);

}

public ActionResult Edit(int id = 0)

{

Goal goal = unitOfWork.Repository<Goal>().Get(c => c.Id == id);

if (goal == null)

{

Response.StatusCode = 404;

return View("Error");

}

var model = new GoalEditModel { Id=goal.Id,Description=goal.Description,DateCompleted=goal.DateCompleted.ToString(),Comments=goal.Comments };

return View(model);

}

//

// POST: /Contacts/Edit/5

[HttpPost]

public ActionResult Edit(GoalEditModel goal)

{

if (ModelState.IsValid)

{

Goal editedGoal = unitOfWork.Repository<Goal>().Get(c => c.Id == goal.Id);

editedGoal.Description = goal.Description;

editedGoal.Comments = goal.Comments;

if(String.IsNullOrEmpty(goal.DateCompleted))

{

editedGoal.DateCompleted = null;

}

else

{

editedGoal.DateCompleted = Convert.ToDateTime(goal.DateCompleted);

}

//unitOfWork.Repository<Goal>().Attach(goal);

unitOfWork.SaveChanges();

return RedirectToAction("Index");

}

return View(goal);

}

public ActionResult Delete(int id = 0)

{

Goal goal = unitOfWork.Repository<Goal>().Get(c => c.Id == id);

if (goal == null)

{

return View();

}

return View(goal);

}

//

// POST: /Contacts/Delete/5

[HttpPost, ActionName("Delete")]

public ActionResult DeleteConfirmed(int id)

{

Goal goal = unitOfWork.Repository<Goal>().Get(c => c.Id == id);

unitOfWork.Repository<Goal>().Delete(goal);

unitOfWork.SaveChanges();

return RedirectToAction("Index");

}

**Razor for a table…**

<table class="table table-hover table-striped">

<thead class=" w-auto">

<tr class="bg-warning">

<th>@Html.DisplayNameFor(model => model.JobTitle)</th>

<th>@Html.DisplayNameFor(model => model.CompanyAppliedTo)</th>

<th>@Html.DisplayNameFor(model => model.Source)</th>

<th>@Html.DisplayNameFor(model => model.HaveTheyContactedMe)</th>

<th>@Html.DisplayNameFor(model => model.Status)</th>

<th></th>

<th></th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>@Html.DisplayFor(model => item.JobTitle)</td>

<td>@Html.DisplayFor(model => item.CompanyAppliedTo)</td>

<td>@Html.DisplayFor(model => item.Source)</td>

<td>@Html.DisplayFor(model => item.HaveTheyContactedMe)</td>

<td>@Html.DisplayFor(model => item.Status)</td>

<td><**a** class="btn btn-primary btn-xs" **asp-action**="Details" **asp-route-id**="@item.Id">Details</**a**></td>

<td><**a** class="btn btn-info btn-xs" **asp-action**="Edit" **asp-route-id**="@item.Id">Edit</**a**></td>

<td><**a** class="btn btn-primary btn-xs" **asp-action**="Delete" **asp-route-id**="@item.Id">Delete</**a**></td>

</tr>

}

</tbody>

</table>

**Razor for a for….with SelectListItem**

<div class="form-group">

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

<**select** **asp-for**="StatusId" **asp-items**="@Model.Statuses">

<**option**>Please select one</**option**>

</**select**>

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

</div>

**Controller code to create SelectList items and the view model**

var job = unitOfWork.Repository<Job>().Get(x => x.Id == Id);

var sources = unitOfWork.Repository<Source>().GetAll().Select(u => new SelectListItem

{

Text = u.Description,

Value = u.Id.ToString()

});

var statuses = unitOfWork.Repository<Status>().GetAll().Select(u => new SelectListItem

{

Text = u.Description,

Value = u.Id.ToString()

});

var model = new JobEditViewModel { Statuses = statuses, Sources = sources,Id=job.Id,JobTitle=job.JobTitle,StatusId=job.StatusId,SourceId=job.SourceId,HaveTheyContactedMe=job.HaveTheyContactedMe,CompanyAppliedTo=job.CompanyAppliedTo };

return View(model);

public class JobEditViewModel

{

public int Id { get; set; }

[Display(Name = "Job Title")]

public string JobTitle { get; set; }

public int SourceId { get; set; }

[Display(Name = "Source")]

public IEnumerable<SelectListItem> Sources { get; set; }

[Display(Name = "Company Applied To")]

public string CompanyAppliedTo { get; set; }

[Display(Name = "Have They Contacted Me?")]

public bool HaveTheyContactedMe { get; set; }

[Display(Name = "Status")]

public IEnumerable<SelectListItem> Statuses { get; set; }

public int StatusId { get; set; }

}