IT2030 Assignment 12 Matt Brown

**ClassConfig.cs**

namespace ClassSchedule.Models

{

internal class ClassConfig : IEntityTypeConfiguration<Class>

{

public void Configure(EntityTypeBuilder<Class> entity)

{

entity.HasOne(c => c.Teacher)

.WithMany(t => t.Classes)

.OnDelete(DeleteBehavior.Restrict);

entity.HasData(

new Class { ClassId = 1, Title = "Sign Language", Number = 101, TeacherId = 1, DayId = 1, MilitaryTime = "1500" },

new Class { ClassId = 2, Title = "Sign Language", Number = 301, TeacherId = 1, DayId = 2, MilitaryTime = "1100" },

new Class { ClassId = 3, Title = "Logic", Number = 101, TeacherId = 4, DayId = 4, MilitaryTime = "1300" },

new Class { ClassId = 4, Title = "Logic", Number = 201, TeacherId = 4, DayId = 4, MilitaryTime = "1500" },

new Class { ClassId = 5, Title = "Early Childhood Education", Number = 101, TeacherId = 2, DayId = 3, MilitaryTime = "1000" },

new Class { ClassId = 6, Title = "Early Childhood Education", Number = 401, TeacherId = 2, DayId = 5, MilitaryTime = "1000" },

new Class { ClassId = 7, Title = "Calculus", Number = 101, TeacherId = 5, DayId = 1, MilitaryTime = "1300" },

new Class { ClassId = 8, Title = "Calculus", Number = 201, TeacherId = 5, DayId = 3, MilitaryTime = "1300" },

new Class { ClassId = 9, Title = "Nonviolence and Social Change", Number = 101, TeacherId = 3, DayId = 4, MilitaryTime = "1400" },

new Class { ClassId = 10, Title = "Nonviolence and Social Change", Number = 201, TeacherId = 3, DayId = 5, MilitaryTime = "1400" }

);

}

}

}

**ClassScheduleUnitOfWork.cs**

namespace ClassSchedule.Models

{

public class ClassScheduleUnitOfWork : IClassScheduleUnitOfWork

{

private ClassScheduleContext context { get; set; }

public ClassScheduleUnitOfWork(ClassScheduleContext ctx) => context = ctx;

private Repository<Day> dayData;

public Repository<Day> Days

{

get

{

if (dayData == null)

dayData = new Repository<Day>(context);

return dayData;

}

}

private Repository<Teacher> teacherData;

public Repository<Teacher> Teachers

{

get

{

if (teacherData == null)

teacherData = new Repository<Teacher>(context);

return teacherData;

}

}

private Repository<Class> classData;

public Repository<Class> Classes

{

get

{

if (classData == null)

classData = new Repository<Class>(context);

return classData;

}

}

public void Save() => context.SaveChanges();

}

}

**IClassScheduleUnitOfWork.cs**

namespace ClassSchedule.Models

{

interface IClassScheduleUnitOfWork

{

public Repository<Class> Classes { get; }

public Repository<Teacher> Teachers { get; }

public Repository<Day> Days { get; }

public void Save();

}

}

**ClassController.cs**

namespace ClassSchedule.Controllers

{

public class ClassController : Controller

{

private ClassScheduleUnitOfWork data { get; set; }

public ClassController(ClassScheduleContext ctx) => data = new ClassScheduleUnitOfWork(ctx);

public RedirectToActionResult Index() => RedirectToAction("Index", "Home");

[HttpGet]

public ViewResult Add()

{

this.LoadViewBag("Add");

return View();

}

[HttpGet]

public ViewResult Edit(int id)

{

this.LoadViewBag("Edit");

var c = this.GetClass(id);

return View("Add", c);

}

[HttpPost]

public IActionResult Add(Class c)

{

if (ModelState.IsValid) {

if (c.ClassId == 0)

data.Classes.Insert(c);

else

data.Classes.Update(c);

data.Classes.Save();

return RedirectToAction("Index", "Home");

}

else {

string operation = (c.ClassId == 0) ? "Add" : "Edit";

this.LoadViewBag(operation);

return View();

}

}

[HttpGet]

public ViewResult Delete(int id)

{

var c = this.GetClass(id);

return View(c);

}

[HttpPost]

public RedirectToActionResult Delete(Class c)

{

data.Classes.Delete(c);

data.Classes.Save();

return RedirectToAction("Index", "Home");

}

// private helper methods

private Class GetClass(int id)

{

var classOptions = new QueryOptions<Class> {

Includes = "Teacher, Day",

Where = c => c.ClassId == id

};

return data.Classes.Get(classOptions);

}

private void LoadViewBag(string operation)

{

ViewBag.Days = data.Days.List(new QueryOptions<Day> {

OrderBy = d => d.DayId

});

ViewBag.Teachers = data.Teachers.List(new QueryOptions<Teacher> {

OrderBy = t => t.LastName

});

ViewBag.Operation = operation;

}

}

}

**HomeController.cs**

namespace ClassSchedule.Controllers

{

public class HomeController : Controller

{

private ClassScheduleUnitOfWork data { get; set; }

public HomeController(ClassScheduleContext ctx) => data = new ClassScheduleUnitOfWork(ctx);

public ViewResult Index(int id)

{

// options for Days query

var dayOptions = new QueryOptions<Day> {

OrderBy = d => d.DayId

};

// options for Classes query

var classOptions = new QueryOptions<Class> {

Includes = "Teacher, Day"

};

// order by Day if no filter. Otherwise, filter by day and order by time.

if (id == 0)

{

classOptions.OrderBy = c => c.DayId;

classOptions.ThenOrderBy = c => c.MilitaryTime;

}

else

{

classOptions.Where = c => c.DayId == id;

classOptions.OrderBy = c => c.MilitaryTime;

}

// execute queries

ViewBag.Days = data.Days.List(dayOptions);

return View(data.Classes.List(classOptions));

}

}

}

**QueryOptions.cs**

namespace ClassSchedule.Models

{

public class QueryOptions<T>

{

// public properties for sorting and filtering

public Expression<Func<T, Object>> OrderBy { get; set; }

public Expression<Func<T, Object>> ThenOrderBy { get; set; }

public Expression<Func<T, bool>> Where { get; set; }

// private string array for include statements

private string[] includes;

// public write-only property for include statements - converts string to array

public string Includes {

set => includes = value.Replace(" ", "").Split(',');

}

// public get method for include statements - returns private string array or empty string array

public string[] GetIncludes() => includes ?? new string[0];

// read-only properties

public bool HasWhere => Where != null;

public bool HasOrderBy => OrderBy != null;

public bool HasThenOrderBy => ThenOrderBy != null;

}

}

**IRepository.cs**

namespace ClassSchedule.Models

{

public interface IRepository<T> where T : class

{

IEnumerable<T> List(QueryOptions<T> options);

T Get(int id);

T Get(QueryOptions<T> options);

void Insert(T entity);

void Update(T entity);

void Delete(T entity);

void Save();

}

}

**Repository.cs**

namespace ClassSchedule.Models

{

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

{

protected ClassScheduleContext context { get; set; }

private DbSet<T> dbset { get; set; }

public Repository(ClassScheduleContext ctx)

{

context = ctx;

dbset = context.Set<T>();

}

public virtual IEnumerable<T> List(QueryOptions<T> options)

{

IQueryable<T> query = dbset;

foreach (string include in options.GetIncludes())

{

query = query.Include(include);

}

if (options.HasWhere)

query = query.Where(options.Where);

if (options.HasOrderBy)

{

if (options.HasThenOrderBy)

{

query = query.OrderBy(options.OrderBy).ThenBy(options.ThenOrderBy);

}

else

{

query = query.OrderBy(options.OrderBy);

}

}

return query.ToList();

}

public virtual T Get(int id) => dbset.Find(id);

public virtual T Get(QueryOptions<T> options)

{

IQueryable<T> query = dbset;

foreach (string include in options.GetIncludes())

{

query = query.Include(include);

}

if (options.HasWhere)

query = query.Where(options.Where);

return query.FirstOrDefault();

}

public virtual void Insert(T entity) => dbset.Add(entity);

public virtual void Update(T entity) => dbset.Update(entity);

public virtual void Delete(T entity) => dbset.Remove(entity);

public virtual void Save() => context.SaveChanges();

}

}