

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

namespace Domain.Entities

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

public partial class Category

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Category()

{

Tasks = new HashSet<Task>();

}

public int Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Task> Tasks { get; set; }

}

}

namespace Domain.Entities

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

public partial class Message

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Message()

{

UsersMessages = new HashSet<UsersMessage>();

}

public int Id { get; set; }

[Required]

[StringLength(500)]

public string Description { get; set; }

public int Status { get; set; }

[Column(TypeName = "datetime2")]

public DateTime Date { get; set; }

public int? Answer { get; set; }

public int PublicOrPrivate { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<UsersMessage> UsersMessages { get; set; }

}

}

namespace Domain.Entities

{

using System;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

public partial class MessangerTaskContext : DbContext

{

public MessangerTaskContext()

: base("name=MessangerTaskContext")

{

this.Configuration.LazyLoadingEnabled = false;

}

public virtual DbSet<Category> Categories { get; set; }

public virtual DbSet<Message> Messages { get; set; }

public virtual DbSet<Task> Tasks { get; set; }

public virtual DbSet<User> Users { get; set; }

public virtual DbSet<UsersMessage> UsersMessages { get; set; }

public virtual DbSet<UsersTask> UsersTasks { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Category>()

.HasMany(e => e.Tasks)

.WithRequired(e => e.Category)

.WillCascadeOnDelete(false);

modelBuilder.Entity<Message>()

.Property(e => e.Description)

.IsFixedLength();

modelBuilder.Entity<Message>()

.HasMany(e => e.UsersMessages)

.WithRequired(e => e.Message)

.HasForeignKey(e => e.MessagesId)

.WillCascadeOnDelete(false);

modelBuilder.Entity<Task>()

.HasMany(e => e.UsersTasks)

.WithRequired(e => e.Task)

.HasForeignKey(e => e.TasksId)

.WillCascadeOnDelete(false);

modelBuilder.Entity<User>()

.HasMany(e => e.UsersMessages)

.WithRequired(e => e.User)

.WillCascadeOnDelete(false);

modelBuilder.Entity<User>()

.HasMany(e => e.UsersTasks)

.WithRequired(e => e.User)

.WillCascadeOnDelete(false);

}

}

}

namespace Domain.Entities

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

public partial class Task

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Task()

{

UsersTasks = new HashSet<UsersTask>();

}

public int Id { get; set; }

[Required]

[StringLength(2000)]

public string Description { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

public int CategoryId { get; set; }

[Column(TypeName = "smalldatetime")]

public DateTime Date { get; set; }

public virtual Category Category { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<UsersTask> UsersTasks { get; set; }

}

}

namespace Domain.Entities

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

[Table("User")]

public partial class User

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public User()

{

UsersMessages = new HashSet<UsersMessage>();

UsersTasks = new HashSet<UsersTask>();

}

public int Id { get; set; }

[Required]

[StringLength(50)]

public string FirstName { get; set; }

[Required]

[StringLength(50)]

public string LastName { get; set; }

[Required]

[StringLength(250)]

public string Email { get; set; }

[Required]

[StringLength(50)]

public string Password { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<UsersMessage> UsersMessages { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<UsersTask> UsersTasks { get; set; }

}

}

namespace Domain.Entities

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

public partial class UsersMessage

{

[Key]

[Column(Order = 0)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int UserId { get; set; }

[Key]

[Column(Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int MessagesId { get; set; }

public int ReceiverSenderStatus { get; set; }

public virtual Message Message { get; set; }

public virtual User User { get; set; }

}

}

namespace Domain.Entities

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

public partial class UsersTask

{

[Key]

[Column(Order = 0)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int UserId { get; set; }

[Key]

[Column(Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int TasksId { get; set; }

public int? Count { get; set; }

public virtual Task Task { get; set; }

public virtual User User { get; set; }

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Domain.Repository

{

public class TaskVM

{

public string TaskName { get; set; }

public string Name { get; set; }

public DateTime Date { get; set; }

public string Description { get; set; }

public string Category { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Domain.Entities;

namespace Domain.Repository

{

public interface IRepository<TEntity> : IDisposable where TEntity : class

{

void Create(TEntity item);

IQueryable<TEntity> Get();

IEnumerable<TaskVM> GetByCategoryAndName(string category, string name, int skip, int take);

void Remove(TEntity item);

void Update(TEntity item);

int Count();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.Entity;

using System.Linq.Expressions;

using Domain.Entities;

namespace Domain.Repository

{

public class MSSQLEFRepository<TEntity> : IRepository<TEntity> where TEntity : class

{

DbContext context;

DbSet<TEntity> dbSet;

public MSSQLEFRepository(DbContext context)

{

this.context = context;

this.dbSet = context.Set<TEntity>();

}

public void Create(TEntity item)

{

dbSet.Add(item);

context.SaveChanges();

}

public IQueryable<TEntity> Get()

{

return dbSet;

}

/// <summary>

/// Get the number of events that satisfy the filter.

/// </summary>

/// <param name="category">Filter category by which the filter is implemented</param>

/// <param name="name">Name of the participant by which the filter is implemented</param>

/// <param name="skip">Number of elements to be skipped (the number of elements that we already have)</param>

/// <param name="take">number of elements to be taken (in our case 15)</param>

public IEnumerable<TaskVM> GetByCategoryAndName(string category, string name, int skip, int take)

{

// id from which we begin to take events

int first = 0;

// id after which we finish taking events

int last = 0;

// Number of events satisfying the filter

int count = 0;

if (category == "All")

category = null;

if (name == "All")

name = null;

// Call the method that will return to us the first and last element, as well as the number of elements.

GetFirstLastEventId(category, name, (skip > 0 ? skip : 0), (take > 0 ? take : 0), ref first, ref last,ref count);

// If we do not have events for this filter, then we skip the query

if (count > 0)

{

// Send the database query to the PartEvent table. Filter by category, name and range between the first and last events Id.

var list = from p in (this.dbSet as DbSet<UsersTask>)

where category == null || p.Task.Category.Name == category

where name == null || ((p.User.FirstName + " " + p.User.LastName) == name)

where (first == last) ? (p.TasksId == first) : (p.TasksId >= first && p.TasksId <= last)

// Make a selection in an anonymous type. We take: Id events. The date of the event. Description of the event.

//The name of the event. The name of the event participant.

select new

{

Id = p.TasksId,

Date = p.Task.Date,

Desription = p.Task.Description,

EventName = p.Task.Name,

Part = p.User.FirstName + " " + p.User.LastName

};

var list2 = list.ToList().OrderBy(x => x.Id);

// List of events that we will send to the client

List<TaskVM> list3 = new List<TaskVM>();

// Group the resulting list by the event Id

var list4 = from m in list2

group m by new { Id = m.Id };

var list5 = list4.OrderBy(x => x.Key.Id);

foreach (var group in list5)

{

int index = 0;

TaskVM a = new TaskVM();

foreach (var i in group.OrderBy(x=>x.Part))

{

if(index<=0)

{

a.TaskName = i.EventName;

a.Date = i.Date.ToLocalTime();

a.Description = i.Desription;

a.Name = i.Part;

index++;

}

else

{

a.Name += ", " + i.Part;

}

}

list3.Add(a);

}

return list3;

}

return new List<TaskVM>();

}

/// <summary>

/// Get the first and last id of the events that satisfy the filter. Get the number of events that satisfy the filter.

/// </summary>

/// <param name="category">Filter category by which the filter is implemented</param>

/// <param name="name">Name of the participant by which the filter is implemented</param>

/// <param name="skip">Number of elements to be skipped (the number of elements that we already have)</param>

/// <param name="take">number of elements to be taken (in our case 15)</param>

/// <param name="first">Id of the element from which the list of events begins</param>

/// <param name="last">Id of the element on which the list of events ends</param>

/// <param name="count">Number of events satisfying the filter</param>

void GetFirstLastEventId(string category, string name, int skip, int take, ref int first, ref int last,ref int count)

{

// Send the database query to the PartEvent table. Filter by category and name

var list = from p in (this.dbSet as DbSet<UsersTask>)

where category == null || p.Task.Category.Name == category

where name == null || ((p.User.FirstName + " " + p.User.LastName) == name)

select new

{

Id = p.TasksId,

};

// Get the list of events. Skip the number of elements equal to skip. And take the number of elements equal to take.

var number = list.Distinct().OrderBy(p => p.Id).Skip(skip).Take(take).ToList();

count = number.Count;

if (count > 0)

{

first = number.First().Id;

last = number.Last().Id;

}

}

public int Count()

{

return dbSet.Count();

}

public void Remove(TEntity item)

{

dbSet.Remove(item);

context.SaveChanges();

}

public void Update(TEntity item)

{

context.Entry(item).State = EntityState.Modified;

context.SaveChanges();

}

private bool disposed = false;

public virtual void Dispose(bool disposing)

{

if (!this.disposed)

{

if (disposing)

{

context.Dispose();

}

}

this.disposed = true;

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

}

}