Agenda: CRUD Operations using Entity Framework

- Basic CRUD Operations using Scaffold Templates
- Separation of work using BO Classes
- Using Single Database Context Object across all Business Objects
- Writing Generic Class / Repository
- Caching in Repository
- Custom Controller Factory and Dependency Injection

Basic CRUD Operations & Scaffold Templates

1. File → New Project → Visual C# → ASP.NET Web Application → OK
2. Select MVC → OK
3. Go to Tools → **NuGet Package Manager** → Manage Nuget Packages for Solution → Add reference to **EntityFramework**
4. Under Models folder add the following Employee and Context classes

```
public class Employee
{
   [Key]
   public int Id { get; set; }
   public string Name { get; set; }
   public decimal Salary { get; set; }
}
public class Department
{
   [Key]
   public int DeptId { get; set; }
   public string DeptName { get; set; }
}
public class EFDemoDbEntities : DbContext
{
   public DbSet<Employee> Employees { get; set; }
   public DbSet<Department> Departments { get; set; }
}
```

5. In Web.Config add the following

```
<connectionStrings>
```

```
    <add name="EFDemoDbEntities" connectionString="Data Source=.\sqlexpress;Integrated
Security=True;database=DemoOrganization" providerName="System.Data.SqlClient"/>
  </connectionStrings>
```

6. Build the project
7. Solution Explorer → Right click on Controller → New → Controller → MVC 5 Controller with Views using Entity Framework
8. Model Class=Employee, DataContextClass=DemoContext, ControllerName=EmployeesController → Add
9. Build and run the application http://localhost:XXXX/Employees (replace XXXX with actual port number)
10. Understand all the code as explained in the Video

Using BO Classes

1. Create a New Folder, Name = BO
2. Under BO Folder, add the EmployeeBO class to the project

```
public class EmployeeBO
{
  private EFDemoDbEntities db = new EFDemoDbEntities();
  public Employee GetById(int id)
  {
    Employee employee = db.Employees.Find(id);
    return employee;
  }
  public IEnumerable<Employee> GetAll ()
  {
    return db.Employees.Include("Department");
  }
  public Employee Add(Employee emp)
  {
    db.Employees.Add(emp);
    db.SaveChanges();
    //Send welcome message to emp by email/sms...
    return emp;
  }
  public void Update(Employee emp)
  {
    db.Entry(emp).State = EntityState.Modified;
```

```
        db.SaveChanges();
    }
    public void Delete(int id)
    {
        Employee employee = db.Employees.Find(id);
        db.Employees.Remove(employee);
        db.SaveChanges();
    }
    protected void Dispose(bool disposing)
    {
        db.Dispose();
    }
}
```

3. Under BO Folder, add the DepartmentBO class to the project

```
public class DepartmentBO
{
    EFDemoDbEntities db = new EFDemoDbEntities();
    public IEnumerable<Department> GetAll ()
    {
        return db.Departments;
    }
}
```

4. Edit the code of EmployeesContoller as below:

```
public class EmployeesController : Controller
{
    EmployeeBO objEmpBO = new EmployeeBO();
    // GET: /Employee/
    public ActionResult Index()
    {
        var employees = objEmpBO.GetAll();
        return View(employees.ToList());
    }
    // GET: /Employee/Details/5
    public ActionResult Details(int id = 0)
```

```
{
    Employee employee = objEmpBO.GetById(id);
    if (employee == null)
    {
        return HttpNotFound();
    }
    return View(employee);
}
// GET: /Employee/Create
public ActionResult Create()
{
    DepartmentBO objDeptBO = new DepartmentBO();
    ViewBag.DeptId = new SelectList(objDeptBO.GetAll(), "DeptId", "DeptName");
    return View();
}
// POST: /Employee/Create
[HttpPost]
public ActionResult Create(Employee employee)
{
    if (ModelState.IsValid)
    {
        objEmpBO.Add(employee);
        return RedirectToAction("Index");
    }
    DepartmentBO objDeptBO = new DepartmentBO();
    ViewBag.DeptId = new SelectList(objDeptBO.GetAll(), "DeptId", "DeptName");
    return View(employee);
}
// GET: /Employee/Edit/5
public ActionResult Edit(int id = 0)
{
    Employee employee = objEmpBO.GetById(id);
    if (employee == null)
    {
        return HttpNotFound();
    }
```

```
    DepartmentBO objDeptBO = new DepartmentBO();
    ViewBag.DeptId = new SelectList(objDeptBO.GetAll(), "DeptId", "DeptName",employee.DeptId);
    return View(employee);
}
// POST: /Employee/Edit/5
[HttpPost]
public ActionResult Edit(Employee employee)
{
    if (ModelState.IsValid)
    {
        objEmpBO.Update(employee);
        return RedirectToAction("Index");
    }
    DepartmentBO objDeptBO = new DepartmentBO();
    ViewBag.DeptId = new SelectList(objDeptBO.GetAll (), "DeptId", "DeptName",employee.DeptId);
    return View(employee);
}

//
// GET: /Employee/Delete/5
public ActionResult Delete(int id = 0)
{
    Employee employee = objEmpBO.GetById(id);
    if (employee == null)
    {
        return HttpNotFound();
    }
    return View(employee);
}
//
// POST: /Employee/Delete/5
[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
{
    objEmpBO.Delete(id);
    return RedirectToAction("Index");
```

```
    }
}
```

5. Build and Run the application

Writing Generic Class / Repository

1. To the project add the following GenericBO class in BO folder

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
namespace EFDemoApp.BusinessObjects
{
    public class Repository<T> : IDisposable where T:class
    {
        DemoDbEntities db = new DemoDbEntities();
        DbSet<T> ds;
        public Repository()
        {
            ds = db.Set<T>();
        }
        public IEnumerable<T> GetAll()
        {
            return ds.ToList();
        }
        public T GetById(int id)
        {
            return ds.Find(id);
        }
        public T Add(T entity)
        {
            ds.Add(entity);
            db.SaveChanges();
            //Welcome email
            return entity;
        }
        public void Update(T entity)
```

```
    {
        db.Entry<T>(entity).State = EntityState.Modified;
        db.SaveChanges();
    }
    public void Delete(int id)
    {
        T entity = ds.Find(id);
        ds.Remove(entity);
        db.SaveChanges();
    }
    public void Dispose()
    {
        db.Dispose();
    }
  }
}
```

2. Edit EmployeeBO and DepartmentBO as below

```
public class EmployeeBO
{
  Repository<Employee> rep = new Repository<Employee>();
  public IEnumerable<Employee> GetAll()
  {
    return rep.GetAll();
  }
  public Employee GetById(int id)
  {
    return rep.GetById(id);
  }
  public Employee Add(Employee entity)
  {
    return rep.Add(entity);
  }
  public void Update(Employee entity)
  {
    rep.Update(entity);
```

```
    }
    public void Delete(int id)
    {
        rep.Delete(id);
    }
    public void Dispose()
    {
        rep.Dispose();
    }
}
```

```
public class DepartmentBO
{
    private Repository<Department> rep = new Repository<Department>();
    public IEnumerable<Department> GetAll()
    {
        return rep.GetAll();
    }
    public Department GetById(int id)
    {
        return rep.GetById(id);
    }
    public Department Add(Department dept)
    {
        rep.Add(dept);
        return dept;
    }
    public void Update(Department dept)
    {
        rep.Update(dept);
    }
    public void Delete(int id)
    {
        rep.Delete(id);
    }
    public void Dispose()
```

```
  {
    rep.Dispose();
  }
}
```

3. Build and Run the application

Using Single Context Object across all Business Objects

With the current code, both BO class are creating their own Context object and this is not recommended. Instead we should be able to create only one context and share the same between all the BO objects using in a given request.

4. Add the following to the project

```
public class ContextHelper
{
  public static EFDemoDbEntities GetContext()
  {
    if (HttpContext.Current.Items["context"] == null)
      HttpContext.Current.Items.Add("context", new EFDemoDbEntities());
    return context (EFDemoDbEntities)HttpContext.Current.Items["context"];
  }
}
```

5. In Repository Class

   Replace `private EFDemoDbEntities db = new EFDemoDbEntities();`

   with

   `private EFDemoDbEntities db = ContextHelper.GetContext();`

6. Build and Run the application.

Caching in Repository

1. Add reference to **System.Runtime.Caching**
2. Edit the GenericBO as below:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
```

```
using System.Linq;
using System.Runtime.Caching;



namespace EFDemoApp.BusinessObjects
{
   public class Repository<T> : IDisposable where T:class
   {
      DemoDbEntities db = ContextHelper.GetDataContext();// new DemoDbEntities();
      DbSet<T> ds;
      ObjectCache cache = MemoryCache.Default;
      public Repository()
      {
         ds = db.Set<T>();
      }
      public IEnumerable<T> GetAll()
      {
         Type tp = typeof(T);
         string type = tp.ToString();
         if (cache[type] == null)
         {
            cache[type] = ds.ToList();
         }
         return (IEnumerable<T>) cache[type];
      }
      public T GetById(int id)
      {
         return ds.Find(id);
      }

      public T Add(T entity)
      {
         ds.Add(entity);
         db.SaveChanges();
         Type tp = typeof(T);
         string type = tp.ToString();
```

```
        cache.Remove(type);
        //Welcome email
        return entity;
    }
    public void Update(T entity)
    {
        db.Entry<T>(entity).State = EntityState.Modified;
        db.SaveChanges();
        Type tp = typeof(T);
        string type = tp.ToString();
        cache.Remove(type);
    }
    public void Delete(int id)
    {
        T entity = ds.Find(id);
        ds.Remove(entity);
        db.SaveChanges();
        Type tp = typeof(T);
        string type = tp.ToString();
        cache.Remove(type);
    }
    public void Dispose()
    {
        db.Dispose();
    }
  }
}
```


Dependency Injection and Custom Controller Factory

Dependency Injection is a technique to separate the creation of dependencies from the main class under consideration. Using DI you inject the objects needed by a class typically through a constructor.

```
public class CustomControllerFactory : DefaultControllerFactory
{
  public override IController CreateController(RequestContext requestContext, string controllerName)
  {
    if (controllerName == "Employees")
```

```
        {
            return new EmployeesController(new EmployeeBO(), new DepartmentBO());
        }
        return base.CreateController(requestContext, controllerName);
    }
}
protected void Application_Start()
{
    IControllerFactory factory = new CustomControllerFactory();
    ControllerBuilder.Current.SetControllerFactory(factory);
}
```