Generic Repository Pattern with ASP.NET MVC and Entity Framework

Introduction
You may introduce with an Object Oriented Design Principle name DRY – Don't repeat yourself. It is more important in Multi-tier architecture. We can use generic repository pattern to implement DRY.

What is Repository Pattern?
In most of the business operation we have to perform CRUD (Create, Read, Update and Delete) operation. A repository basically works as a mediator between our business logic layer and our data access layer of the application

Benefits of Repository Pattern
- Centralizes data logic or service logic.
- Provides a substitution point for the unit tests for both business logic and data access logic
- Provides a flexible architecture
- Can adopt new change easily
- Domain driven development is easier

What is Generic Repository Pattern?
Generic Repository is a pattern by which we can use single repository for data access of all models. Generally, we used one repository for one model to access data.

Benefits of Generic Repository Pattern
- Reduce redundancy of code
- Force developer to work same pattern – Possibility of less error or no error
- Easy to maintain – Centralize data access logic

Implementation Repository Pattern with ASP.NET MVC and Entity Framework
Let's consider a project to keep Employee Information. Here I will show CRUD operation on employee information.

Tools and Technology used
I used following tools and technology to develop the project – Implementation of generic repository

1. Visual Studio 2013
2. Visual C#
3. ASP.NET MVC 5
4. Entity Framework 6
5. Razor view engine

Step 1: Create an ASP.NET MVC 5 application using Visual Studio 2013. I kept the application name "GenericRepo". Help: How to create first application using asp.net MVC

Step 2: Configure connection string in web.config

```
  <connectionStrings>
```

```
    <add name="DefaultConnection" connectionString="Data Source=localhost;Initial
Catalog=GenericRepoDB;User ID=sa; Password=leads@123"
providerName="System.Data.SqlClient" />
  </connectionStrings>
```

Step 3: Create Model – “Employee”

```
    public class Employee
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string FatherName { get; set; }
        public string MotherName { get; set; }
        public string Designation { get; set; }
        public string Dept { get; set; }
    }
```

Step 4: Create a DbContext name GenericDbContext in Repository folder.

```
    public class GenericRepoContext : DbContext
    {
        public GenericRepoContext()
            : base("DefaultConnection")
        {
        }

        public DbSet<Employee> Employees { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {

        }
    }
```

Step 5: Create IGenericRepository and GenericRepository in Repository folder

```
    interface IGenericRepository<T> where T : class
    {
        IEnumerable<T> SelectAll();
        T SelectByID(object id);
        void Insert(T obj);
        void Update(T obj);
        void Delete(object id);
        void Save();
    }



public class GenericRepository<T> : IGenericRepository<T> where T : class
    {
        private GenericRepoContext db = null;
        private DbSet<T> table = null;
        public GenericRepository()
        {
            this.db = new GenericRepoContext();
            table = db.Set<T>();
        }
        public GenericRepository(GenericRepoContext db)
        {
            this.db = db;
            table = db.Set<T>();
        }
        public IEnumerable<T> SelectAll()
        {
            return table.ToList();
        }
        public T SelectByID(object id)
        {
            return table.Find(id);
        }
        public void Insert(T obj)
        {
            table.Add(obj);
        }
        public void Update(T obj)
        {
            table.Attach(obj);
            db.Entry(obj).State = EntityState.Modified;
        }
        public void Delete(object id)
        {
            T existing = table.Find(id);
            table.Remove(existing);
        }
        public void Save()
        {
            db.SaveChanges();
        }
    }
```

Step 6: Create a controller name – EmployeeController. Select template “MVC5 Controller with read/write action”

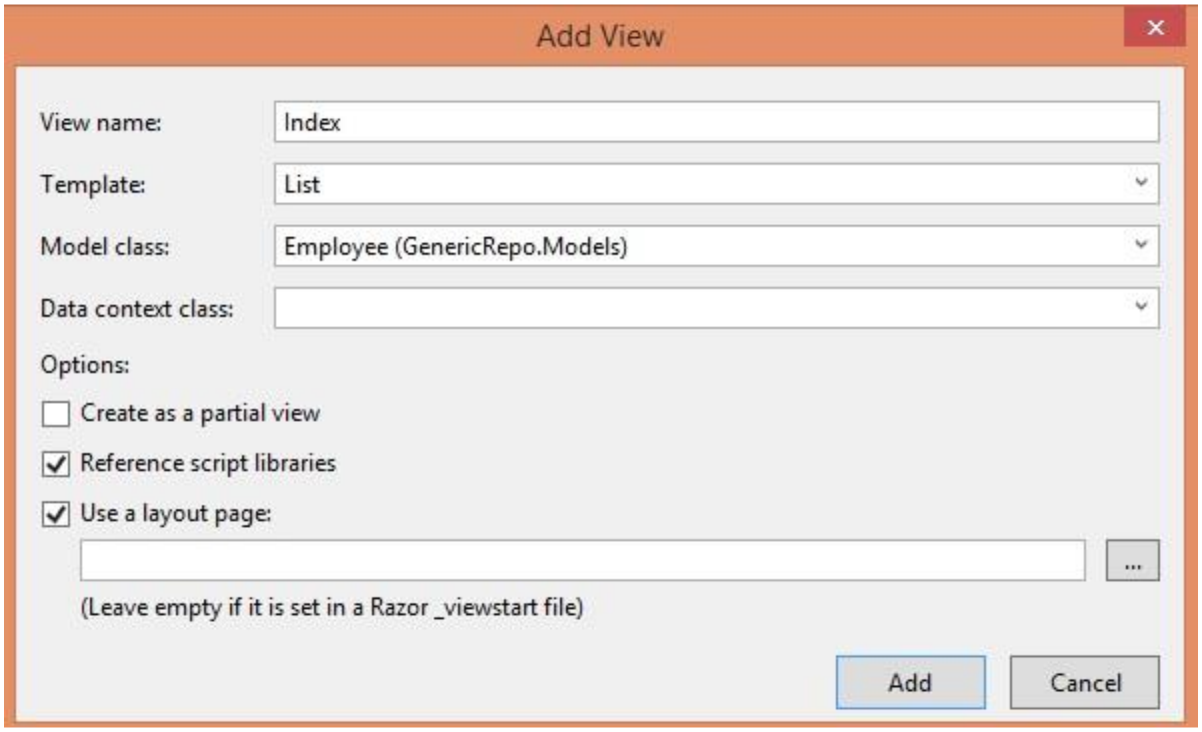
```
    public class EmployeeController : Controller
    {
        private IGenericRepository<Employee> repository = null;
        public EmployeeController()
        {
            this.repository = new GenericRepository<Employee>();
        }


        // GET: Employee
        public ActionResult Index()
        {
            var employee = repository.SelectAll().ToList();
            return View(employee);
        }

        // GET: Employee/Details/5
        public ActionResult Details(int id)
```

```
{
    var employee = repository.SelectByID(id);
    return View(employee);
}

// GET: Employee/Create
public ActionResult Create()
{
    return View();
}

// POST: Employee/Create
[HttpPost]
public ActionResult Create(Employee employee)
{
    if (ModelState.IsValid)
    {
        repository.Insert(employee);
        repository.Save();

        return RedirectToAction("Index");
    }
    return View(employee);
}

// GET: Employee/Edit/5
public ActionResult Edit(int id)
{
    var employee = repository.SelectByID(id);
    return View(employee);
}

// POST: Employee/Edit/5
[HttpPost]
public ActionResult Edit(Employee employee)
{
    try
    {
        repository.Update(employee);
        repository.Save();
        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}

// GET: Employee/Delete/5
public ActionResult Delete(int id)
{
    var employee = repository.SelectByID(id);
    return View(employee);
}

// POST: Employee/Delete/5
[HttpPost]
public ActionResult Delete(int id, FormCollection collection)
```

```
        {
            try
            {
                repository.Delete(id);
                repository.Save();
                return RedirectToAction("Index");
            }
            catch
            {
                return View();
            }
        }
    }
```

Step 7: Create List, Edit, Delete and details page against EmployeeController.

Create a list view



View name: Index
Template: List
Model class: Employee (GenericRepo.Models)
Data context class:
Options:
☐ Create as a partial view
☑ Reference script libraries
☑ Use a layout page:
(Leave empty if it is set in a Razor _viewstart file)

Index.cshtml

```
@model IEnumerable<GenericRepo.Models.Employee>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

```
<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.FatherName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.MotherName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Designation)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Dept)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FatherName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.MotherName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Designation)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Dept)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}

</table>
```

Create Edit View

Create edit view as like list view and choose edit template for that.

_CreateOrEdit.cshtml

```
@model GenericRepo.Models.Employee

    <div class="form-group">
        @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label
col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class =
"form-control" } })
            @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-
danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.FatherName, htmlAttributes: new { @class =
"control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.FatherName, new { htmlAttributes = new {
@class = "form-control" } })
            @Html.ValidationMessageFor(model => model.FatherName, "", new { @class =
"text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.MotherName, htmlAttributes: new { @class =
"control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.MotherName, new { htmlAttributes = new {
@class = "form-control" } })
            @Html.ValidationMessageFor(model => model.MotherName, "", new { @class =
"text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Designation, htmlAttributes: new { @class =
"control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Designation, new { htmlAttributes = new {
@class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Designation, "", new { @class =
"text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Dept, htmlAttributes: new { @class = "control-label
col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Dept, new { htmlAttributes = new { @class =
"form-control" } })
            @Html.ValidationMessageFor(model => model.Dept, "", new { @class = "text-
danger" })
        </div>
    </div>

    <div class="form-group">
```

```
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Create" class="btn btn-default" />
        </div>
    </div>
```

Edit.cshtml

```
@model GenericRepo.Models.Employee


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Employee</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.Partial("_CreateOrEdit", Model)
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Create Delete View

Create delete view as like list view and choose delete template for that.

Delete.cshtml

```
@model GenericRepo.Models.Employee

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Employee</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.FatherName)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.FatherName)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.MotherName)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.MotherName)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Designation)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Designation)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Dept)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Dept)
        </dd>

    </dl>

    @using (Html.BeginForm()) {
        @Html.AntiForgeryToken()

        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            @Html.ActionLink("Back to List", "Index")
        </div>
    }
</div>
```

Create Details View

Create delete view as like list view and choose delete template for that.

Details.cshtml

```
@model GenericRepo.Models.Employee

<div>
    <h4>Employee</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.FatherName)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.FatherName)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.MotherName)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.MotherName)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Designation)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Designation)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Dept)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Dept)
        </dd>

    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "Edit", new { id = Model.Id }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

Step 8: Add a link “Employee” to _Layout page like below

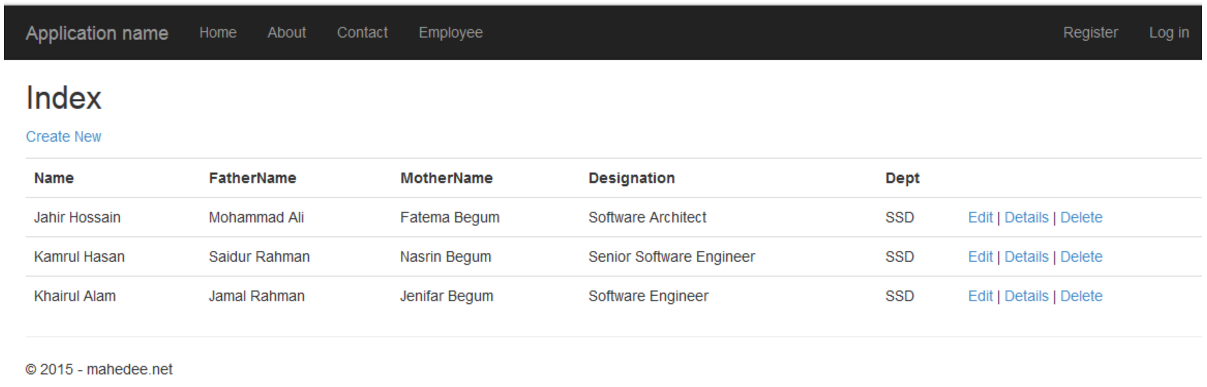
```
         <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                    <li>@Html.ActionLink("Employee", "Index", "Employee")</li>
          </ul>
```

Step 9: Write following command in package manager console

```
PM> Enable-Migrations -ContextTypeName GenericRepoContext
PM> Add-Migration initalcreate
PM> Update-Database -Verbose -Force
```

Now your project is ready. Run application and execute CRUD operation on it. Output of the application like below.

Output:



Index

Create New

Name	FatherName	MotherName	Designation	Dept	
Jahir Hossain	Mohammad Ali	Fatema Begum	Software Architect	SSD	Edit | Details | Delete
Kamrul Hasan	Saidur Rahman	Nasrin Begum	Senior Software Engineer	SSD	Edit | Details | Delete
Khairul Alam	Jamal Rahman	Jenifar Begum	Software Engineer	SSD	Edit | Details | Delete

© 2015 - mahedee.net

References:

1. http://www.tugberkugurlu.com/archive/generic-repository-pattern-entity-framework-asp-net-mvc-and-unit-testing-triangle
2. http://www.codeproject.com/Articles/631668/Learning-MVC-Part-Repository-Pattern-in-MVC-App
3. http://www.codeproject.com/Articles/631668/Learning-MVC-Part-Repository-Pattern-in-MVC-App
4. http://www.codeproject.com/Articles/688929/Repository-Pattern-and-Unit-of
5. http://blog.falafel.com/implement-step-step-generic-repository-pattern-c/
6.
7.