2.16 Using Generic Classes To Perform CRUD Operations



This section will guide you to:

* Perform CRUD operations with generic class

This guide has fifteen subsections, namely:

2.16.1 Creating an ASP.NET MVC project

2.16.2 Creating a Student table in a database

2.16.3 Adding a connection string to web.config

2.16.4 Creating a Student model

2.16.5 Creating a Generic Database Context class

2.16.6 Creating an interface IGenericRepository

2.16.7 Creating GenericRepository to implement IGenericRepository interface

2.16.8 Creating Index.cshml to list students

2.16.9 Creating CreateOrEdit.cshtml to add/edit a student

2.16.10 Creating Details.cshtml to see details of a student

2.16.11 Creating Delete.cshtml to delete a student

2.16.12 Creating StudentController to manage the views and the model

2.16.13 Building the project

2.16.14 Publishing and running the project

2.16.15 Pushing the code to your GitHub repositories

**Step 2.16.1:** Creating an ASP.NET MVC project to use generic classes to do CRUD operations

* Open Visual Studio.
* From the top menu, select **File->New->Project**.
* In **Create A New Project** screen, select **ASP.NET Web Application(.Net Framework)** from the list of available project types and click on **Next.**
* Please select the project type where the **language** of the project is **C#.**
* Enter **Project Name** as **Phase3Section2.20** and click on **Create.**
* From the list of project sub-types, choose **Web MVC** and uncheck **Configure for HTTPS.** Click on **Create.**
* This will create the files for an ASP.NET MVC project.

**Step 2.16.2:** Creating a Student table in a database

* Open SQL Server Management Studio. In the login screen, make a note of the **Server Name** value as you will need to put in your ASP.NET application.
* In **Object Explorer**,right click **Databases** and choose **New Database**.
* Enter **Database name** as **School1** and click **Ok**.
* In the **Object Explorer**,expand **School1->Tables.** Right click **Tables** and choose **New->Table**.
* For the first row enter ID as **Column Name**,int as **Data Type**, anduncheck **Allow Nulls**.
* In **Column Properties**,go to **Identity Specification** and expand it. Double click **Is Identity** to make it **Yes**.
* For the next row enter Name as **Column Name**,varchar(100) as **Data Type**, anduncheck **Allow Nulls**.
* For the next row enter Address as **Column Name**,varchar(100) as **Data Type**,anduncheck **Allow Nulls**.
* For the next row enter Email as **Column Name**,varchar(75) as **Data Type**, anduncheck **Allow Nulls**.
* For the next row enter Class as **Column Name**,varchar(5) as **Data Type**, anduncheck **Allow Nulls**.
* Click on the x icon to close the table grid tab. Click **Yes** on the save dialog.
* For **Enter a name** put Student and press **Ok**.
* In the **Object Explorer**,expand **School1->Tables->Student.** Right click **Tables** and choose **Edit Top 200 Rows**.
* Add in a few rows of data with random values.
* Close the Management Studio.

**Step 2.16.3:** Adding connection string to web.config

* In the **Solution Explorer**, double click **web-config.xml**.
* Add the following script above the ending </configuration> tag.

<connectionStrings>

<add name="School1Entities" connectionString="metadata=res://\*/Student.csdl|res://\*/Student.ssdl|res://\*/Student.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=<your local sql server>;initial catalog=School1;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

* Be sure to change <Your local sql server> to the actual name of your SQL server instance.

**Step 2.16.4:** Creating a Student model

* In **Solution Explorer**,right click **Models** and choose **Add->Class.**
* Enter Student.cs as **Name** and click **Add.**
* Add the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Web;

**namespace** Phase3Section2.\_32.Models

{

**public** **class** Student

{

**public** **int** ID { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **string** Address { **get**; **set**; }

**public** **string** Email { **get**; **set**; }

**public** **string** Class { **get**; **set**; }

}

}

**Step 2.16.5:** Creating a Generic Database Context class

* In **Solution Explorer**,right click **Models** and choose **Add->Class**.
* Enter Student.cs as **Name** and click **Add**.
* Add the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Data.Entity;

**using** System.Linq;

**using** System.Web;

**namespace** Phase3Section2.\_32.Models

{

**public** **class** GenericDBContext: DbContext

{

**public** GenericDBContext()

: **base**("School1Entities")

{

}

**public** DbSet<Student> Students{ **get**; **set**; }

**protected** **override** **void** OnModelCreating(DbModelBuilder modelBuilder)

{

}

}

}

**Step 2.16.6:** Creating an interface IGenericRepository

* In the **Solution Explorer**,right click **Phase3Section2.32** and choose **Add->New Folder**.
* Enter **Repository** asthe name.
* Right click **Repository** and choose **Add->Class**.
* Enter IGenericRepository.cs as **Name** and click **Add**.
* Enter the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Linq.Expressions;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** Phase3Section2.\_32.Repository

{

**public** **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();

}

}

**Step 2.16.7:** Creating GenericRepository to implement IGenericRepository interface

* In the **Solution Explorer**,right click **Repository** and choose **Add->Class**.
* Enter GenericRepository.cs as **Name** and click **Add**.
* Enter the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Data.Entity;

**using** System.Linq;

**using** System.Web;

**using** Phase3Section2.\_32.Models;

**namespace** Phase3Section2.\_32.Repository

{

**public** **class** GenericRepository<t> : IGenericRepository<t> where T : **class**

{

**private** GenericDBContext db = **null**;

**private** DbSet<t> table = **null**;

**public** GenericRepository()

{

**this**.db = **new** GenericDBContext();

table = db.Set<t>();

}

**public** GenericRepository(GenericDBContext 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 2.16.8:** Creating Index.cshml to list students

* In **Solution Explorer**,right click **Views** and click **Add->New Folder**.
* Enter Student as folder name.
* Right click **Student** and choose **Add->View**.
* Enter **View Name** as Index and click **Add**.
* Enter the following script:

@model IEnumerable<Phase3Section2.\_32.Models.Student>

@{

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.Address)

</**th**>

<**th**>

@Html.DisplayNameFor(model => model.Email)

</**th**>

<**th**>

@Html.DisplayNameFor(model => model.Class)

</**th**>

<**th**></**th**>

</**tr**>

@foreach (var item in Model) {

<**tr**>

<**td**>

@Html.DisplayFor(modelItem => item.Name)

</**td**>

<**td**>

@Html.DisplayFor(modelItem => item.Address)

</**td**>

<**td**>

@Html.DisplayFor(modelItem => item.Email)

</**td**>

<**td**>

@Html.DisplayFor(modelItem => item.Class)

</**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**>

**Step 2.16.9:** Creating CreateOrEdit.cshtml to add/edit a student

* In **Solution Explorer**, **r**ight click **Student** and choose **Add->View**.
* Enter **View Name** as CreateOrEdit and click **Add**.
* Enter the following script:

@model Phase3Section2.\_32.Models.Student

@{

ViewBag.Title = "CreateOrEdit";

}

<**h2**>CreateOrEdit</**h2**>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<**div** class="form-horizontal">

<**h4**>Student</**h4**>

<**hr** />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

@Html.HiddenFor(model => model.ID)

<**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.Address, htmlAttributes: new { @class = "control-label col-md-2" })

<**div** class="col-md-10">

@Html.EditorFor(model => model.Address, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Address, "", new { @class = "text-danger" })

</**div**>

</**div**>

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

@Html.LabelFor(model => model.Email, htmlAttributes: new { @class = "control-label col-md-2" })

<**div** class="col-md-10">

@Html.EditorFor(model => model.Email, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Email, "", new { @class = "text-danger" })

</**div**>

</**div**>

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

@Html.LabelFor(model => model.Class, htmlAttributes: new { @class = "control-label col-md-2" })

<**div** class="col-md-10">

@Html.EditorFor(model => model.Class, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Class, "", new { @class = "text-danger" })

</**div**>

</**div**>

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

<**div** class="col-md-offset-2 col-md-10">

<**input** type="submit" value="Save" class="btn btn-default" />

</**div**>

</**div**>

</**div**>

}

<**div**>

@Html.ActionLink("Back to List", "Index")

</**div**>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

**Step 2.16.10:** Creating Details.cshtml to see details of a student

* In **Solution Explorer**, **r**ight click **Student** and choose **Add->View**.
* Enter **View Name** as Details and click **Add**.
* Enter the following script:

@model Phase3Section2.\_32.Models.Student

@{

ViewBag.Title = "Details";

}

<**h2**>Details</**h2**>

<**div**>

<**h4**>Student</**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.Address)

</**dt**>

<**dd**>

@Html.DisplayFor(model => model.Address)

</**dd**>

<**dt**>

@Html.DisplayNameFor(model => model.Email)

</**dt**>

<**dd**>

@Html.DisplayFor(model => model.Email)

</**dd**>

<**dt**>

@Html.DisplayNameFor(model => model.Class)

</**dt**>

<**dd**>

@Html.DisplayFor(model => model.Class)

</**dd**>

</**dl**>

</**div**>

<**p**>

@Html.ActionLink("Edit", "Edit", new { id = Model.ID }) |

@Html.ActionLink("Back to List", "Index")

</**p**>

**Step 2.16.11:** Creating Delete.cshtml to delete a student

* In **Solution Explorer**, right click **Student** and choose **Add->View**.
* Enter **View Name** as Delete and click **Add**.
* Enter the following script:

@model Phase3Section2.\_32.Models.Student

@{

ViewBag.Title = "Delete";

}

<**h2**>Delete</**h2**>

<**h3**>Are you sure you want to delete this?</**h3**>

<**div**>

<**h4**>Student</**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.Address)

</**dt**>

<**dd**>

@Html.DisplayFor(model => model.Address)

</**dd**>

<**dt**>

@Html.DisplayNameFor(model => model.Email)

</**dt**>

<**dd**>

@Html.DisplayFor(model => model.Email)

</**dd**>

<**dt**>

@Html.DisplayNameFor(model => model.Class)

</**dt**>

<**dd**>

@Html.DisplayFor(model => model.Class)

</**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**>

**Step 2.16.12:** Creating StudentController to auto-create the views for CRUD operations

* In the **Solution Explorer**,expand **Controllers.** Right click **Controllers** and choose **Add->Controller**.
* From the list of types, choose **MVC5 Controller Empty**.
* Add the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Web;

**using** System.Web.Mvc;

**using** Phase3Section2.\_32.Models;

**using** Phase3Section2.\_32.Repository;

**namespace** Phase3Section2.\_32.Controllers

{

**public** **class** StudentController : Controller

{

**private** IGenericRepository<Student> repository = **null**;

**public** StudentController()

{

**this**.repository = **new** GenericRepository<Student>();

}

// GET: Employee

**public** ActionResult Index()

{

**var** student = repository.SelectAll().ToList();

**return** View(student);

}

// GET: Employee/Details/5

**public** ActionResult Details(**int** id)

{

**var** student = repository.SelectByID(id);

**return** View(student);

}

// GET: Employee/Create

**public** ActionResult Create()

{

**return** View();

}

// POST: Employee/Create

[HttpPost]

**public** ActionResult Create(Student student)

{

**if** (ModelState.IsValid)

{

repository.Insert(student);

repository.Save();

**return** RedirectToAction("Index");

}

**return** View(student);

}

// GET: Employee/Edit/5

**public** ActionResult Edit(**int** id)

{

**var** student = repository.SelectByID(id);

**return** View(student);

}

// POST: Employee/Edit/5

[HttpPost]

**public** ActionResult Edit(Student student)

{

**try**

{

repository.Update(student);

repository.Save();

**return** RedirectToAction("Index");

}

**catch**

{

**return** View();

}

}

// GET: Employee/Delete/5

**public** ActionResult Delete(**int** id)

{

**var** student = repository.SelectByID(id);

**return** View(student);

}

// 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 2.16.13:** Building the project

* From the top menu, choose **Build->Build Solution**.
* If any compile errors are shown, fix them as required.

**Step 2.16.14:** Publishing and running the project

* From the top menu, select **Debug->Start Without Debugging**.
* This will execute the program in the default browser.
* To see the student pages, go to the url : http://localhost:xxxx/student.

**Step 2.16.15:** Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.

cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add .

Commit the changes using the following command:

git commit -m “Changes have been committed.”

Push the files to the folder you created initially using the following command:

git push -u origin master