CRUD Operations In ASP.NET MVC 5 Using ADO.NET

**Step 1 :**Create an MVC Application.  
  
Now let us start with a stepbystep approach from the creation of simple MVC application as in the following:

1. "Start", then "All Programs" and select "Microsoft Visual Studio 2015".
2. "File", then "New" and click "Project..." then select "ASP.NET Web Application Template", then provide the Project a name as you wish and click on *OK.*After clicking, the following window will appear:  
     
   Graphical user interface, application, email

   Description automatically generated
3. As shown in the preceding screenshot, click on *Empty*template and check *MVC*option, then click *OK*. This will create an empty MVC web application whose Solution Explorer will look like the following:  
     
   Graphical user interface, text, application

   Description automatically generated

**Step 2:** Create Model Class

Now let us create the model class named EmpModel.cs by right clicking on model folder as in the following screenshot:

Graphical user interface, application

Description automatically generated  
  
**Note:**It is not mandatory that Model class should be in Model folder, it is just for better readability you can create this class anywhere in the solution explorer. This can be done by creating different folder name or without folder name or in a separate class library.  
  
EmpModel.cs class code snippet:

**public** **class** EmpModel

  {

      [Display(Name = "Id")]

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

      [Required(ErrorMessage = "First name is required.")]

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

      [Required(ErrorMessage = "City is required.")]

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

      [Required(ErrorMessage = "Address is required.")]

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

  } 

In the above model class we have added some validation on properties with the help of DataAnnotations. **Step 3:**  Create Controller.  
  
Now let us add the MVC 5 controller as in the following screenshot:

Graphical user interface, application

Description automatically generated

After clicking on *Add*button it will show the following window. Now specify the *Controller name* as *Employee*with suffix *Controller*as in the following screenshot:

Graphical user interface, application, Teams

Description automatically generated with medium confidence  
  
**Note:** The controller name must be having suffix as 'Controller' after specifying the name of controller.

After clicking on *Add* button controller is created with by default code that support CRUD operations and later on we can configure it as per our requirements. **Step 4 :** Create Table and Stored procedures.  
  
Now before creating the views let us create the table name *Employee*in database according to our model fields to store the details:

Table

Description automatically generated  
  
I hope you have created the same table structure as shown above. Now create the stored procedures to insert, update, view and delete the details as in the following code snippet:  
  
**To Insert Records**

**Create** **procedure** [dbo].[AddNewEmpDetails]

(

   @**Name** **varchar** (50),

   @City **varchar** (50),

   @Address **varchar** (50)

)

**as**

**begin**

**Insert** **into** Employee **values**(@**Name**,@City,@Address)

**End** 

**To View Added Records**

**Create** **Procedure** [dbo].[GetEmployees]

**as**

**begin**

**select** \***from** Employee

**End** 

**To Update Records**

**Create** **procedure** [dbo].[UpdateEmpDetails]

(

   @EmpId **int**,

   @**Name** **varchar** (50),

   @City **varchar** (50),

   @Address **varchar** (50)

)

**as**

**begin**

**Update** Employee

**set** **Name**=@**Name**,

   City=@City,

   Address=@Address

**where** Id=@EmpId

**End** 

**To Delete Records**

**Create** **procedure** [dbo].[DeleteEmpById]

(

   @EmpId **int**

)

**as**

**begin**

**Delete** **from** Employee **where** Id=@EmpId

**End** 

**Step 5:** Create Repository class.  
  
Now create Repository folder and Add EmpRepository.cs class for database related operations, after adding the solution explorer will look like the following screenshot:

Graphical user interface, application

Description automatically generated  
  
Now create methods in EmpRepository.cs to handle the CRUD operation as in the following screenshot:  
  
**EmpRepository.cs**

**public** **class** EmpRepository

{

**private** SqlConnection con;

    //To Handle connection related activities

**private** **void** connection()

    {

**string** constr = ConfigurationManager.ConnectionStrings["getconn"].ToString();

        con = **new** SqlConnection(constr);

    }

//To Add Employee details

**public** **bool** AddEmployee(EmpModel obj)

{

connection();

SqlCommand com = **new** SqlCommand("AddNewEmpDetails", con);

com.CommandType = CommandType.StoredProcedure;

com.Parameters.AddWithValue("@Name", obj.Name);

com.Parameters.AddWithValue("@City", obj.City);

com.Parameters.AddWithValue("@Address", obj.Address);

con.Open();

**int** i = com.ExecuteNonQuery();

con.Close();

**if** (i >= 1)

{

**return** **true**;

}

**else**

{

**return** **false**;

}

    }

    //To view employee details with generic list

**public** List<EmpModel> GetAllEmployees()

    {

        connection();

        List<EmpModel> EmpList =**new** List<EmpModel>();

        SqlCommand com = **new** SqlCommand("GetEmployees", con);

        com.CommandType = CommandType.StoredProcedure;

        SqlDataAdapter da = **new** SqlDataAdapter(com);

        DataTable dt = **new** DataTable();

        con.Open();

        da.Fill(dt);

        con.Close();

        //Bind EmpModel generic list using dataRow

**foreach** (DataRow dr **in** dt.Rows)

        {

            EmpList.Add(

**new** EmpModel {

                    Empid = Convert.ToInt32(dr["Id"]),

                    Name =Convert.ToString( dr["Name"]),

                    City = Convert.ToString( dr["City"]),

                    Address = Convert.ToString(dr["Address"])

                }

                );

        }

**return** EmpList;

    }

    //To Update Employee details

**public** **bool** UpdateEmployee(EmpModel obj)

    {

        connection();

        SqlCommand com = **new** SqlCommand("UpdateEmpDetails", con);

        com.CommandType = CommandType.StoredProcedure;

        com.Parameters.AddWithValue("@EmpId", obj.Empid);

        com.Parameters.AddWithValue("@Name", obj.Name);

        com.Parameters.AddWithValue("@City", obj.City);

        com.Parameters.AddWithValue("@Address", obj.Address);

        con.Open();

**int** i = com.ExecuteNonQuery();

        con.Close();

**if** (i >= 1)

        {

**return** **true**;

        }

**else**

        {

**return** **false**;

        }

    }

    //To delete Employee details

**public** **bool** DeleteEmployee(**int** Id)

    {

        connection();

        SqlCommand com = **new** SqlCommand("DeleteEmpById", con);

        com.CommandType = CommandType.StoredProcedure;

        com.Parameters.AddWithValue("@EmpId", Id);

        con.Open();

**int** i = com.ExecuteNonQuery();

        con.Close();

**if** (i >= 1)

        {

**return** **true**;

        }

**else**

        {

**return** **false**;

        }

    }

}

**Step 6 :** Create Methods into the EmployeeController.cs file.  
  
Now open the EmployeeController.cs and create the following action methods:

1. **public** **class** EmployeeController : Controller
2. {

    // GET: Employee/GetAllEmpDetails

**public** ActionResult GetAllEmpDetails()

    {

        EmpRepository EmpRepo = **new** EmpRepository();

        ModelState.Clear();

**return** View(EmpRepo. GetAllEmployees ());

    }

    // GET: Employee/AddEmployee

**public** ActionResult AddEmployee()

    {

**return** View();

    }

    // POST: Employee/AddEmployee

    [HttpPost]

**public** ActionResult AddEmployee(EmpModel Emp)

    {

**try**

        {

**if** (ModelState.IsValid)

            {

                EmpRepository EmpRepo = **new** EmpRepository();

**if** (EmpRepo.AddEmployee(Emp))

                {

                    ViewBag.Message = "Employee details added successfully";

                }

            }

**return** View();

        }

**catch**

        {

**return** View();

        }

    }



    // GET: Employee/EditEmpDetails/5

**public** ActionResult EditEmpDetails(**int** id)

    {

        EmpRepository EmpRepo = **new** EmpRepository();

**return** View(EmpRepo.GetAllEmployees().Find(Emp => Emp.Empid == id));     }

      // POST: Employee/EditEmpDetails/5

    [HttpPost]

**public** ActionResult EditEmpDetails(**int** id,EmpModel obj)

    {

**try**

        {

                EmpRepository EmpRepo = **new** EmpRepository();

                EmpRepo.UpdateEmployee(obj);

**return** RedirectToAction("GetAllEmpDetails");

        }

**catch**

        {

**return** View();

        }

    }

    // GET: Employee/DeleteEmp/5

**public** ActionResult DeleteEmp(**int** id)

    {

**try**

        {

            EmpRepository EmpRepo = **new** EmpRepository();

**if** (EmpRepo.DeleteEmployee(id))

            {

                ViewBag.AlertMsg = "Employee details deleted successfully";

            }

**return** RedirectToAction("GetAllEmpDetails");

        }

**catch**

        {

**return** View();

        }

    }

}

**Step 7:**Create Views. **Create the Partial view to Add the employees**  
  
To create the Partial View to add Employees, right click on ActionResult method and then click *Add view*. Now specify the view name, template name and model class in EmpModel.cs and click on *Add*button as in the following screenshot:

Graphical user interface, text, application, email

Description automatically generated  
  
After clicking on Add button it generates the strongly typed view whose code is given below:  
  
**AddEmployee.cshtml**

1. @model CRUDUsingMVC.Models.EmpModel

4. @**using** (Html.BeginForm())
5. {
7. @Html.AntiForgeryToken()
9. <div **class**="form-horizontal">
10. <h4>Add Employee</h4>
11. <div>
12. @Html.ActionLink("Back to Employee List", "GetAllEmpDetails")
13. </div>
14. <hr />
15. @Html.ValidationSummary(**true**, "", **new** { @**class** = "text-danger" })

18. <div **class**="form-group">
19. @Html.LabelFor(model => model.Name, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
20. <div **class**="col-md-10">
21. @Html.EditorFor(model => model.Name, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
22. @Html.ValidationMessageFor(model => model.Name, "", **new** { @**class** = "text-danger" })
23. </div>
24. </div>
26. <div **class**="form-group">
27. @Html.LabelFor(model => model.City, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
28. <div **class**="col-md-10">
29. @Html.EditorFor(model => model.City, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
30. @Html.ValidationMessageFor(model => model.City, "", **new** { @**class** = "text-danger" })
31. </div>
32. </div>
34. <div **class**="form-group">
35. @Html.LabelFor(model => model.Address, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
36. <div **class**="col-md-10">
37. @Html.EditorFor(model => model.Address, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
38. @Html.ValidationMessageFor(model => model.Address, "", **new** { @**class** = "text-danger" })
39. </div>
40. </div>
42. <div **class**="form-group">
43. <div **class**="col-md-offset-2 col-md-10">
44. <input type="submit" value="Save" **class**="btn btn-default" />
45. </div>
46. </div>
47. <div **class**="form-group">
48. <div **class**="col-md-offset-2 col-md-10" style="color:green">
49. @ViewBag.Message
51. </div>
52. </div>
53. </div>
55. }
57. <script src="~/Scripts/jquery-1.10.2.min.js"></script>
58. <script src="~/Scripts/jquery.validate.min.js"></script>
59. <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>

**To View Added Employees**  
  
To view the employee details let us create the partial view named GetAllEmpDetails:

Graphical user interface, text, application, email

Description automatically generated  
  
Now click on add button, it will create GetAllEmpDetails.cshtml strongly typed view whose code is given below:  
  
**GetAllEmpDetails.CsHtml**

1. @model IEnumerable<CRUDUsingMVC.Models.EmpModel>
3. <p>
4. @Html.ActionLink("Add New Employee", "AddEmployee")
5. </p>
7. <table **class**="table">
8. <tr>
10. <th>
11. @Html.DisplayNameFor(model => model.Name)
12. </th>
13. <th>
14. @Html.DisplayNameFor(model => model.City)
15. </th>
16. <th>
17. @Html.DisplayNameFor(model => model.Address)
18. </th>
19. <th></th>
20. </tr>
22. @**foreach** (var item **in** Model)
23. {
24. @Html.HiddenFor(model => item.Empid)
25. <tr>
27. <td>
28. @Html.DisplayFor(modelItem => item.Name)
29. </td>
30. <td>
31. @Html.DisplayFor(modelItem => item.City)
32. </td>
33. <td>
34. @Html.DisplayFor(modelItem => item.Address)
35. </td>
36. <td>
37. @Html.ActionLink("Edit", "EditEmpDetails", **new** { id = item.Empid }) |
38. @Html.ActionLink("Delete", "DeleteEmp", **new** { id = item.Empid }, **new** { onclick = "return confirm('Are sure wants to delete?');" })
39. </td>
40. </tr>
42. }
44. </table>

**To Update Added Employees**  
  
Follow the same procedure and create EditEmpDetails view to edit the employees. After creating the view the code will be like the following: **EditEmpDetails.cshtml**

1. @model CRUDUsingMVC.Models.EmpModel

4. @**using** (Html.BeginForm())
5. {
6. @Html.AntiForgeryToken()
8. <div **class**="form-horizontal">
9. <h4>Update Employee Details</h4>
10. <hr />
11. <div>
12. @Html.ActionLink("Back to Details", "GetAllEmployees")
13. </div>
14. <hr />
15. @Html.ValidationSummary(**true**, "", **new** { @**class** = "text-danger" })
16. @Html.HiddenFor(model => model.Empid)
18. <div **class**="form-group">
19. @Html.LabelFor(model => model.Name, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
20. <div **class**="col-md-10">
21. @Html.EditorFor(model => model.Name, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
22. @Html.ValidationMessageFor(model => model.Name, "", **new** { @**class** = "text-danger" })
23. </div>
24. </div>
26. <div **class**="form-group">
27. @Html.LabelFor(model => model.City, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
28. <div **class**="col-md-10">
29. @Html.EditorFor(model => model.City, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
30. @Html.ValidationMessageFor(model => model.City, "", **new** { @**class** = "text-danger" })
31. </div>
32. </div>
34. <div **class**="form-group">
35. @Html.LabelFor(model => model.Address, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
36. <div **class**="col-md-10">
37. @Html.EditorFor(model => model.Address, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
38. @Html.ValidationMessageFor(model => model.Address, "", **new** { @**class** = "text-danger" })
39. </div>
40. </div>
42. <div **class**="form-group">
43. <div **class**="col-md-offset-2 col-md-10">
44. <input type="submit" value="Update" **class**="btn btn-default" />
45. </div>
46. </div>
47. </div>
48. }
49. <script src="~/Scripts/jquery-1.10.2.min.js"></script>
50. <script src="~/Scripts/jquery.validate.min.js"></script>
51. <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>

**Step 8 :** Configure Action Link to *Edit*and delete the records as in the following figure:

Graphical user interface, application, Word

Description automatically generated  
  
The above ActionLink I have added inGetAllEmpDetails.CsHtml view because from there we will delete and update the records.

**Step 9:** Configure RouteConfig.cs to set default action as in the following code snippet:

1. **public** **class** RouteConfig
2. {
3. **public** **static** **void** RegisterRoutes(RouteCollection routes)
4. {
5. routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");
7. routes.MapRoute(
8. name: "Default",
9. url: "{controller}/{action}/{id}",
10. defaults: **new** { controller = "Employee", action = "AddEmployee", id = UrlParameter.Optional }
11. );
12. }
13. }

From the above RouteConfig.cs the default action method we have set is AddEmployee. It means that after running the application the AddEmployee view will be executed first.  
  
Now after adding the all model, views and controller our solution explorer will be look like as in the following screenshot:

Graphical user interface

Description automatically generated with medium confidence

**Step 10:**Run the Application

Now run the application the AddEmployee view will appear as:

Graphical user interface, website

Description automatically generated

Click on save button, the model state validation will fire, as per validation we have set into the EmpModel.cs class:

Graphical user interface, website

Description automatically generated

Now enter the details and on clicking save button, the records get added into the database and the following message appears.  
  
Graphical user interface, application

Description automatically generated

Now click on *Back to Employee List* hyperlink, it will be redirected to employee details grid as in the following screenshot:

Graphical user interface, text, application, email

Description automatically generated

Now similar to above screenshot, add another record, then the list will be as in the following screenshot:

Graphical user interface, text, application

Description automatically generated

Now click on *Edit*button of one of the record, then it will be redirected to Edit view as in the following screenshot:

Graphical user interface, application

Description automatically generated

Now click on Update button, on clicking, the records will be updated into the database. Click*Back to Details* hyperlink then it will be redirected to the Employee list table with updated records as in the following screenshot:

Graphical user interface, text, email

Description automatically generated

Now click on delete button for one of the records, then the following confirmation box appears (we have set configuration in ActionLink):

Graphical user interface

Description automatically generated

Now click on *OK* button, then the updated Employee list table will be like the following screenshot:

Graphical user interface, text, application

Description automatically generated  
  
From the preceding examples we have learned how to implement CRUD operations in ASP.NET MVC using ADO.NET.  
  
**Note:**

* Configure the database connection in the web.config file depending on your database server location.
* Download the Zip file of the sample application for a better understanding.
* Since this is a demo, it might not be using proper standards, so improve it depending on your skills
* This application is created completely focusing on beginners.

**Summary**My next article explains the types of controllers in MVC. I hope this article is useful for all readers. If you have any suggestion then please contact me.