Procedure In ASP.NET MVC Of Real-Time Project

**Introduction**

CRUD means create update and delete.

With ASP.NET in MVC, we can also perform CRUD operations using stored procedure.

**Description**

Using a single stored procedure means selecting, updating, inserting and deleting all SQL queries; we can put in one stored procedure and perform this CRUD operation by calling only this single stored procedure.

**Steps to build MVC Application**

**Step 1**

Create MVC Application named “SatyaMvc4Crud”.



**Step 2**

Create a model class file called Customer.cs.

**Code Ref**

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel.DataAnnotations;
4. using System.Linq;
5. using System.Web;
6. namespace MVC4crud.Models
7. {
8. **public** **class** Customer
9. {
10. [Key]
11. **public** **int** CustomerID { get; set; }
13. [Required(ErrorMessage = "Enter Your Name")]
14. [StringLength(4, ErrorMessage = "Name should be less than or equal to four characters.")]
15. **public** string Name { get; set; }
17. [Required(ErrorMessage = "Enter Your Address")]
18. [StringLength(10, ErrorMessage = "Address should be less than or equal to ten characters.")]
19. **public** string Address { get; set; }
20. [Required(ErrorMessage = "Your must provide a PhoneNumber")]
21. [Display(Name = "Home Phone")]
22. [DataType(DataType.PhoneNumber)]
23. [RegularExpression(@"^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$", ErrorMessage = "Not a valid Phone number")]
24. **public** string Mobileno { get; set; }
25. [DataType(DataType.Date)]
26. [Required(ErrorMessage = "Enter Your DOB.")]
27. [DisplayFormat(DataFormatString = "{0:MM/dd/yyyy}", ApplyFormatInEditMode = **true**)]
28. [MVC4crud.Models.CustomValidationAttributeDemo.ValidBirthDate(ErrorMessage = "Birth Date can not be greater than current date")]
29. **public** DateTime Birthdate { get; set; }
30. [Required(ErrorMessage = "Enter Your EmailID")]
31. [RegularExpression(@"^[\w-\.\_\+%]+@(?:[\w-]+\.)+[\w]{2,6}$", ErrorMessage = "Please enter a valid email address")]
32. **public** string EmailID { get; set; }
34. **public** List<Customer> ShowallCustomer { get; set; }
35. }
37. }

**Code description**

Here, I have created some attributes to check validation of controls, which are based on control values. For textbox name and address validation, I have put [Required(ErrorMessage = "Your Message")].

This code will be executed, if your input is empty in controls.

Now, if the user puts something but this does not satisfy the standard validation, then the code will be, as given below.

[StringLength(4, ErrorMessage = "Name should be less than or equal to four characters.")]

The user can put only four characters to control the input values.

Like this, you can check for other attributes, which are based on control validation values.

Here, I will declare 6 different entities to access the user and inputs. For every entity, I required an attribute to show the validation message failed for the end users.  
  
**e.g**. [Required(ErrorMessage = "Enter Your Name")]  
  
Like this required attribute, I used StringLength, Display, DisplayFormat, RegularExpression attributes.  
  
We have used some attributes. For this, we have to add one namespace.

using System.ComponentModel.DataAnnotations;

In name part, I can enter up to 4 characters.  
  
*[StringLength(4, ErrorMessage = "Name should be less than or equal to four characters.")]*  
  
In address part, I can enter up to 10 characters.  
  
*[StringLength(10, ErrorMessage = "Address should be less than or equal to ten characters.")]*

In MobileNo. part, I can enter only 10 digit valid phone no.

1. [DataType(DataType.PhoneNumber)]
2. [RegularExpression(@"^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$", ErrorMessage = "Not a valid Phone number")]

In an E-mail Id part, I can enter only a valid E-mail Id with @ symbol.

1. [RegularExpression(@"^[\w-\.\_\+%]+@(?:[\w-]+\.)+[\w]{2,6}$", ErrorMessage = "Please enter a valid email address")]

In Date Time part, I can enter only valid date, which should be less than current date.

1. [DisplayFormat(DataFormatString = "{0:MM/dd/yyyy}", ApplyFormatInEditMode = **true**)]
2. [val.Models.UserCustomValidation.ValidBirthDate(ErrorMessage = "Birth Date can not be greater than current date")]

**For Custom validation of the entities, Code Ref is given below.**

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel.DataAnnotations;
4. using System.Linq;
5. using System.Web;
7. namespace MVC4crud.Models
8. {
9. **public** **class** CustomValidationAttributeDemo
10. {
11. [AttributeUsage(AttributeTargets.Property, AllowMultiple = **false**, Inherited = **true**)]
12. **public** sealed **class** ValidBirthDate : ValidationAttribute
13. {
14. **protected** override ValidationResult IsValid(object value, ValidationContext validationContext)
15. {
16. **if** (value != **null**)
17. {
18. DateTime \_birthJoin = Convert.ToDateTime(value);
19. **if** (\_birthJoin > DateTime.Now)
20. {
21. **return** **new** ValidationResult("Birth date can not be greater than current date.");
22. }
23. }
24. **return** ValidationResult.Success;
25. }
26. }
27. }
28. }

**Code description**

Here, I have used one Custom Validation class to customize your Date time validation. For this, I created one class file in Models folder named “CustomValidationAttributeDemo.cs” .

Here, I used one date time variable to access date time .

1. DateTime \_birthJoin = Convert.ToDateTime(value);

Thus, I put some code to take the user’s birth date. Birth date should always be less than today’s date.

1. **if** (\_birthJoin > DateTime.Now)
2. {
3. **return** **new** ValidationResult("Birth date can not be greater than current date.");
4. }

Here, I used one class,  ValidBirthDate, that is inherited from ValidationAttribute class.

1. **public** sealed **class** ValidBirthDate : ValidationAttribute

**What is ValidationAttribute class?**  
It serves as a base class for all the validation attributes.  
  
Go to the definition of this ValidationAttribute class.

Here, System.ComponentModel.DataAnnotations.dll file references for this class files.

#region Assembly System.ComponentModel.DataAnnotations.dll, v4.0.0.0  // C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\System.ComponentModel.DataAnnotations.dll  #endregion

The ValidationResult override method is used to represent a container for the result of the validation request. The ValidationContext class acts as a parameter inside ValidationResult override method. It is used to describe a context in which validation check is performed.

1. **protected** override ValidationResult IsValid(object value, ValidationContext validationContext)
2. **return** **new** ValidationResult("Birth date can not be greater than current date.");
3. **return** ValidationResult.Success;

In custom validation class, I used Student.cs for DATE TIME entity. According to this, the user input date should be less than today’s date.

1. [val.Models.CustomValidationAttributeDemo.ValidBirthDate(ErrorMessage = "Birth Date can not be greater than current date")]

Here, CustomValidationAttributeDemo is the name of the Model Class and ValidBirthDate class is the child class of ValidateAttribute base class.

  
**Step 3**

Create a table to add stored procedure to perform CRUD operation in MVC.

**Table SQL Ref**

1. **CREATE** **TABLE** [dbo].[Customer]
2. (
3. [CustomerID] [**int**] NOT NULL **Primary** **key** identity(1,1),
4. [**Name**] [**varchar**](100) NULL,
5. [Address] [**varchar**](300) NULL,
6. [Mobileno] [**varchar**](15) NULL,
7. [Birthdate] [datetime] NULL,
8. [EmailID] [**varchar**](300) NULL
9. )

**Table SQL Description**

Here, six columns are same as entities declared in models class  “Customer.cs”.

Here “[CustomerID]” is the primary key and auto increment feature is added in these columns.

**Step 4**

Now, create a  stored procedure to perform CRUD operation in MVC.

**Stored procedure Script Ref**

1. USE [database\_name]
2. GO
3. **SET** ANSI\_NULLS **ON**
4. GO
5. **SET** QUOTED\_IDENTIFIER **ON**
6. GO
7. **CREATE** **PROCEDURE** [dbo].[Usp\_InsertUpdateDelete\_Customer]
8. @CustomerID **INT** = NULL
9. ,@**Name** NVARCHAR(100) = NULL
10. ,@Mobileno NVARCHAR(15) = NULL
11. ,@Address NVARCHAR(300) = 0
12. ,@Birthdate DATETIME = NULL
13. ,@EmailID NVARCHAR(15) = NULL
14. ,@Query **INT**
15. **AS**
16. **BEGIN**
17. IF (@Query = 1)
18. **BEGIN**


22. **INSERT** **INTO** Customer(
23. **NAME**
24. ,Address
25. ,Mobileno
26. ,Birthdate
27. ,EmailID
28. )
29. **VALUES** (
30. @**Name**
31. ,@Address
32. ,@Mobileno
33. ,@Birthdate
34. ,@EmailID
35. )
36. IF (@@ROWCOUNT > 0)
37. **BEGIN**
38. **SELECT** 'Insert'
39. **END**
40. **END**
41. IF (@Query = 2)
42. **BEGIN**
43. **UPDATE** Customer
44. **SET** **NAME** = @**Name**
45. ,Address = @Address
46. ,Mobileno = @Mobileno
47. ,Birthdate = @Birthdate
48. ,EmailID = @EmailID
49. **WHERE** Customer.CustomerID = @CustomerID
50. **SELECT** 'Update'
51. **END**
52. IF (@Query = 3)
53. **BEGIN**
54. **DELETE**
55. **FROM** Customer
56. **WHERE** Customer.CustomerID = @CustomerID
57. **SELECT** 'Deleted'
58. **END**
59. IF (@Query = 4)
60. **BEGIN**
61. **SELECT** \*
62. **FROM** Customer
63. **END**
64. **END**
65. IF (@Query = 5)
66. **BEGIN**
67. **SELECT** \*
68. **FROM** Customer
69. **WHERE** Customer.CustomerID = @CustomerID
70. **END**

**Stored procedure Script description**

Here, five @Query parameters with different value for Insert/ Update/ Delete/ Select/Search statement.

Here (@Query = 1) means for insertion of the records.

1. F (@Query = 1)
2. **BEGIN**
3. **INSERT** **INTO** Customer(
4. **NAME**
5. ,Address
6. ,Mobileno
7. ,Birthdate
8. ,EmailID
9. )
10. **VALUES** (
11. @**Name**
12. ,@Address
13. ,@Mobileno
14. ,@Birthdate
15. ,@EmailID
16. )

Like this, other (@Query = 2 to 5) is assigned for other operation performances.

By using single procedure by using this individual query parameter values; we can perform different operations.



**Step 5**

Create a class file called DataAccessLayer.cs inside manually created folder named DataAccess.

To add connection string, add name as well as stored procedure name to perform CRUD role.

**Code Ref**

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using MVC4crud.Models;
6. using System.Data.SqlClient;
7. using System.Data;
8. using System.Configuration;
9. namespace MVC4crud.DataAccess
10. {
11. **public** **class** DataAccessLayer
12. {
13. **public** string InsertData(Customer objcust)
14. {
15. SqlConnection con = **null**;
17. string result = "";
18. **try**
19. {
20. con = **new** SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
21. SqlCommand cmd = **new** SqlCommand("Usp\_InsertUpdateDelete\_Customer", con);
22. cmd.CommandType = CommandType.StoredProcedure;
23. //cmd.Parameters.AddWithValue("@CustomerID", 0);
24. cmd.Parameters.AddWithValue("@Name", objcust.Name);
25. cmd.Parameters.AddWithValue("@Address", objcust.Address);
26. cmd.Parameters.AddWithValue("@Mobileno", objcust.Mobileno);
27. cmd.Parameters.AddWithValue("@Birthdate", objcust.Birthdate);
28. cmd.Parameters.AddWithValue("@EmailID", objcust.EmailID);
29. cmd.Parameters.AddWithValue("@Query", 1);
30. con.Open();
31. result = cmd.ExecuteScalar().ToString();
32. **return** result;
33. }
34. **catch**
35. {
36. **return** result = "";
37. }
38. **finally**
39. {
40. con.Close();
41. }
42. }
43. **public** string UpdateData(Customer objcust)
44. {
45. SqlConnection con = **null**;
46. string result = "";
47. **try**
48. {
49. con = **new** SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
50. SqlCommand cmd = **new** SqlCommand("Usp\_InsertUpdateDelete\_Customer", con);
51. cmd.CommandType = CommandType.StoredProcedure;
52. cmd.Parameters.AddWithValue("@CustomerID", objcust.CustomerID);
53. cmd.Parameters.AddWithValue("@Name", objcust.Name);
54. cmd.Parameters.AddWithValue("@Address", objcust.Address);
55. cmd.Parameters.AddWithValue("@Mobileno", objcust.Mobileno);
56. cmd.Parameters.AddWithValue("@Birthdate", objcust.Birthdate);
57. cmd.Parameters.AddWithValue("@EmailID", objcust.EmailID);
58. cmd.Parameters.AddWithValue("@Query", 2);
59. con.Open();
60. result = cmd.ExecuteScalar().ToString();
61. **return** result;
62. }
63. **catch**
64. {
65. **return** result = "";
66. }
67. **finally**
68. {
69. con.Close();
70. }
71. }
72. **public** **int** DeleteData(String ID)
73. {
74. SqlConnection con = **null**;
75. **int** result;
76. **try**
77. {
78. con = **new** SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
79. SqlCommand cmd = **new** SqlCommand("Usp\_InsertUpdateDelete\_Customer", con);
80. cmd.CommandType = CommandType.StoredProcedure;
81. cmd.Parameters.AddWithValue("@CustomerID", ID);
82. cmd.Parameters.AddWithValue("@Name", **null**);
83. cmd.Parameters.AddWithValue("@Address", **null**);
84. cmd.Parameters.AddWithValue("@Mobileno", **null**);
85. cmd.Parameters.AddWithValue("@Birthdate", **null**);
86. cmd.Parameters.AddWithValue("@EmailID", **null**);
87. cmd.Parameters.AddWithValue("@Query", 3);
88. con.Open();
89. result = cmd.ExecuteNonQuery();
90. **return** result;
91. }
92. **catch**
93. {
94. **return** result = 0;
95. }
96. **finally**
97. {
98. con.Close();
99. }
100. }
101. **public** List<Customer> Selectalldata()
102. {
103. SqlConnection con = **null**;
104. DataSet ds = **null**;
105. List<Customer> custlist = **null**;
106. **try**
107. {
108. con = **new** SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
109. SqlCommand cmd = **new** SqlCommand("Usp\_InsertUpdateDelete\_Customer", con);
110. cmd.CommandType = CommandType.StoredProcedure;
111. cmd.Parameters.AddWithValue("@CustomerID", **null**);
112. cmd.Parameters.AddWithValue("@Name", **null**);
113. cmd.Parameters.AddWithValue("@Address", **null**);
114. cmd.Parameters.AddWithValue("@Mobileno", **null**);
115. cmd.Parameters.AddWithValue("@Birthdate", **null**);
116. cmd.Parameters.AddWithValue("@EmailID", **null**);
117. cmd.Parameters.AddWithValue("@Query", 4);
118. con.Open();
119. SqlDataAdapter da = **new** SqlDataAdapter();
120. da.SelectCommand = cmd;
121. ds = **new** DataSet();
122. da.Fill(ds);
123. custlist = **new** List<Customer>();
124. **for** (**int** i = 0; i < ds.Tables[0].Rows.Count; i++)
125. {
126. Customer cobj = **new** Customer();
127. cobj.CustomerID = Convert.ToInt32(ds.Tables[0].Rows[i]["CustomerID"].ToString());
128. cobj.Name = ds.Tables[0].Rows[i]["Name"].ToString();
129. cobj.Address = ds.Tables[0].Rows[i]["Address"].ToString();
130. cobj.Mobileno = ds.Tables[0].Rows[i]["Mobileno"].ToString();
131. cobj.EmailID = ds.Tables[0].Rows[i]["EmailID"].ToString();
132. cobj.Birthdate = Convert.ToDateTime(ds.Tables[0].Rows[i]["Birthdate"].ToString());
134. custlist.Add(cobj);
135. }
136. **return** custlist;
137. }
138. **catch**
139. {
140. **return** custlist;
141. }
142. **finally**
143. {
144. con.Close();
145. }
146. }
148. **public** Customer SelectDatabyID(string CustomerID)
149. {
150. SqlConnection con = **null**;
151. DataSet ds = **null**;
152. Customer cobj = **null**;
153. **try**
154. {
155. con = **new** SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
156. SqlCommand cmd = **new** SqlCommand("Usp\_InsertUpdateDelete\_Customer", con);
157. cmd.CommandType = CommandType.StoredProcedure;
158. cmd.Parameters.AddWithValue("@CustomerID", CustomerID);
159. cmd.Parameters.AddWithValue("@Name", **null**);
160. cmd.Parameters.AddWithValue("@Address", **null**);
161. cmd.Parameters.AddWithValue("@Mobileno", **null**);
162. cmd.Parameters.AddWithValue("@Birthdate", **null**);
163. cmd.Parameters.AddWithValue("@EmailID", **null**);
164. cmd.Parameters.AddWithValue("@Query", 5);
165. SqlDataAdapter da = **new** SqlDataAdapter();
166. da.SelectCommand = cmd;
167. ds = **new** DataSet();
168. da.Fill(ds);
169. **for** (**int** i = 0; i < ds.Tables[0].Rows.Count; i++)
170. {
171. cobj = **new** Customer();
172. cobj.CustomerID = Convert.ToInt32(ds.Tables[0].Rows[i]["CustomerID"].ToString());
173. cobj.Name = ds.Tables[0].Rows[i]["Name"].ToString();
174. cobj.Address = ds.Tables[0].Rows[i]["Address"].ToString();
175. cobj.Mobileno = ds.Tables[0].Rows[i]["Mobileno"].ToString();
176. cobj.EmailID = ds.Tables[0].Rows[i]["EmailID"].ToString();
177. cobj.Birthdate = Convert.ToDateTime(ds.Tables[0].Rows[i]["Birthdate"].ToString());
179. }
180. **return** cobj;
181. }
182. **catch**
183. {
184. **return** cobj;
185. }
186. **finally**
187. {
188. con.Close();
189. }
190. }
191. }
192. }

**Code description**

Here, I will show how to implement query parameter value, which is 1 to implement in this class file to perform insert operation.

1. **public** string InsertData(Customer objcust)
2. {
3. SqlConnection con = **null**;
5. string result = "";
6. **try**
7. {
8. con = **new** SqlConnection(ConfigurationManager.ConnectionStrings["mycon"].ToString());
9. SqlCommand cmd = **new** SqlCommand("Usp\_InsertUpdateDelete\_Customer", con);
10. cmd.CommandType = CommandType.StoredProcedure;
11. //cmd.Parameters.AddWithValue("@CustomerID", 0);
12. cmd.Parameters.AddWithValue("@Name", objcust.Name);
13. cmd.Parameters.AddWithValue("@Address", objcust.Address);
14. cmd.Parameters.AddWithValue("@Mobileno", objcust.Mobileno);
15. cmd.Parameters.AddWithValue("@Birthdate", objcust.Birthdate);
16. cmd.Parameters.AddWithValue("@EmailID", objcust.EmailID);
17. cmd.Parameters.AddWithValue("@Query", 1);
18. con.Open();
19. result = cmd.ExecuteScalar().ToString();
20. **return** result;
21. }
22. **catch**
23. {
24. **return** result = "";
25. }
26. **finally**
27. {
28. con.Close();
29. }
30. }

In this InsertData() function, I used @Query = 1 value to perform insert operation. Here, I have added stored procedure name.

1. SqlCommand cmd = new SqlCommand("Usp\_InsertUpdateDelete\_Customer", con);

To perform update operation, I have added the code in UpdateData function.

1. cmd.Parameters.AddWithValue("@Query", 2);

To perform delete operation, I have added the code in DeleteData function.

1. cmd.Parameters.AddWithValue("@Query", 3);

To perform select list of data, I have added the code in Selectalldata list function of customer model class.

1. cmd.Parameters.AddWithValue("@Query", 4);

I have added one for loop to perform selection of all the data loop wise by using customer model class.

1. **for** (**int** i = 0; i < ds.Tables[0].Rows.Count; i++)
2. {
3. cobj = **new** Customer();
4. cobj.CustomerID = Convert.ToInt32(ds.Tables[0].Rows[i]["CustomerID"].ToString());
5. cobj.Name = ds.Tables[0].Rows[i]["Name"].ToString();
6. cobj.Address = ds.Tables[0].Rows[i]["Address"].ToString();
7. cobj.Mobileno = ds.Tables[0].Rows[i]["Mobileno"].ToString();
8. cobj.EmailID = ds.Tables[0].Rows[i]["EmailID"].ToString();
9. cobj.Birthdate = Convert.ToDateTime(ds.Tables[0].Rows[i]["Birthdate"].ToString());
11. }

Now, I will filter the records by using customer Id values.

I have added the code given below.

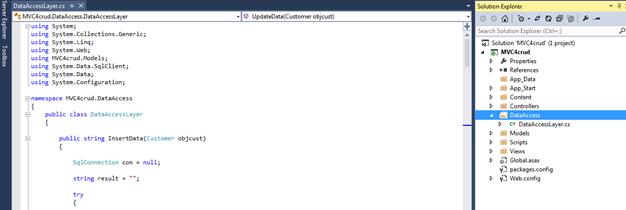
1. cmd.Parameters.AddWithValue("@Query", 5);

Thus, I have added customer Id parameter in a function of customer model class.

1. **public** Customer SelectDatabyID(string CustomerID)

Now, I have closed the connection in every functions in this class by using catch and finally block.

1. **finally**
2. {
3. con.Close();
4. }



**Step 6**

Create a controller class file called CustomerController.cs.

**Code Ref**

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Web.Mvc;
6. using MVC4crud.Models;
7. using MVC4crud.DataAccess;
9. namespace MVC4crud.Controllers
10. {
11. **public** **class** CustomerController : Controller
12. {
13. //
14. // GET: /Customer/
15. [HttpGet]
16. **public** ActionResult InsertCustomer()
17. {
18. **return** View();
19. }
20. [HttpPost]
21. **public** ActionResult InsertCustomer(Customer objCustomer)
22. {
24. objCustomer.Birthdate = Convert.ToDateTime(objCustomer.Birthdate);
25. **if** (ModelState.IsValid) //checking model is valid or not
26. {
27. DataAccessLayer objDB = **new** DataAccessLayer();
28. string result = objDB.InsertData(objCustomer);
29. //ViewData["result"] = result;
30. TempData["result1"] = result;
31. ModelState.Clear(); //clearing model
32. //return View();
33. **return** RedirectToAction("ShowAllCustomerDetails");
34. }
36. **else**
37. {
38. ModelState.AddModelError("", "Error in saving data");
39. **return** View();
40. }
41. }
43. [HttpGet]
44. **public** ActionResult ShowAllCustomerDetails()
45. {
46. Customer objCustomer = **new** Customer();
47. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
48. objCustomer.ShowallCustomer = objDB.Selectalldata();
49. **return** View(objCustomer);
50. }
51. [HttpGet]
52. **public** ActionResult Details(string ID)
53. {
54. //Customer objCustomer = new Customer();
55. //DataAccessLayer objDB = new DataAccessLayer(); //calling class DBdata
56. //objCustomer.ShowallCustomer = objDB.Selectalldata();
57. //return View(objCustomer);
58. Customer objCustomer = **new** Customer();
59. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
60. **return** View(objDB.SelectDatabyID(ID));
61. }
62. [HttpGet]
63. **public** ActionResult Edit(string ID)
64. {
65. Customer objCustomer = **new** Customer();
66. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
67. **return** View(objDB.SelectDatabyID(ID));
68. }
70. [HttpPost]
71. **public** ActionResult Edit(Customer objCustomer)
72. {
73. objCustomer.Birthdate = Convert.ToDateTime(objCustomer.Birthdate);
74. **if** (ModelState.IsValid) //checking model is valid or not
75. {
76. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
77. string result = objDB.UpdateData(objCustomer);
78. //ViewData["result"] = result;
79. TempData["result2"] = result;
80. ModelState.Clear(); //clearing model
81. //return View();
82. **return** RedirectToAction("ShowAllCustomerDetails");
83. }
84. **else**
85. {
86. ModelState.AddModelError("", "Error in saving data");
87. **return** View();
88. }
89. }
91. [HttpGet]
92. **public** ActionResult Delete(String ID)
93. {
94. DataAccessLayer objDB = **new** DataAccessLayer();
95. **int** result = objDB.DeleteData(ID);
96. TempData["result3"] = result;
97. ModelState.Clear(); //clearing model
98. //return View();
99. **return** RedirectToAction("ShowAllCustomerDetails");
100. }
101. }
102. }

**Code description**

In this controller class file, I have created one controller action method.

1. **public** ActionResult InsertCustomer(Customer objCustomer)
2. {
4. objCustomer.Birthdate = Convert.ToDateTime(objCustomer.Birthdate);
5. **if** (ModelState.IsValid)
6. {
7. DataAccessLayer objDB = **new** DataAccessLayer();
8. string result = objDB.InsertData(objCustomer);
9. //ViewData["result"] = result;
10. TempData["result1"] = result;
11. ModelState.Clear();                    //return View();
12. **return** RedirectToAction("ShowAllCustomerDetails");
13. }
15. **else**
16. {
17. ModelState.AddModelError("", "Error in saving data");
18. **return** View();
19. }
20. }

Here, DataAccessLayer is added as a reference to use all its methods.

1. DataAccessLayer objDB = **new** DataAccessLayer();
2. string result = objDB.InsertData(objCustomer);

The code is given below for checking model is valid or not.

1. **if** (ModelState.IsValid)
2. **for**  “clearing model”
4. ModelState.Clear();

Here, if the condition satisfies, then model state will be valid, else the data will not save properly.

1. **else**
2. {
3. ModelState.AddModelError("", "Error in saving data");
4. **return** View();
5. }

Here, I added the show details view page name as soon as the data is inserted successfully, the page will show you the list of inserted data.

1. **return** RedirectToAction("ShowAllCustomerDetails");

In ShowAllCustomerDetails action result method, the Selectalldata of DataAccessLayer class is used.

1. [HttpGet]
2. **public** ActionResult ShowAllCustomerDetails()
3. {
4. Customer objCustomer = **new** Customer();
5. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
6. objCustomer.ShowallCustomer = objDB.Selectalldata();
7. **return** View(objCustomer);
8. }

In Details action result method, the SelectDatabyID of DataAccessLayer class is used.

The passed parameter value with the related data will be shown in the corresponding view page.

1. [HttpGet]
2. **public** ActionResult Details(string ID)
3. {
4. Customer objCustomer = **new** Customer();
5. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
6. **return** View(objDB.SelectDatabyID(ID));
7. }

The edit action result method has two attributes httpget and httppost.

For httpget attribute In “Edit” action result method, the “SelectDatabyID” of DataAccessLayer class is used to step update data by using this particular data.

The passed parameter value with the related data will be shown in the corresponding view page.

1. [HttpGet]
2. **public** ActionResult Edit(string ID)
3. {
4. Customer objCustomer = **new** Customer();
5. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
6. **return** View(objDB.SelectDatabyID(ID));
7. }

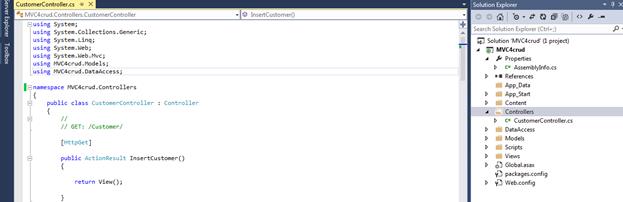
For httppost attribute, the edit controller action method takes customer model class object and UpdateData of DataAccessLayer class is  used.

1. [HttpPost]
2. **public** ActionResult Edit(Customer objCustomer)
3. {
4. objCustomer.Birthdate = Convert.ToDateTime(objCustomer.Birthdate);
5. **if** (ModelState.IsValid) //checking model is valid or not
6. {
7. DataAccessLayer objDB = **new** DataAccessLayer(); //calling class DBdata
8. string result = objDB.UpdateData(objCustomer);
9. //ViewData["result"] = result;
10. TempData["result2"] = result;
11. ModelState.Clear(); //clearing model
12. //return View();
13. **return** RedirectToAction("ShowAllCustomerDetails");
14. }
15. **else**
16. {
17. ModelState.AddModelError("", "Error in saving data");
18. **return** View();
19. }
20. }

In Delete action result method, the DeleteData of DataAccessLayer class is used.

The passed parameter value with the related data will show to perform delete operation in the corresponding view page.

1. [HttpGet]
2. **public** ActionResult Delete(String ID)
3. {
4. DataAccessLayer objDB = **new** DataAccessLayer();
5. **int** result = objDB.DeleteData(ID);
6. TempData["result3"] = result;
7. ModelState.Clear(); //clearing model
8. //return View();
9. **return** RedirectToAction("ShowAllCustomerDetails");
10. }



**Step7**

Now, create view cshtml file called ShowAllCustomerDetails.cshtml, InsertCustomer.cshtml.

,” Edit.cshtml”,” Details.cshtml”,” Delete.cshtml”.

**Code ref. of InsertCustomer.cshtml**

1. @model MVC4crud.Models.Customer
2. @{
3. Layout = **null**;
4. }
5. <!DOCTYPE html>
6. <html>
7. <head>
8. <meta name="viewport" content="width=device-width" />
9. <title>InsertCustomer</title>
10. </head>
11. <body>
12. <script src="~/Scripts/jquery-1.7.1.min.js"></script>
13. <script src="~/Scripts/jquery.validate.min.js"></script>
14. <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
15. @using (Html.BeginForm())
16. {
18. @Html.ValidationSummary(**true**)
19. <fieldset>
20. <legend style="font-family:Arial Black;color:Green">Customer</legend>
21. <div **class**="editor-label" style="font-family:Arial Black">
22. @Html.LabelFor(model => model.Name)
23. </div>
24. <div **class**="editor-field" style="color:Red;font-family:Arial">
25. @Html.EditorFor(model => model.Name)
26. @Html.ValidationMessageFor(model => model.Name)
27. </div>
28. <div **class**="editor-label" style="font-family:Arial Black">
29. @Html.LabelFor(model => model.Address)
30. </div>
31. <div **class**="editor-field" style="color:Red;font-family:Arial">
32. @Html.EditorFor(model => model.Address)
33. @Html.ValidationMessageFor(model => model.Address)
34. </div>
35. <div **class**="editor-label" style="font-family:Arial Black">
36. @Html.LabelFor(model => model.Mobileno)
37. </div>
38. <div **class**="editor-field" style="color:Red;font-family:Arial">
39. @Html.EditorFor(model => model.Mobileno)
40. @Html.ValidationMessageFor(model => model.Mobileno)
41. </div>
42. <div **class**="editor-label" style="font-family:Arial Black">
43. @Html.LabelFor(model => model.Birthdate)
44. </div>
45. <div **class**="editor-field" style="color:Red;font-family:Arial">
46. @Html.EditorFor(model => model.Birthdate)
47. @Html.ValidationMessageFor(model => model.Birthdate)
48. </div>
49. <div **class**="editor-label" style="font-family:Arial Black">
50. @Html.LabelFor(model => model.EmailID)
51. </div>
52. <div **class**="editor-field" style="color:Red;font-family:Arial">
53. @Html.EditorFor(model => model.EmailID)
54. @Html.ValidationMessageFor(model => model.EmailID)
55. </div>
56. <p>
57. <input type="submit" value="Insert" style="color:Navy;font-family:Arial; font-size:large" />
58. <input type="reset" value="Reset" style="color:Navy;font-family:Arial; font-size:large" />
59. </p>
60. </fieldset>
61. }
62. <div style="color:Blue;font-family:Arial">
63. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")
64. </div>
65. </body>
66. </html>

**Code description**

In this view page, I have added customer class reference or namespace.

1. @model MVC4crud.Models.Customer

To make validation summary active, use this code.

1. @Html.ValidationSummary(**true**)

Now, I have added some code to make textbox and label control according to customer model class entities.

1. <div **class**="editor-label" style="font-family:Arial Black">
2. @Html.LabelFor(model => model.Name)
3. </div>
4. <div **class**="editor-field" style="color:Red;font-family:Arial">
5. @Html.EditorFor(model => model.Name)
6. @Html.ValidationMessageFor(model => model.Name)
7. </div>
8. <div **class**="editor-label" style="font-family:Arial Black">
9. @Html.LabelFor(model => model.Address)
10. </div>
11. <div **class**="editor-field" style="color:Red;font-family:Arial">
12. @Html.EditorFor(model => model.Address)
13. @Html.ValidationMessageFor(model => model.Address)
14. </div>
15. <div **class**="editor-label" style="font-family:Arial Black">
16. @Html.LabelFor(model => model.Mobileno)
17. </div>
18. <div **class**="editor-field" style="color:Red;font-family:Arial">
19. @Html.EditorFor(model => model.Mobileno)
20. @Html.ValidationMessageFor(model => model.Mobileno)
21. </div>
22. <div **class**="editor-label" style="font-family:Arial Black">
23. @Html.LabelFor(model => model.Birthdate)
24. </div>
25. <div **class**="editor-field" style="color:Red;font-family:Arial">
26. @Html.EditorFor(model => model.Birthdate)
27. @Html.ValidationMessageFor(model => model.Birthdate)
28. </div>
29. <div **class**="editor-label" style="font-family:Arial Black">
30. @Html.LabelFor(model => model.EmailID)
31. </div>
32. <div **class**="editor-field" style="color:Red;font-family:Arial">
33. @Html.EditorFor(model => model.EmailID)
34. @Html.ValidationMessageFor(model => model.EmailID)
35. </div>

To add label control, the code is given below.

*@Html.LabelFor()*

To add textbox control, the code is given below.

*@Html.EditorFor()*

To add validation messages, as defined in customer model class and customized model validation class, the code is given below.

*@Html.ValidationMessageFor()*

Here, two types of buttons are used to save the data.

1. <input type="submit" value="Insert" style="color:Navy;font-family:Arial; font-size:large" />

Here, two types of buttons are used; where one is to reset the data.

1. <input type="reset" value="Reset" style="color:Navy;font-family:Arial; font-size:large" />

After saving data, the details view page will come with all the saved data.

1. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")

Here, ShowAllCustomerDetails is the name of the controller action method as well as view name.

Here, I have added one hyperlink to redirect to other page.

1. @Html.ActionLink()

**Code ref. of Edit.cshtml**

1. @model MVC4crud.Models.Customer
2. @{
3. Layout = **null**;
4. }
5. <!DOCTYPE html>
6. <html>
7. <head>
8. <meta name="viewport" content="width=device-width" />
10. <title>Edit</title>
11. </head>
12. <body>
13. <script src="~/Scripts/jquery-1.7.1.min.js"></script>
14. <script src="~/Scripts/jquery.validate.min.js"></script>
15. <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
16. @using (Html.BeginForm())
17. {
18. @Html.ValidationSummary(**true**)
19. <fieldset>
20. <legend style="font-family:Arial Black;color:Green">Customer</legend>
21. @Html.HiddenFor(model => model.CustomerID)
22. <div **class**="editor-label" style="font-family:Arial Black">
23. @Html.LabelFor(model => model.Name)
24. </div>
25. <div **class**="editor-field" style="color:Red;font-family:Arial">
26. @Html.EditorFor(model => model.Name)
27. @Html.ValidationMessageFor(model => model.Name)
28. </div>
29. <div **class**="editor-label" style="font-family:Arial Black">
30. @Html.LabelFor(model => model.Address)
31. </div>
32. <div **class**="editor-field" style="color:Red;font-family:Arial">
33. @Html.EditorFor(model => model.Address)
34. @Html.ValidationMessageFor(model => model.Address)
35. </div>
36. <div **class**="editor-label" style="font-family:Arial Black">
37. @Html.LabelFor(model => model.Mobileno)
38. </div>
39. <div **class**="editor-field" style="color:Red;font-family:Arial">
40. @Html.EditorFor(model => model.Mobileno)
41. @Html.ValidationMessageFor(model => model.Mobileno)
42. </div>
43. <div **class**="editor-label" style="font-family:Arial Black">
44. @Html.LabelFor(model => model.Birthdate)
45. </div>
46. <div **class**="editor-field" style="color:Red;font-family:Arial">
47. @Html.EditorFor(model => model.Birthdate)
48. @Html.ValidationMessageFor(model => model.Birthdate)
49. </div>
50. <div **class**="editor-label" style="font-family:Arial Black">
51. @Html.LabelFor(model => model.EmailID)
52. </div>
53. <div **class**="editor-field" style="color:Red;font-family:Arial">
54. @Html.EditorFor(model => model.EmailID)
55. @Html.ValidationMessageFor(model => model.EmailID)
56. </div>
57. <p>
58. <input type="submit" value="Save" style="color:Navy;font-family:Arial; font-size:large" />
59. </p>
60. </fieldset>
61. }
62. <div style="color:Blue;font-family:Arial">
63. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")
64. </div>
65. </body>
66. </html>

**Code description**

In this view pag,e the editor and label controls will bound existing data to update it. Hence, the data as in insert view page will load in corresponding HTML helper control and the user will update, as per requirement.

After update process completes, the view details page will come.

1. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")

Here, I used multiple submit button for different actions.

1. <input type="submit" value="Save" style="color:Navy;font-family:Arial; font-size:large"/>

Here, I have added hidden field control associate with customerid to perform an edit operation.

1. @Html.HiddenFor(model => model.CustomerID)

**Code ref. of  Details.cshtml**

1. @model MVC4crud.Models.Customer
2. @{
3. Layout = **null**;
4. }
6. <!DOCTYPE html>
8. <html>
9. <head>
10. <meta name="viewport" content="width=device-width" />
11. <title>Details</title>
12. </head>
13. <body>
14. <fieldset>
15. <legend style="font-family:Arial Black;color:Green">Customer Report</legend>
17. <div **class**="display-label" style="font-family:Arial Black">
18. @Html.DisplayNameFor(model => model.Name)
19. </div>
20. <div **class**="display-field" style="color:Blue">
21. @Html.DisplayFor(model => model.Name)
22. </div>
24. <div **class**="display-label" style="font-family:Arial Black">
25. @Html.DisplayNameFor(model => model.Address)
26. </div>
27. <div **class**="display-field" style="color:Blue">
28. @Html.DisplayFor(model => model.Address)
29. </div>
31. <div **class**="display-label" style="font-family:Arial Black">
32. @Html.DisplayNameFor(model => model.Mobileno)
33. </div>
34. <div **class**="display-field" style="color:Blue">
35. @Html.DisplayFor(model => model.Mobileno)
36. </div>
38. <div **class**="display-label" style="font-family:Arial Black">
39. @Html.DisplayNameFor(model => model.Birthdate)
40. </div>
41. <div **class**="display-field" style="color:Blue">
42. @Html.DisplayFor(model => model.Birthdate)
43. </div>
45. <div **class**="display-label" style="font-family:Arial Black">
46. @Html.DisplayNameFor(model => model.EmailID)
47. </div>
48. <div **class**="display-field" style="color:Blue">
49. @Html.DisplayFor(model => model.EmailID)
50. </div>
51. </fieldset>
52. <p style="color:Blue;font-family:Arial">
53. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")
54. </p>
55. </body>
56. </html>

**Code description**

In this view page, the data inserted and updated data will show for reporting purpose.

Here, no submit button is required to take action in page event.

Afterwards, the page will redirect to view details page.

1. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")

**Code ref. of  Delete.cshtml**

1. @model MVC4crud.Models.Customer
2. @{
3. Layout = **null**;
4. }
5. <!DOCTYPE html>
6. <html>
7. <head>
8. <meta name="viewport" content="width=device-width" />
9. <title>Delete</title>
10. </head>
11. <body>
12. <h3 style="color:Red">Do You Want To Remove This Record?</h3>
13. @using (Html.BeginForm())
14. {
15. <fieldset>
16. <legend style="font-family:Arial Black;color:Green">Customer</legend>
17. @Html.HiddenFor(model => model.CustomerID)
18. <div **class**="display-label" style="font-family:Arial Black">
19. @Html.DisplayNameFor(model => model.Name)
20. </div>
21. <div **class**="display-field" style="color:Blue">
22. @Html.DisplayFor(model => model.Name)
23. </div>
24. <div **class**="display-label" style="font-family:Arial Black">
25. @Html.DisplayNameFor(model => model.Address)
26. </div>
27. <div **class**="display-field" style="color:Blue">
28. @Html.DisplayFor(model => model.Address)
29. </div>
30. <div **class**="display-label" style="font-family:Arial Black">
31. @Html.DisplayNameFor(model => model.Mobileno)
32. </div>
33. <div **class**="display-field" style="color:Blue">
34. @Html.DisplayFor(model => model.Mobileno)
35. </div>
36. <div **class**="display-label" style="font-family:Arial Black">
37. @Html.DisplayNameFor(model => model.Birthdate)
38. </div>
39. <div **class**="display-field" style="color:Blue">
40. @Html.DisplayFor(model => model.Birthdate)
41. </div>
43. <div **class**="display-label" style="font-family:Arial Black">
44. @Html.DisplayNameFor(model => model.EmailID)
45. </div>
46. <div **class**="display-field" style="color:Blue">
47. @Html.DisplayFor(model => model.EmailID)
48. </div>
49. <p>
50. <input id="Submit1" onclick="return confirm('Are You Sure To Remove ?')"
51. type="submit" value="Remove" style="color:Navy;font-family:Arial; font-size:large" />
52. </p>
53. </fieldset>
54. }
55. <div style="color:Blue;font-family:Arial">
56. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")
57. </div>
58. </body>
59. </html>

**Code description**

In this view page, the delete confirmation text message is added in header.

1. <h3 style="color:Red">Do You Want To Remove This Record?</h3>

In button event, JavaScript message was added  and the user will decide whether it will be deleted or not.

1. <input id="Submit1" onclick="return confirm('Are You Sure To Remove ?')"type="submit" value="Remove" style="color:Navy;font-family:Arial; font-size:large" />

The data is loaded and is based on the selection of customer id and data associated with the entities defined in customer model class, which will be deleted.

Here, I have added hidden field control associate with customerid to perform delete operation.

1. @Html.HiddenFor(model => model.CustomerID)

After this the page will redirect to view details page.

1. @Html.ActionLink("Back to Customer Records", "ShowAllCustomerDetails")

**Code ref. of  ShowAllCustomerDetails.cshtml**

1. @model MVC4crud.Models.Customer
2. @{
3. Layout = **null**;
4. }
5. <!DOCTYPE html>
6. <html>
7. <head>
8. <meta name="viewport" content="width=device-width" />
9. <title>ShowAllCustomerDetails</title>
10. </head>
11. <body>
12. <script src="~/Scripts/jquery-1.7.1.min.js"></script>
13. <script src="~/Scripts/jquery.validate.min.js"></script>
14. <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
15. <div style="font-family:Arial Black;">
16. <h2 style="background-color: Yellow;color: Blue; text-align: center; font-style: oblique">SATYAPRAKASH's MVC Customer CRUD Operation</h2>
17. @\*<h2 style="text-align:center"></h2>\*@
18. <p> @\*<p style="text-align:center">\*@
19. @Html.ActionLink(linkText: "New Customer", actionName: "InsertCustomer", controllerName: "Customer")
20. </p>
21. <br />
22. <br />
23. <table border="1" align="center">
24. <tr>
25. <th style="background-color:Yellow;color:blue">
26. @Html.DisplayNameFor(model => model.Name)
28. </th>
29. <th style="background-color: Yellow;color: blue">
30. @Html.DisplayNameFor(model => model.Address)
31. </th>
32. <th style="background-color: Yellow;color: blue">
33. @Html.DisplayNameFor(model => model.Mobileno)
34. </th>
35. <th style="background-color: Yellow;color: blue">
36. @Html.DisplayNameFor(model => model.Birthdate)
37. </th>
38. <th style="background-color: Yellow;color: blue">
39. @Html.DisplayNameFor(model => model.EmailID)
40. </th>
41. <th style="background-color: Yellow;color: blue">
42. Operation
43. </th>
44. </tr>
46. @foreach (**var** item **in** Model.ShowallCustomer)
47. {
48. <tr>
49. <td>
50. @Html.DisplayFor(modelItem => item.Name)
51. </td>
52. <td>
53. @Html.DisplayFor(modelItem => item.Address)
54. </td>
55. <td>
56. @Html.DisplayFor(modelItem => item.Mobileno)
57. </td>
58. <td>
59. @Html.DisplayFor(modelItem => item.Birthdate)
60. </td>
61. <td>
62. @Html.DisplayFor(modelItem => item.EmailID)
63. </td>
64. <td>
66. @Html.ActionLink("Modify", "Edit", **new** { id = item.CustomerID }) |
68. @Html.ActionLink("Detail", "Details", **new** { id = item.CustomerID }) |
69. @\*@Html.ActionLink("Remove", "Delete", **new** { id = item.CustomerID })\*@
70. @Html.ActionLink("Remove", "Delete", **new** { id = item.CustomerID},
71. **new** { onclick = "return confirm('Are sure wants to delete?');" })
72. </td>
73. </tr>
74. }
75. </table>
76. @**if** (TempData["result1"] != **null**)
77. {
78. <script type="text/javascript">
79. alert("Record Is Inserted Successfully");
80. </script>
81. }
82. @**if** (TempData["result2"] != **null**)
83. {
84. <script type="text/javascript">
85. alert("Record Is Updated Successfully");
86. </script>
87. }
88. @**if** (TempData["result3"] != **null**)
89. {
90. <script type="text/javascript">
91. alert("Record Is Deleted Successfully");
92. </script>
93. }
94. <br/>
95. <br/>
96. <footer>
97. <p style="background-color: Yellow;text-align:center ; color:blue">© @DateTime.Now.ToLocalTime()</p> @\*Add Date Time\*@
98. </footer>
99. </div>
100. </body> </html>

**Code description**In this page, all the data will be visible along with EDIT/ DELETE/ DETAILS link to perform Crud operation.

1. @Html.ActionLink("Modify", "Edit", **new** { id = item.CustomerID }) |
3. @Html.ActionLink("Detail", "Details", **new** { id = item.CustomerID }) |
4. @\*@Html.ActionLink("Remove", "Delete", **new** { id = item.CustomerID })\*@
5. @Html.ActionLink("Remove", "Delete", **new** { id = item.CustomerID},
6. **new** { onclick = "return confirm('Are sure wants to delete?');" })

The data will be shown, using “@Html.DisplayFor” html helper control in looping.

1. @foreach (**var** item **in** Model.ShowallCustomer)
3. <td>
4. @Html.DisplayFor(modelItem => item.Name)
5. </td>
6. <td>
7. @Html.DisplayFor(modelItem => item.Address)
8. </td>
9. <td>
10. @Html.DisplayFor(modelItem => item.Mobileno)
11. </td>
12. <td>
13. @Html.DisplayFor(modelItem => item.Birthdate)
14. </td>
15. <td>
16. @Html.DisplayFor(modelItem => item.EmailID)
17. </td>

20. The header of data will be shown as mentioned **in** code.
21. <th style="background-color: Yellow;color: blue">
22. @Html.DisplayNameFor(model => model.Name)
24. </th>
25. <th style="background-color: Yellow;color: blue">
26. @Html.DisplayNameFor(model => model.Address)
27. </th>
28. <th style="background-color: Yellow;color: blue">
29. @Html.DisplayNameFor(model => model.Mobileno)
30. </th>
31. <th style="background-color: Yellow;color: blue">
32. @Html.DisplayNameFor(model => model.Birthdate)
33. </th>
34. <th style="background-color: Yellow;color: blue">
35. @Html.DisplayNameFor(model => model.EmailID)
36. </th>
37. <th style="background-color: Yellow;color: blue">
38. Operation
39. </th>

Here, I added the namespace of customer model class**.**

1. @model MVC4crud.Models.Customer

The title of the page will be written here.

1. <title>ShowAllCustomerDetails</title>

To go to new customer insertion view page, the code is given below.

1. @Html.ActionLink(linkText: "New Customer", actionName: "InsertCustomer", controllerName: "Customer")

Here ,

Link name : "New Customer",

The method defined in customer control class file as well as data access layer class file is InsertCustomer,

Controller Name: "Customer"

Here, I am using tempdata method mechanism to transfer the data from one page to other.

To show insertion successful message, the code is given below.

1. @**if** (TempData["result1"] != **null**)
2. {
3. <script type="text/javascript">
4. alert("Record Is Inserted Successfully");
5. </script>

To show update successful message, the code is given below.

1. @**if** (TempData["result2"] != **null**)
2. {
3. <script type="text/javascript">
4. alert("Record Is Updated Successfully");
5. </script>
6. }

To show delete successful message, the code is given below.

1. @**if** (TempData["result3"] != **null**)
2. {
3. <script type="text/javascript">
4. alert("Record Is Deleted Successfully");
5. </script>

To get current data time for better visualization to the client, the code is given below.

1. <footer>
2. <p style="background-color: Yellow;text-align:center ; color:blue">© @DateTime.Now.ToLocalTime()</p> @\*Add Date Time\*@
3. </footer>



**Step 8**

Add connection string in Web.config file.

**Code Ref**

1. <connectionStrings>
2. <add name="mycon" providerName="System.Data.SqlClient" connectionString="Your Connection string put here" />
3. </connectionStrings>

**Code description**

Here, “mycon” is the connection string name to be mentioned in Data Access Layer class file to make connection to the database as well as make CRUD operation.

1. add name="mycon"

Now, put your correct connection string .

1. connectionString = “”



**Step 9**

Set start page when MVC page loads first time.

**Code Ref**

1. routes.MapRoute(
2. name: "Default",
3. url: "{controller}/{action}/{id}",
4. defaults: **new** { controller = "Customer", action = "ShowAllCustomerDetails", id = UrlParameter.Optional }
5. );

**Code description**

Here, I have mentioned set start page.

1. defaults: **new** { controller = "Customer", action = "ShowAllCustomerDetails", id = UrlParameter.Optional }

Here, Controller name is Customer.

Now, the view name/ controller action method is ShowAllCustomerDetails.



**Step 10**

You can customize the style of your own view design by using Site.css.

Here, you can add color, font size, font style and margin etc.

ASP.NET

**Step 11**

Add and check reference Dll/ Assembly files to check version and other information.

In the References folder, you can check all DLL file information by right click and going to properties.



**OUTPUT**

The set start page URL is given below.

*http://localhost:62159/Customer/ShowAllCustomerDetails*

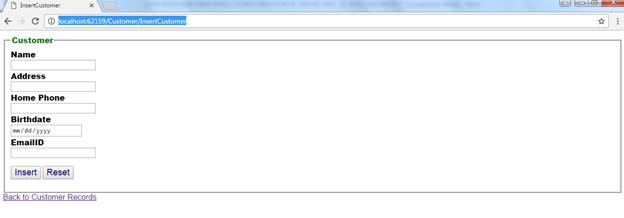
**Load data details with CRUD functionalities**

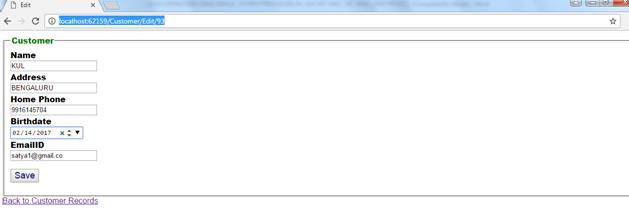
*http://localhost:62159/Customer/ShowAllCustomerDetails*

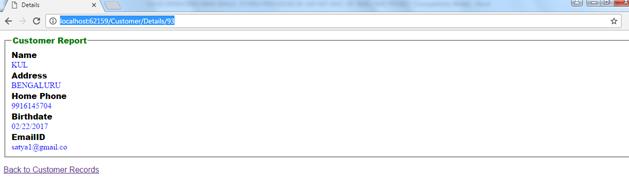


**Insert page**

*http://localhost:62159/Customer/InsertCustomer*

  
  
**Update  page**

*http://localhost:62159/Customer/Edit/93  
  
*

**Details page for report requirement***http://localhost:62159/Customer/Details/93*  
  


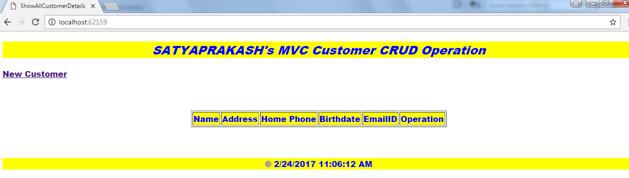
**Delete page**

For delete confirmation, it is, as shown below.

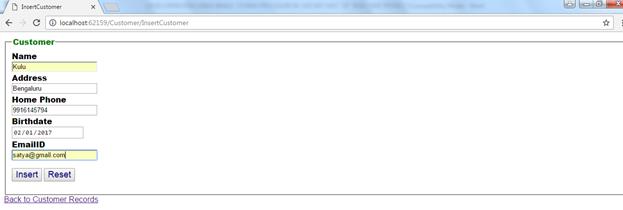


Total Operations In One Flow

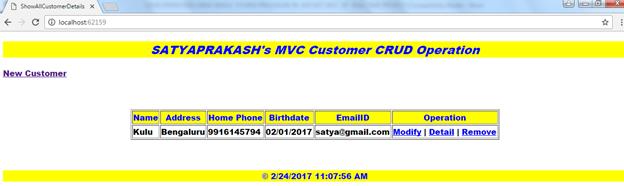
Show all the data on the page is given below.

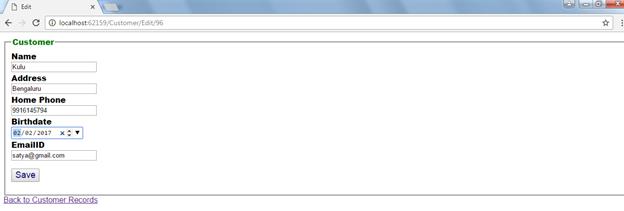


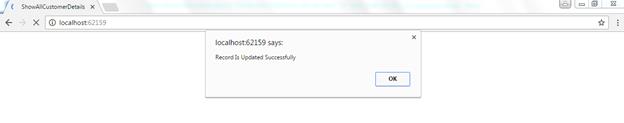
**Insert some data**



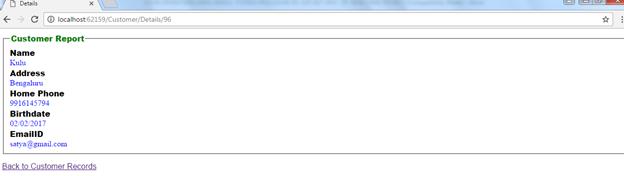


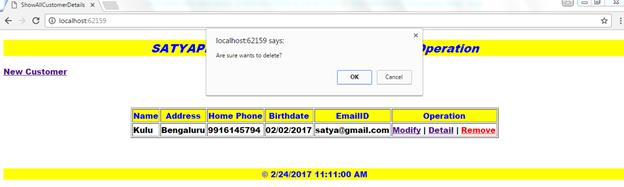
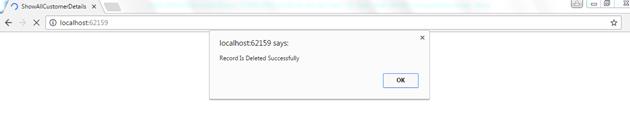


**Update some data**

****

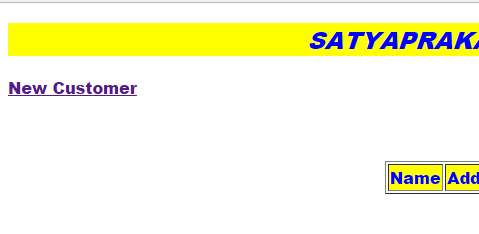
**Details of some data**



**Delete of some data**  
  
  
  


Now, the deleted data is not showing that it is empty now.  
  


To insert new records, click the [New Customer](http://localhost:62159/Customer/InsertCustomer) link.



Check date and time at footer of the show details view page.ASP.NET

Like above mentioned methods, you can implement CRU operation, using MVC in your real time scenario.

**Summary**

1. What is CRUD.
2. How to set code to implement CRUD in MVC.
3. Backend set up is same as real time scenario.
4. Output steps.