Introduction

ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework. ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. Once we create a Web API, we can consume it in Windows, Web and Mobile devices easily. I believe that is the big advantage of Web APIs.

In this lab, we will see how to create a simple Web API with all CRUD operations and will connect with an existing MS SQL database. After that, we will create an MVC application and consume this Web API for CRUD actions.

Create "Employees" table in MSSQL database

In this Lab, we will see how to create an Employee data entry application. So, we need to create an “Employees” table first. If you have any existing database, please create this table under that database. Otherwise, create a new database as well.

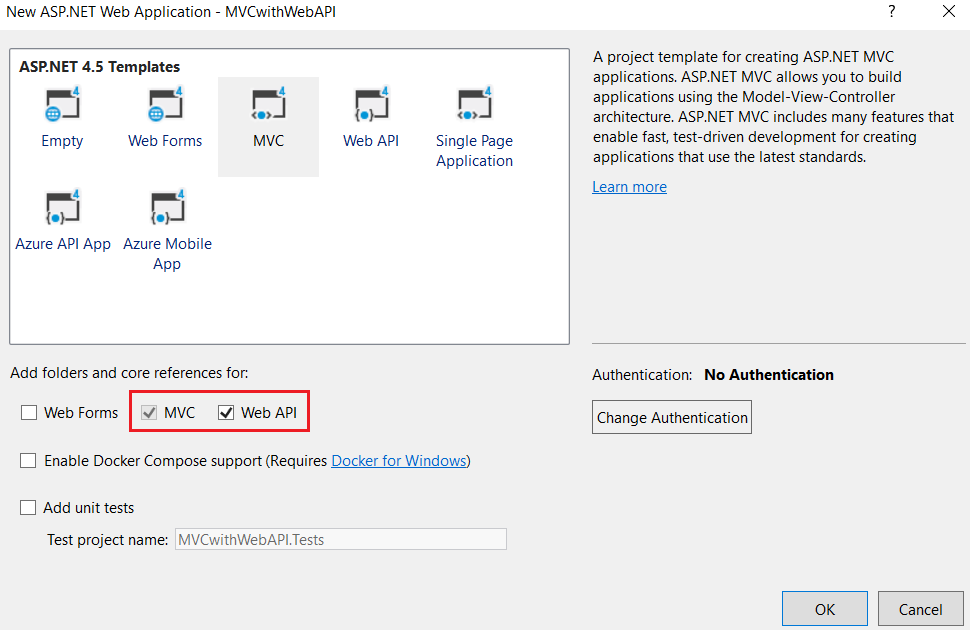
You can use below SQL statement to create a new table.

1. USE [SarathlalDB]
2. GO
4. **CREATE** **TABLE** [dbo].[Employees](
5. [Id] [nvarchar](50) NOT NULL,
6. [**Name**] [nvarchar](50) NULL,
7. [Address] [nvarchar](50) NULL,
8. [Gender] [nvarchar](10) NULL,
9. [Company] [nvarchar](50) NULL,
10. [Designation] [nvarchar](50) NULL,
11. **CONSTRAINT** [PK\_Employees] **PRIMARY** **KEY** CLUSTERED
12. (
13. [Id] **ASC**
14. )
15. )
16. GO

Create a Visual Studio project with MVC and Web API templates

We can create a new project in VS 2015/2017. I am using a free 2017 community edition.

Please choose MVC template and choose the Web API option so, that we can get the bootstrapping for Web API in our project. No need to change any other default parameters.



Our new project will be ready in a few moments.

As I mentioned earlier, we are creating an Employee data entry application. Hence, please create an Employee class under “Models” folder.

Please note, I am creating all business classes inside this “Models” folder for simplicity. If you are following other design patterns, you can keep the files accordingly.

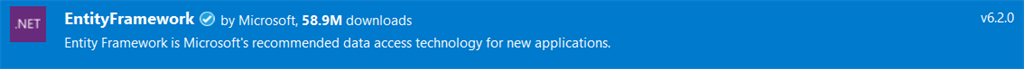
**Employee.cs**

1. **namespace** MVCwithWebAPI.Models
2. {
3. **public** **class** Employee
4. {
5. **public** **string** Id { **get**; **set**; }
6. **public** **string** Name { **get**; **set**; }
7. **public** **string** Address { **get**; **set**; }
8. **public** **string** Gender { **get**; **set**; }
9. **public** **string** Company { **get**; **set**; }
10. **public** **string** Designation { **get**; **set**; }
11. }
12. }

We can create a “DbContext” class for database connectivity.

The class that derives DbContext is called context class in entity framework. DbContext is an important class in Entity Framework API. It is a bridge between domain or entity classes and the database. DbContext is the primary class that is responsible for interacting with the database.

Before creating our DbContext class, we must install “EntityFramework” NuGet package in our project.

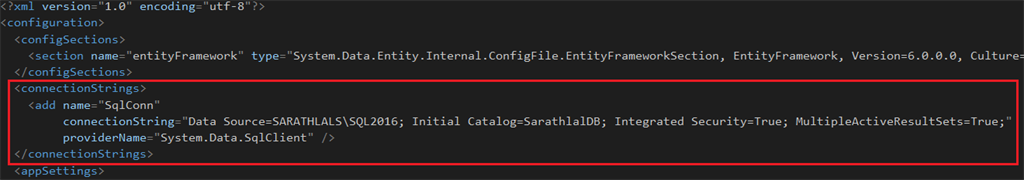


We can create “SqlDbContext” class and derives DbContext class insides this class.

**SqlDbContext.cs**

1. **using** System.Data.Entity;
3. **namespace** MVCwithWebAPI.Models
4. {
5. **public** **class** SqlDbContext : DbContext
6. {
7. **public** SqlDbContext() : **base**("name=SqlConn")
8. {
9. }
10. **public** DbSet<Employee> Employees { **get**; **set**; }
11. }
12. }

Please note, we have used a connection “SqlConn” in above DbContext class. Hence, we can create the connection string in Web.Config file.



1. <connectionStrings>
2. <add name="SqlConn"
3. connectionString="Data Source=SARATHLALS\SQL2016; Initial Catalog=SarathlalDB; Integrated Security=True; MultipleActiveResultSets=True;"
4. providerName="System.Data.SqlClient" />
5. </connectionStrings>

We are following the repository pattern in this application. We can create a “IEmployeeRepository” interface and define all the functions there.

**IEmployeeRepository.cs**

1. **using** System.Collections.Generic;
2. **using** System.Threading.Tasks;
4. **namespace** MVCwithWebAPI.Models
5. {
6. **public** **interface** IEmployeeRepository
7. {
8. Task Add(Employee employee);
9. Task Update(Employee employee);
10. Task Delete(**string** id);
11. Task<Employee> GetEmployee(**string** id);
12. Task<IEnumerable<Employee>> GetEmployees();
13. }
14. }

We can implement the exact logic for CRUD actions in “EmployeeRepository” class. We will implement IEmployeeRepository interface in this class.

**EmployeeRepository.cs**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Data.Entity;
4. **using** System.Linq;
5. **using** System.Threading.Tasks;
6. **using** System.Web;
8. **namespace** MVCwithWebAPI.Models
9. {
10. **public** **class** EmployeeRepository : IEmployeeRepository
11. {
12. **private** **readonly** SqlDbContext db = **new** SqlDbContext();
13. **public** async Task Add(Employee employee)
14. {
15. employee.Id = Guid.NewGuid().ToString();
16. db.Employees.Add(employee);
17. **try**
18. {
19. await db.SaveChangesAsync();
20. }
21. **catch**
22. {
23. **throw**;
24. }
25. }
26. **public** async Task<Employee> GetEmployee(**string** id)
27. {
28. **try**
29. {
30. Employee employee = await db.Employees.FindAsync(id);
31. **if** (employee == **null**)
32. {
33. **return** **null**;
34. }
35. **return** employee;
36. }
37. **catch**
38. {
39. **throw**;
40. }
41. }
42. **public** async Task<IEnumerable<Employee>> GetEmployees()
43. {
44. **try**
45. {
46. var employees = await db.Employees.ToListAsync();
47. **return** employees.AsQueryable();
48. }
49. **catch**
50. {
51. **throw**;
52. }
53. }
54. **public** async Task Update(Employee employee)
55. {
56. **try**
57. {
58. db.Entry(employee).State = EntityState.Modified;
59. await db.SaveChangesAsync();
60. }
61. **catch**
62. {
63. **throw**;
64. }
65. }
66. **public** async Task Delete(**string** id)
67. {
68. **try**
69. {
70. Employee employee = await db.Employees.FindAsync(id);
71. db.Employees.Remove(employee);
72. await db.SaveChangesAsync();
73. }
74. **catch**
75. {
76. **throw**;
77. }
78. }
80. **private** **bool** EmployeeExists(**string** id)
81. {
82. **return** db.Employees.Count(e => e.Id == id) > 0;
83. }
85. }
86. }

I have implemented all 5 methods (for CRUD) in this class. All are self-explanatory. If you need further clarification on any terms, please feel free to contact me.

We can create our API Controller now.

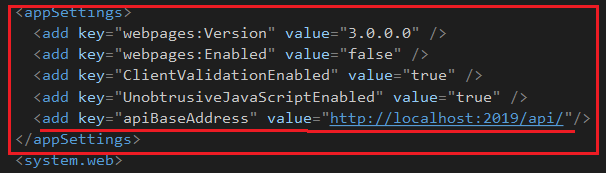
**EmployeesApiController.cs**

1. **using** MVCwithWebAPI.Models;
2. **using** System.Collections.Generic;
3. **using** System.Threading.Tasks;
4. **using** System.Web.Http;
6. **namespace** MVCwithWebAPI.Controllers
7. {
8. **public** **class** EmployeesApiController : ApiController
9. {
10. **private** **readonly** IEmployeeRepository \_iEmployeeRepository = **new** EmployeeRepository();
12. [HttpGet]
13. [Route("api/Employees/Get")]
14. **public** async Task<IEnumerable<Employee>> Get()
15. {
16. **return** await \_iEmployeeRepository.GetEmployees();
17. }
19. [HttpPost]
20. [Route("api/Employees/Create")]
21. **public** async Task CreateAsync([FromBody]Employee employee)
22. {
23. **if** (ModelState.IsValid)
24. {
25. await \_iEmployeeRepository.Add(employee);
26. }
27. }
29. [HttpGet]
30. [Route("api/Employees/Details/{id}")]
31. **public** async Task<Employee> Details(**string** id)
32. {
33. var result = await \_iEmployeeRepository.GetEmployee(id);
34. **return** result;
35. }
37. [HttpPut]
38. [Route("api/Employees/Edit")]
39. **public** async Task EditAsync([FromBody]Employee employee)
40. {
41. **if** (ModelState.IsValid)
42. {
43. await \_iEmployeeRepository.Update(employee);
44. }
45. }
47. [HttpDelete]
48. [Route("api/Employees/Delete/{id}")]
49. **public** async Task DeleteConfirmedAsync(**string** id)
50. {
51. await \_iEmployeeRepository.Delete(id);
52. }
53. }
54. }

All the CRUD actions are derived in this API class. We have created an instance for EmployeeRepository class and with the help of this instance, we have accessed all the methods from EmployeeRepository class in our API class.

We have completed our Web API design. If needed, you can check the Web API using Postman or any other tool. Please note down the URL of the application. We need to add this base URL in our Web.Config file because we will use this base URL in MVC controllers.

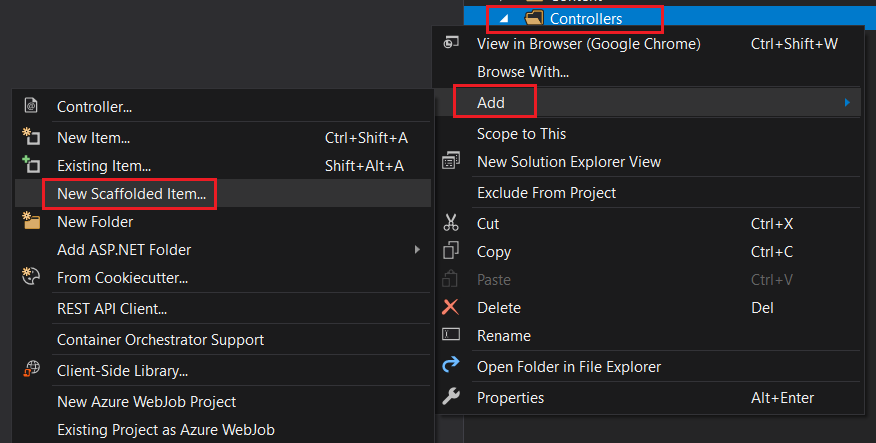
We can create a new key-value pair in Web.Config file under “appSettings” section.



We have named the key as “apiBaseAddress” and gave the project URL as value.

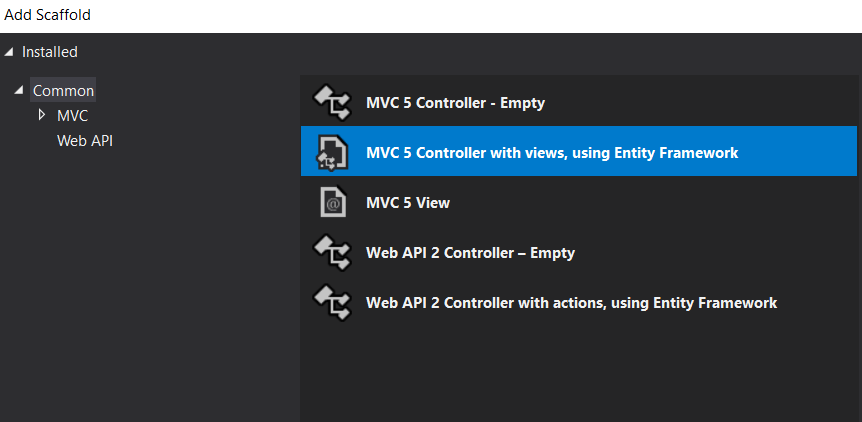
Create the MVC Controller using Scaffolding

We can create the MVC Controller using scaffolding template so that we will get all the views for CRUD actions easily. We will modify these views later.

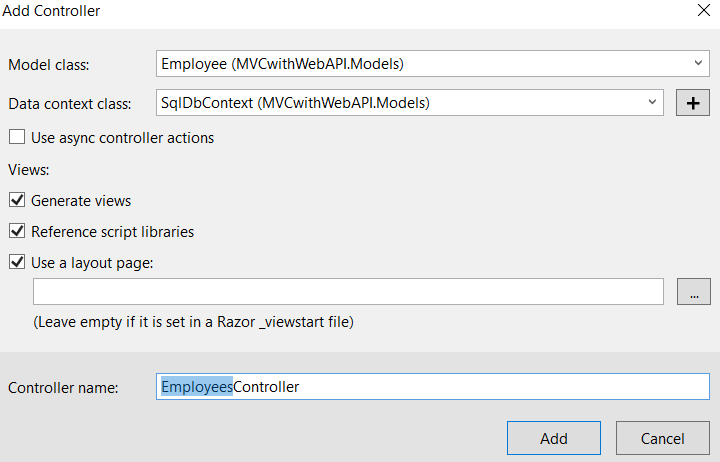


We can right-click the “Controller” folder and click “Add” and choose “New Scaffolded Item” to create a new MVC controller.

You can choose, “MVC 5 Controller with views, using Entity Framework” option. This will create all views for CRUD operations.

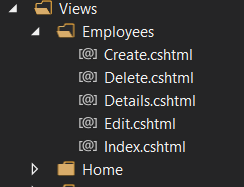


We can choose the model class, data context class and give controller name as well. Please note, the system will automatically give a name for a controller. If needed, you can change it.



You can click the “Add” button to proceed further. After a few moments, our Employees controller is created with all views.

You can see the views under “Views” -> “Employees” folder.



We will modify some of the view files later. Let us modify the “EmployeesController” class now. Please note, by default MVC controller does not create actions as async. We have converted all these actions asynchronously. You can copy the below code and paste it inside the controller class.

**EmployeesController.cs**

1. **using** MVCwithWebAPI.Models;
2. **using** System;
3. **using** System.Collections.Generic;
4. **using** System.Configuration;
5. **using** System.Linq;
6. **using** System.Net;
7. **using** System.Net.Http;
8. **using** System.Threading.Tasks;
9. **using** System.Web.Mvc;
11. **namespace** MVCwithWebAPI.Controllers
12. {
13. **public** **class** EmployeesController : Controller
14. {
15. **readonly** **string** apiBaseAddress = ConfigurationManager.AppSettings["apiBaseAddress"];
16. **public** async Task<ActionResult> Index()
17. {
18. IEnumerable<Employee> employees = **null**;
20. **using** (var client = **new** HttpClient())
21. {
22. client.BaseAddress = **new** Uri(apiBaseAddress);
24. var result = await client.GetAsync("employees/get");
26. **if** (result.IsSuccessStatusCode)
27. {
28. employees = await result.Content.ReadAsAsync<IList<Employee>>();
29. }
30. **else**
31. {
32. employees = Enumerable.Empty<Employee>();
33. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
34. }
35. }
36. **return** View(employees);
37. }
39. **public** async Task<ActionResult> Details(**string** id)
40. {
41. **if** (id == **null**)
42. {
43. **return** **new** HttpStatusCodeResult(HttpStatusCode.BadRequest);
44. }
46. Employee employee = **null**;
47. **using** (var client = **new** HttpClient())
48. {
49. client.BaseAddress = **new** Uri(apiBaseAddress);
51. var result = await client.GetAsync($"employees/details/{id}");
53. **if** (result.IsSuccessStatusCode)
54. {
55. employee = await result.Content.ReadAsAsync<Employee>();
56. }
57. **else**
58. {
59. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
60. }
61. }
63. **if** (employee == **null**)
64. {
65. **return** HttpNotFound();
66. }
67. **return** View(employee);
68. }
70. **public** ActionResult Create()
71. {
72. **return** View();
73. }
75. [HttpPost]
76. [ValidateAntiForgeryToken]
77. **public** async Task<ActionResult> Create([Bind(Include = "Name,Address,Gender,Company,Designation")] Employee employee)
78. {
79. **if** (ModelState.IsValid)
80. {
81. **using** (var client = **new** HttpClient())
82. {
83. client.BaseAddress = **new** Uri(apiBaseAddress);
85. var response = await client.PostAsJsonAsync("employees/Create", employee);
86. **if** (response.IsSuccessStatusCode)
87. {
88. **return** RedirectToAction("Index");
89. }
90. **else**
91. {
92. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
93. }
94. }
95. }
96. **return** View(employee);
97. }
99. **public** async Task<ActionResult> Edit(**string** id)
100. {
101. **if** (id == **null**)
102. {
103. **return** **new** HttpStatusCodeResult(HttpStatusCode.BadRequest);
104. }
105. Employee employee = **null**;
106. **using** (var client = **new** HttpClient())
107. {
108. client.BaseAddress = **new** Uri(apiBaseAddress);
110. var result = await client.GetAsync($"employees/details/{id}");
112. **if** (result.IsSuccessStatusCode)
113. {
114. employee = await result.Content.ReadAsAsync<Employee>();
115. }
116. **else**
117. {
118. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
119. }
120. }
121. **if** (employee == **null**)
122. {
123. **return** HttpNotFound();
124. }
125. **return** View(employee);
126. }
128. [HttpPost]
129. [ValidateAntiForgeryToken]
130. **public** async Task<ActionResult> Edit([Bind(Include = "Id,Name,Address,Gender,Company,Designation")] Employee employee)
131. {
132. **if** (ModelState.IsValid)
133. {
134. **using** (var client = **new** HttpClient())
135. {
136. client.BaseAddress = **new** Uri(apiBaseAddress);
137. var response = await client.PutAsJsonAsync("employees/edit", employee);
138. **if** (response.IsSuccessStatusCode)
139. {
140. **return** RedirectToAction("Index");
141. }
142. **else**
143. {
144. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
145. }
146. }
147. **return** RedirectToAction("Index");
148. }
149. **return** View(employee);
150. }
152. **public** async Task<ActionResult> Delete(**string** id)
153. {
154. **if** (id == **null**)
155. {
156. **return** **new** HttpStatusCodeResult(HttpStatusCode.BadRequest);
157. }
158. Employee employee = **null**;
159. **using** (var client = **new** HttpClient())
160. {
161. client.BaseAddress = **new** Uri(apiBaseAddress);
163. var result = await client.GetAsync($"employees/details/{id}");
165. **if** (result.IsSuccessStatusCode)
166. {
167. employee = await result.Content.ReadAsAsync<Employee>();
168. }
169. **else**
170. {
171. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
172. }
173. }
175. **if** (employee == **null**)
176. {
177. **return** HttpNotFound();
178. }
179. **return** View(employee);
180. }
182. [HttpPost, ActionName("Delete")]
183. [ValidateAntiForgeryToken]
184. **public** async Task<ActionResult> DeleteConfirmed(**string** id)
185. {
186. **using** (var client = **new** HttpClient())
187. {
188. client.BaseAddress = **new** Uri(apiBaseAddress);
190. var response = await client.DeleteAsync($"employees/delete/{id}");
191. **if** (response.IsSuccessStatusCode)
192. {
193. **return** RedirectToAction("Index");
194. }
195. **else**
196. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
197. }
198. **return** View();
199. }
201. }
202. }

You can see, we have defined an “apiBaseAddress” variable globally and got the value for apiBaseAddress from Web.Config file. We will use this value in all our controller actions.

**Index Action**

1. **public** async Task<ActionResult> Index()
2. {
3. IEnumerable<Employee> employees = **null**;
5. **using** (var client = **new** HttpClient())
6. {
7. client.BaseAddress = **new** Uri(apiBaseAddress);
9. var result = await client.GetAsync("employees/get");
11. **if** (result.IsSuccessStatusCode)
12. {
13. employees = await result.Content.ReadAsAsync<IList<Employee>>();
14. }
15. **else**
16. {
17. employees = Enumerable.Empty<Employee>();
18. ModelState.AddModelError(**string**.Empty, "Server error try after some time.");
19. }
20. }
21. **return** View(employees);
22. }

If you look at the index action, you can see, we have declared a “HttpClient” variable and using client.GetAsync method, we have got the employee data result from Web API and store in a “result” variable. We have again read the employee data from this result variable using “ReadAsync” method.

We have used the same approach in other action methods also. All the methods are self-explanatory. If you have any queries, please feel free to contact me.

We can now modify the Index view. Please copy the below and paste to Index view file.

**Index.cshtml**

1. @model IEnumerable<MVCwithWebAPI.Models.Employee>
3. @{
4. ViewBag.Title = "Employee List";
5. }
7. <h3>Employee List</h3>
9. <p>
10. @Html.ActionLink("New Employee", "Create")
11. </p>
12. <table **class**="table">
13. <tr>
14. <th>
15. @Html.DisplayNameFor(model => model.Name)
16. </th>
17. <th>
18. @Html.DisplayNameFor(model => model.Address)
19. </th>
20. <th>
21. @Html.DisplayNameFor(model => model.Gender)
22. </th>
23. <th>
24. @Html.DisplayNameFor(model => model.Company)
25. </th>
26. <th>
27. @Html.DisplayNameFor(model => model.Designation)
28. </th>
29. <th></th>
30. </tr>
32. @**foreach** (var item **in** Model)
33. {
34. <tr>
35. <td>
36. @Html.ActionLink(item.Name, "Details", **new** { id = item.Id })
37. </td>
38. <td>
39. @Html.DisplayFor(modelItem => item.Address)
40. </td>
41. <td>
42. @Html.DisplayFor(modelItem => item.Gender)
43. </td>
44. <td>
45. @Html.DisplayFor(modelItem => item.Company)
46. </td>
47. <td>
48. @Html.DisplayFor(modelItem => item.Designation)
49. </td>
50. <td>
51. @Html.ActionLink("Edit", "Edit", **new** { id = item.Id }) |
52. @Html.ActionLink("Delete", "Delete", **new** { id = item.Id })
53. </td>
54. </tr>
55. }
57. </table>

We have modified the existing “Index” view. We have removed the “Details” link from this view and instead, we have given a hyperlink in the employee name itself for details.

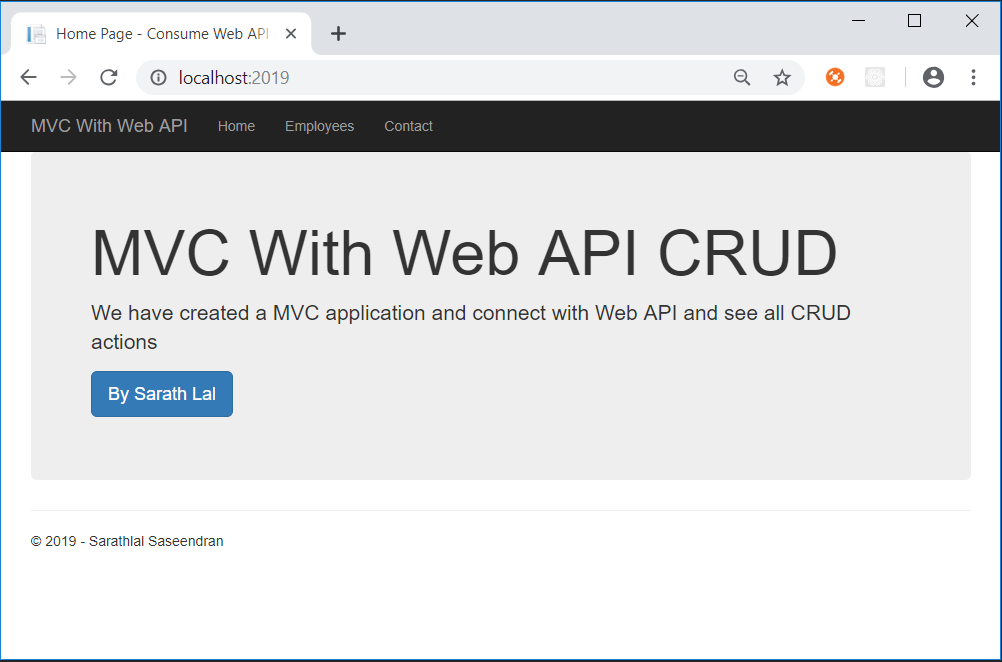
We can modify the “Create” view by removing the Id field. Because for us, employee Id will be created automatically while inserting new data. We have used system GUID for this.

**Create.cshtml**

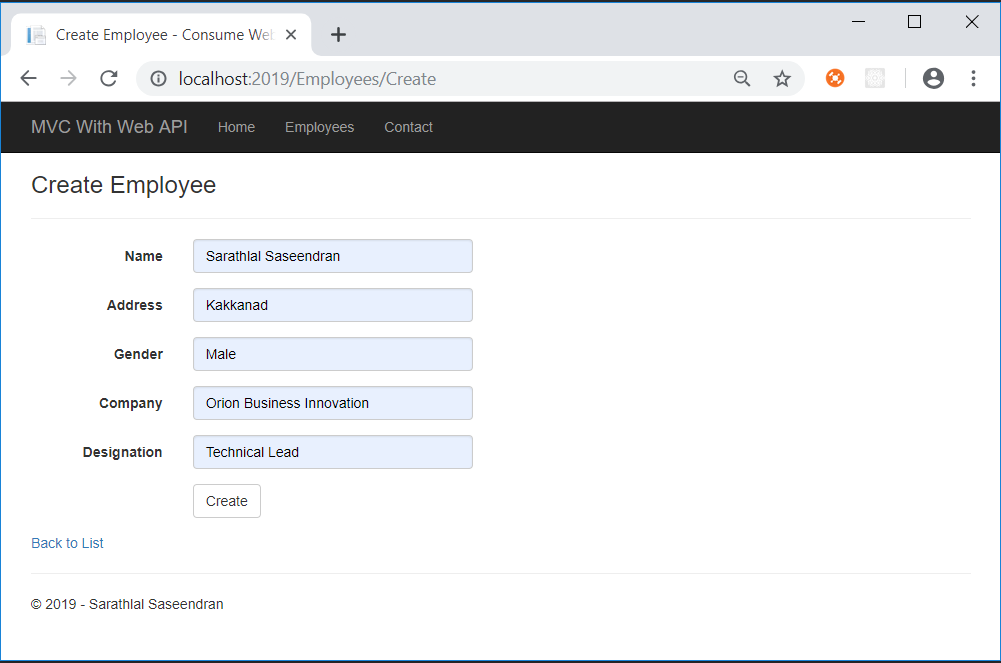
1. @model MVCwithWebAPI.Models.Employee
3. @{
4. ViewBag.Title = "Create Employee";
5. }
7. <h3>Create Employee</h3>

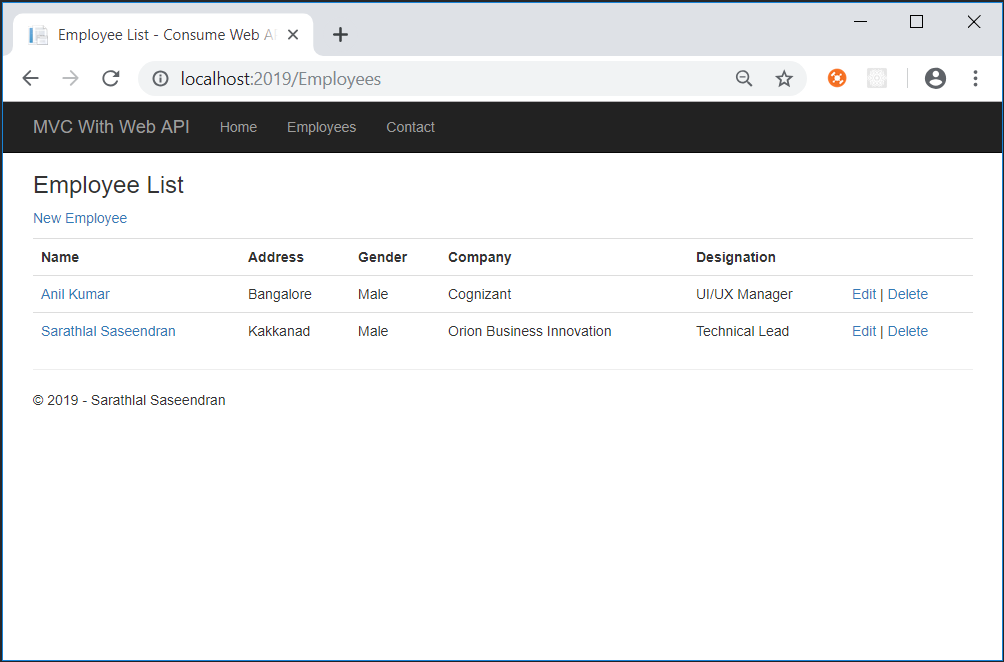
10. @**using** (Html.BeginForm())
11. {
12. @Html.AntiForgeryToken()
14. <div **class**="form-horizontal">
15. <hr />
16. @Html.ValidationSummary(**true**, "", **new** { @**class** = "text-danger" })
18. <div **class**="form-group">
19. @Html.LabelFor(model => model.Name, htmlAttributes: **new** { @**class** = "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.Address, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
28. <div **class**="col-md-10">
29. @Html.EditorFor(model => model.Address, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
30. @Html.ValidationMessageFor(model => model.Address, "", **new** { @**class** = "text-danger" })
31. </div>
32. </div>
34. <div **class**="form-group">
35. @Html.LabelFor(model => model.Gender, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
36. <div **class**="col-md-10">
37. @Html.EditorFor(model => model.Gender, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
38. @Html.ValidationMessageFor(model => model.Gender, "", **new** { @**class** = "text-danger" })
39. </div>
40. </div>
42. <div **class**="form-group">
43. @Html.LabelFor(model => model.Company, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
44. <div **class**="col-md-10">
45. @Html.EditorFor(model => model.Company, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
46. @Html.ValidationMessageFor(model => model.Company, "", **new** { @**class** = "text-danger" })
47. </div>
48. </div>
50. <div **class**="form-group">
51. @Html.LabelFor(model => model.Designation, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
52. <div **class**="col-md-10">
53. @Html.EditorFor(model => model.Designation, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
54. @Html.ValidationMessageFor(model => model.Designation, "", **new** { @**class** = "text-danger" })
55. </div>
56. </div>
58. <div **class**="form-group">
59. <div **class**="col-md-offset-2 col-md-10">
60. <input type="submit" value="Create" **class**="btn btn-default" />
61. </div>
62. </div>
63. </div>
64. }
66. <div>
67. @Html.ActionLink("Back to List", "Index")
68. </div>
70. @section Scripts {
71. @Scripts.Render("~/bundles/jqueryval")
72. }

We can run the application now. The landing page looks like the below screenshot.



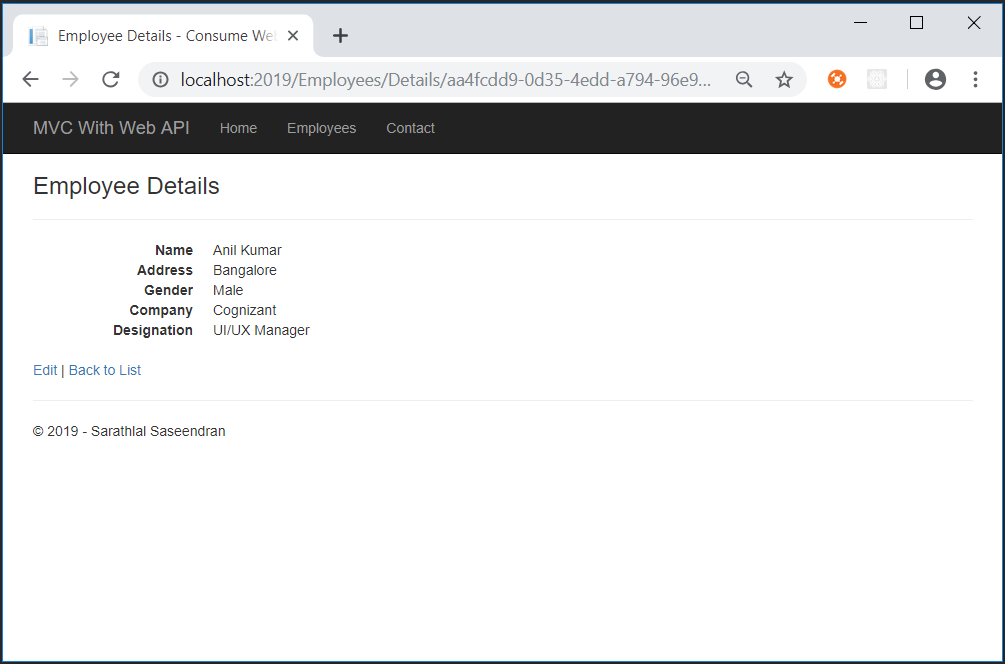
We can click the “Employees” link and click the “New Employee” link to create a new employee.



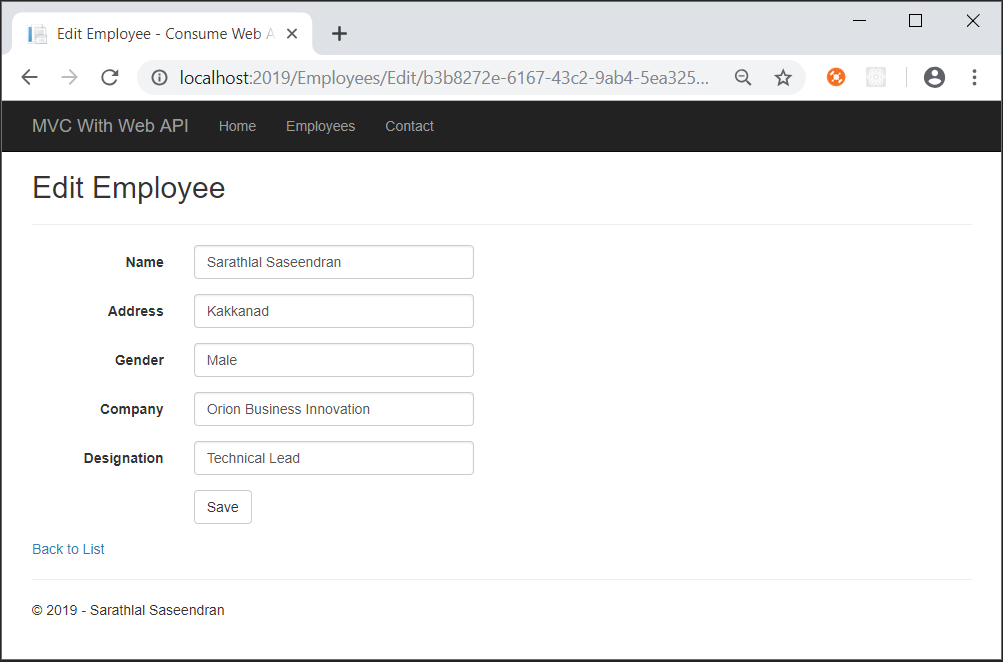


As I mentioned earlier in this Lab, I have removed the default “Details” link from this view and added the hyperlink in employee name itself.

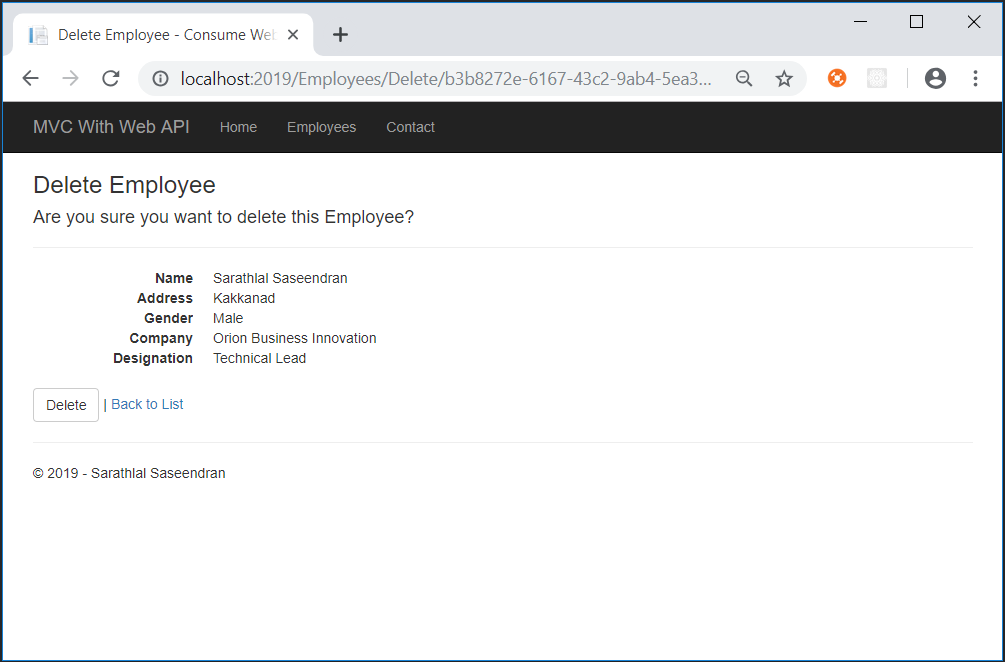
You can click the employee name to show the details.



We can click “Edit” link to edit the employee details.



We can use “Delete” link to delete the record as well.



We have successfully seen all the CRUD actions with this application.