

[Come a member](#)[Login](#)[Post](#)[Ask Question](#)

Debasis Saha

Updated date, Nov 01, 2019

122.7k

21

12

[Download Free .NET & JAVA Files API](#)[Try Free File Format APIs for Word/Excel/PDF](#)[EF_Crud_Samples.rar](#) | [EF_Crud_Samples_ExcelDownload.rar](#)

Entity Framework Core (EF Core) is one of the latest data access technologies for any type of application in the .NET Core framework. Entity Framework Core is based on Object-Relational Mapper or ORM model. Basically, the object-relational model is a technique that helps the developer to establish a relation between the entity model class of the application and the data objects of the relational database. In this technique, developers can create a relation between the entity model class and database objects, i.e., tables in the relational database in an object-oriented way to perform different types of operations like fetch data, save data, etc. Entity Framework Core is a lightweight extensible framework which can support cross-platform development.

Entity Framework Core provides a lot of benefits to the concept of rapid developments. The main advantages of Entity Framework Core are:

1. Entity framework core provides auto generated code.
2. It helps us to reduce development time.
3. It also helps us to reduce development cost.
4. It provides a multiple conceptual model to map with a single schema object.

PREREQUISITES REQUIRED



1. Basic CRUD Operations like Insert, Update, Delete & Read
2. Sort Record
3. Search Record
4. Paging Data for Listing
5. Download Index List Data as Excel File

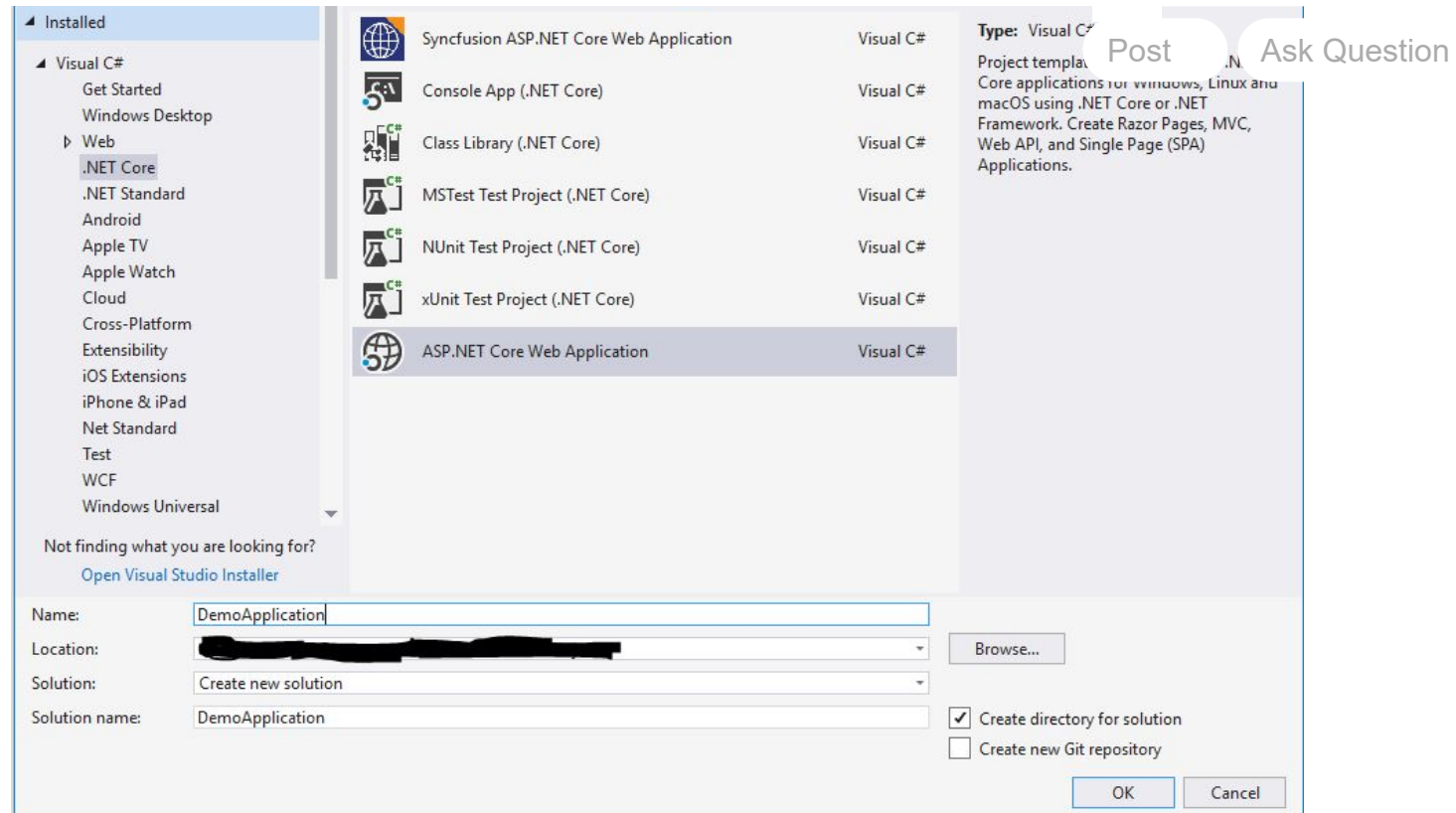
For starting this development, we need to have the below prerequisites.

- Microsoft Visual Studio 2017
- .Net Core 2.1 or Above SDK
- Relation Database like SQL Server

CREATE DATABASE USING ENTITY FRAMEWORK CORE CODE FIRST APPROACH

Step 1

Now, open Microsoft Visual Studio 2017 and click on File --> New --> Projects.



Step 2

Select the Web Application Project template and click the OK button.

.NET Core

ASP.NET Core 2.2

[Learn more](#)

Empty

API

Web Application

Web Application (Model-View-Controller)

Razor Class Library

Angular

React.js

React.js and Redux

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

[Learn more](#)

Author: Microsoft
Source: SDK 2.2.103

Authentication: **No Authentication**

[Change Authentication](#)

[Get additional project templates](#)

☐ Enable Docker Support (Requires [Docker for Windows](#))

OS:

Windows

☒ Configure for HTTPS

OK

Cancel

Step 3

Step 4

Now, a blank project solution has been ready.

[Come a member](#)[Login](#)[Post](#)[Ask Question](#)

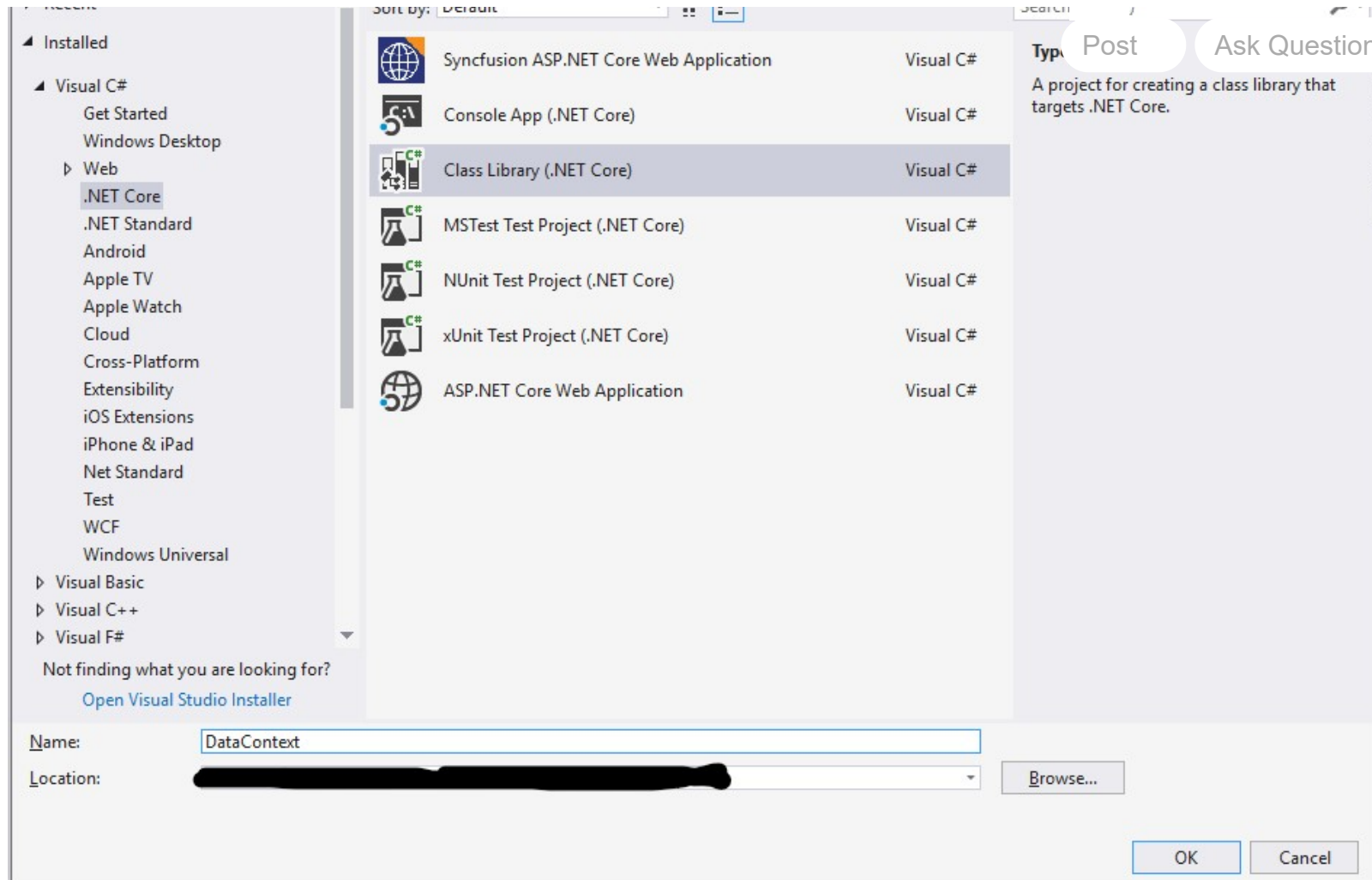
Now, select the solution file and right-click and select Add --> New Projects.

Step 6

Now, select Class Library (.NET Core) Project Template

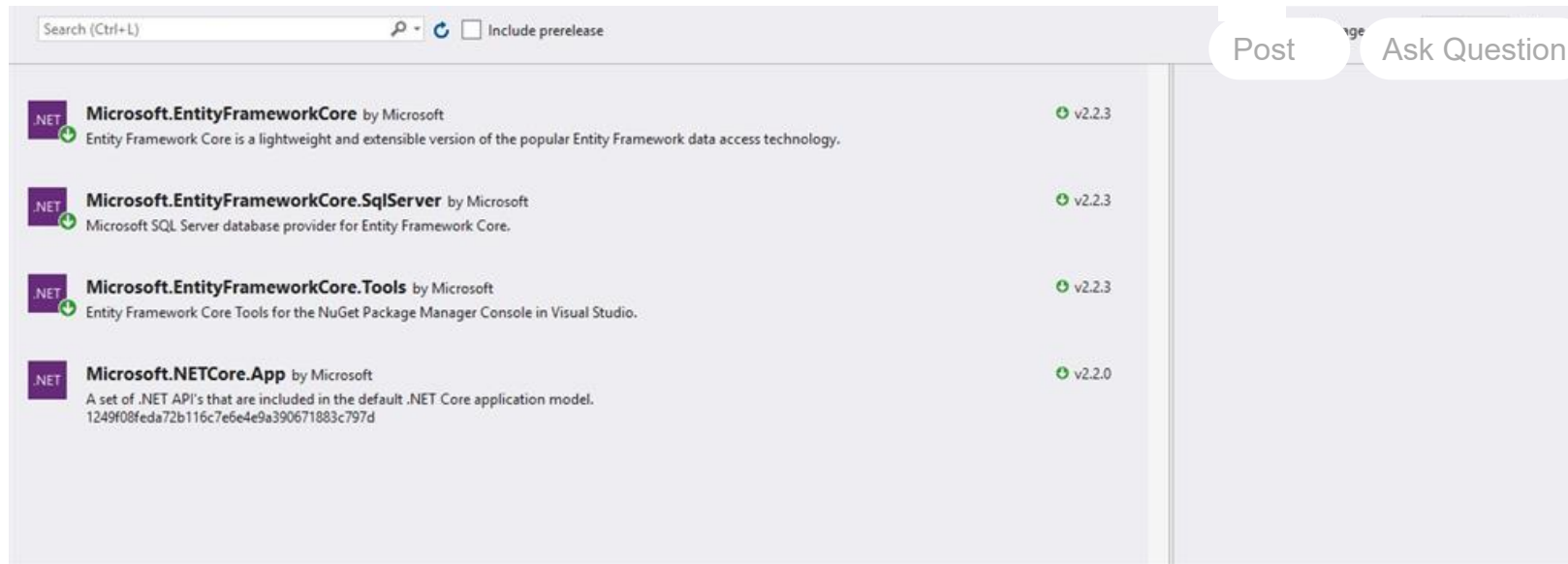
Step 7

Provide a Project Name as DataContext and click OK.



Step 8

- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools



Step 9

Now, create a folder called Models in this project and add the below models.

Department.cs --> Department class which contains Department name list

```
01. using System;
02. using System.Collections.Generic;
03. using System.ComponentModel.DataAnnotations;
04. using System.ComponentModel.DataAnnotations.Schema;
05. using System.Text;
06.
07. namespace DataContext.Models
08. {
09.
10.
11.     {
12.         [Key]
13.         [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
14.         [Display(Name = "Department Id")]
```



```

17.         [Required]
18.         [Column(TypeName = "Nvarchar(20)")]
19.         [Display(Name = "Department Code")]
20.         public string Alias { get; set; }
21.
22.         [Required]
23.         [Column(TypeName = "Nvarchar(100)")]
24.         [Display(Name = "Department Description")]
25.         public string DepartmentName { get; set; }
26.
27.     }
28. }
```

Designation.cs --> Designation class which contains Designation Name List

```

01. using System;
02. using System.Collections.Generic;
03. using System.ComponentModel.DataAnnotations;
04. using System.ComponentModel.DataAnnotations.Schema;
05. using System.Text;
06.
07. namespace DataContext.Models
08. {
09.     [Table("Designation", Schema = "dbo")]
10.     public class Designation
11.     {
12.         [Key]
13.         [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
14.         public int DesignationId { get; set; }
15.
16.         [Required]
17.         [Column(TypeName = "Nvarchar(20)")]
18.
19.
20.
21.         [Required]
22.         [Column(TypeName = "varchar(100)")]
23.         [Display(Name = "Designation Name")]
24.     }
25. }
```




Employee.cs --> Employee class which contains Employee details List. In this entity, DepartmentId and DesignationId is acting as a Foreign Key related to the Department and Designation Entity.

```
01. using System;
02. using System.Collections.Generic;
03. using System.ComponentModel.DataAnnotations;
04. using System.ComponentModel.DataAnnotations.Schema;
05. using System.Text;
06.
07. namespace DataContext.Models
08. {
09.     [Table("Employee", Schema = "dbo")]
10.     public class Employee
11.     {
12.         [Key]
13.         [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
14.         [Display(Name = "Employee Id")]
15.         public int Id { get; set; }
16.
17.         [Required]
18.         [Column(TypeName = "NVarChar(20)")]
19.         [Display(Name = "Code")]
20.         public string EmployeeCode { get; set; }
21.
22.         [Required]
23.         [Column(TypeName = "NVarChar(100)")]
24.         [Display(Name = "Employee Name")]
25.         public string EmployeeName { get; set; }
26.
27.         [Required]
28.
29.         [DataType(DataType.Date)]
30.         public DateTime DateOfBirth { get; set; }
31.
32.
33.         [Required]
```



```
36.         [DataType(DataType.Date)]
37.         public DateTime JoinDate { get; set; }
38.
39.         [Required]
40.         [Column(TypeName = "Decimal(18,2)")]
41.         public decimal Salary { get; set; }
42.
43.         [Column(TypeName = "NVarChar(100)")]
44.         public string Address { get; set; }
45.
46.         [Column(TypeName = "NVarChar(100)")]
47.         public string State { get; set; }
48.
49.         [Column(TypeName = "NVarChar(100)")]
50.         public string City { get; set; }
51.
52.         [Column(TypeName = "NVarChar(20)")]
53.         public string ZipCode { get; set; }
54.
55.         [ForeignKey("DepartmentInfo")]
56.         [Required]
57.         public int DepartmentId { get; set; }
58.
59.         [Display(Name = "Department")]
60.         [NotMapped]
61.         public string DepartmentName { get; set; }
62.
63.         public virtual Department DepartmentInfo { get; set; }
64.
65.         [ForeignKey("DesignationInfo")]
66.         [Required]
67.         public int DesignationId { get; set; }
68.
69.
70.         [NotMapped]
71.         public string DesignationName { get; set; }
72.
73.         public virtual Designation DesignationInfo { get; set; }
74.     }
```

Now, create another folder called DataContext.

Step 11

Within this folder, add the class file name EFDataContext.cs which will contains mapping code related to the entity and database tables. Now, add the below code in this file.

```
01. using DataContext.Models;
02. using Microsoft.EntityFrameworkCore;
03. using System;
04. using System.Collections.Generic;
05. using System.Data.SqlClient;
06. using System.Linq;
07. using System.Text;
08.
09. namespace DataContext.DataContext
10. {
11.     public class EFDataContext : DbContext
12.     {
13.         public DbSet<Department> Departments { get; set; }
14.
15.         public DbSet<Designation> Designations { get; set; }
16.
17.         public DbSet<Employee> Employees { get; set; }
18.
19.         protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
20.         {
21.             optionsBuilder.UseSqlServer(@"data source=xxx; initial catalog=EFCrudDemo;persist securit
22.         }
```

Step 12



so that Entity Framework Core can connect with the database server for the CRUD operations:

Post



Ask Question

Step 13

Now, open the Package Manager Console and run the below command – ***add migration migration1*** (here migration1 is the alias name which can be changed).

Step 14

Now, run the command ***update database.***

Step 15

Now, open the SQL Server and check the mentioned database is already created.

Step 16

Now, in this application, we will create a CRUD operation for Employee data objects. So, we need to insert sample data using a database in the Department and Designation tables so that when we will try to create employee from the application we can generate the department and designation lists.

CREATE EMPLOYEE INDEX UI

Step 1

Now select the DemoApplication Projects (MVC Application) and provide the reference of the DataContext Library which we

Step 2

Now Select the Controller Folder and right click and select Add --> Controller.



In the Controller Template, Select **MVC Controller – Empty** and click on Add Button



Step 4

Provide the Controller Name as EmployeesController and Click on Ok Button.

Step 5

Now in the controller file, create an object for EFDataContext class as below -

```
01. | EFDataContext _dbContext = new EFDataContext();
```

Step 6

Now in the controller file, add a method named Index and write down the below code -

```
01. | public class EmployeesController : Controller
02. | {
03. |     EFDataContext _dbContext = new EFDataContext();
04. |
05. |     public IActionResult Index()
06. |     {
07. |         List<Employee> employees = this._dbContext.Employees.ToList();
08. |         return View(employees);
09. |     }
10. | }
```

Step 7

In the View Folder, create a new folder called Employee and then add an empty view named index.cshtml.



Now open the Index.cshtml view and the the below code.

[Post](#)[Ask Question](#)

```
01. @model IEnumerable<DataContext.Models.Employee>
02.
03. @{
04.     ViewData["Title"] = "Employee List";
05. }
06.
07. <h1>Index</h1>
08.
09. <p>
10.     <a asp-action="Create">Create New</a>
11. </p>
12. <table class="table">
13.     <thead>
14.         <tr>
15.             <th>
16.                 @Html.DisplayNameFor(model => model.EmployeeCode)
17.             </th>
18.             <th>
19.                 @Html.DisplayNameFor(model => model.EmployeeName)
20.             </th>
21.             <th>
22.                 @Html.DisplayNameFor(model => model.DateOfBirth)
23.             </th>
24.             <th>
25.                 @Html.DisplayNameFor(model => model.JoinDate)
26.             </th>
27.             <th>
28.                 @Html.DisplayNameFor(model => model.DepartmentName)
29.             </th>
30.             <th>
31.
32.
33.             <th>
34.                 @Html.DisplayNameFor(model => model.Salary)
35.             </th>
36.             <th>
```



```

39.         <th>
40.             @Html.DisplayNameFor(model => model.City)
41.         </th>
42.     </th></th>
43. </tr>
44. </thead>
45. <tbody>
46. @foreach (var item in Model) {
47.     <tr>
48.         <td>
49.             @Html.DisplayFor(modelItem => item.EmployeeCode)
50.         </td>
51.         <td>
52.             @Html.DisplayFor(modelItem => item.EmployeeName)
53.         </td>
54.         <td>
55.             @Html.DisplayFor(modelItem => item.DateOfBirth)
56.         </td>
57.         <td>
58.             @Html.DisplayFor(modelItem => item.JoinDate)
59.         </td>
60.         <td>
61.             @Html.DisplayFor(modelItem => item.DepartmentName)
62.         </td>
63.         <td>
64.             @Html.DisplayFor(modelItem => item.DesignationName)
65.         </td>
66.         <td>
67.             @Html.DisplayFor(modelItem => item.Salary)
68.         </td>
69.         <td>
70.             @Html.DisplayFor(modelItem => item.State)
71.         </td>
72.         <td>
73.             @Html.ActionLink("Edit", "Edit", new { id = item.Id }) |
74.             @Html.ActionLink("Delete", "Delete", new { id = item.Id })
75.         </td>
76.     </tr>
77. }

```



```
80.     }
81.     </tbody>
82. </table>
```

Step 9

Now open the _Layout.cshtml view in shared folder and the below line of code in that file:

```
01. <header>
02.     <nav class="navbar navbar-expand-sm navbar-togglerable-sm navbar-light bg-white border-
03.         bottom box-shadow mb-3">
04.         <div class="container">
05.             <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
06. action="Index">EF_Crud_Samples</a>
07.             <button class="navbar-toggler" type="button" data-toggle="collapse" data-
08. target=".navbar-collapse" aria-controls="navbarSupportedContent"
09.             aria-expanded="false" aria-label="Toggle navigation">
10.                 <span class="navbar-toggler-icon"></span>
11.             </button>
12.             <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
13.                 <ul class="navbar-nav flex-grow-1">
14.                     <li class="nav-item">
15.                         <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-
16. action="Index">Home</a>
17.                     </li>
18.                     <li class="nav-item">
19.                         <a class="nav-link text-dark" asp-area="" asp-
20. controller="Employees" asp-action="Index">Employee</a>
21.                     </li>
22.                     <li class="nav-item">
23.                         <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-
24. action="Privacy">Privacy</a>
25.                     </li>
26.                 </ul>
27.             </div>
28.         </div>
29.     </nav>
30. </header>
```




Now run the application and click on Employee Link

[Post](#)[Ask Question](#)

CREATE NEW EMPLOYEE UI

Step 1

Now, open the EmployeesController.cs file again and add a new method as below:

```
01. public IActionResult Create()
02.     {
03.         this.GetModelData();
04.         return View();
05.     }
06.
07. private void GetModelData()
08.     {
09.         ViewBag.Departments = this._dbContext.Departments.ToList();
10.         ViewBag.Designations = this._dbContext.Designations.ToList();
11.     }
```

Step 2

In the above method, we create two ViewBag instances for Department and Designation Lists.

Step 3

```
01. @model DataContext.Models.Employee
02.
03. @{
04.     ViewData["Title"] = Model != null ? "Edit" : "Create";
```

```

07. <h2>@ViewData["Title"] Employee</h2>
08.
09. <hr />
10. <div class="row">
11.     <div class="col-lg-12">
12.         <form asp-action="ViewData[" Title"]">
13.             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
14.             <div class="form-group">
15.                 <div class="form-row">
16.                     <div class="col-lg-2">
17.                         <label asp-for="EmployeeCode" class="control-label"></label>
18.                     </div>
19.                     <div class="col-lg-4">
20.                         <input asp-for="EmployeeCode" class="form-control" />
21.                         <span asp-validation-for="EmployeeCode" class="text-danger"></span>
22.                     </div>
23.                     <div class="col-lg-2">
24.                         <label asp-for="EmployeeName" class="control-label"></label>
25.                     </div>
26.                     <div class="col-lg-4">
27.                         <input asp-for="EmployeeName" class="form-control" />
28.                         <span asp-validation-for="EmployeeName" class="text-danger"></span>
29.                     </div>
30.                 </div>
31.
32.                 <div class="form-row">
33.                     <div class="col-lg-2">
34.                         <label asp-for="DateOfBirth" class="control-label"></label>
35.                     </div>
36.                     <div class="col-lg-4">
37.                         <input asp-for="DateOfBirth" class="form-control" />
38.                         <span asp-validation-for="DateOfBirth" class="text-danger"></span>
39.                     </div>
40.                 </div>
41.
42.                 <div class="form-row">
43.                     <div class="col-lg-2">
44.                         <label asp-for="JoinDate" class="control-label"></label>
45.                     </div>
46.                     <div class="col-lg-4">
47.                         <input asp-for="JoinDate" class="form-control" />
48.                         <span asp-validation-for="JoinDate" class="text-danger"></span>
49.                     </div>
50.                 </div>
51.             </div>
52.         </form>
53.     </div>
54. </div>

```

```

48.
49.         <div class="form-row">
50.             <div class="col-lg-2">
51.                 <label asp-for="DepartmentId" class="control-label"></label>
52.             </div>
53.             <div class="col-lg-4">
54.                 <select asp-for="DepartmentId" asp-
items="@((new SelectList(ViewBag.Departments, "DepartmentId", "DepartmentName")))" class="form-
control">
55.                     <option value="">--Select--</option>
56.                 </select>
57.                 <span asp-validation-for="DepartmentId" class="text-danger"></span>
58.             </div>
59.             <div class="col-lg-2">
60.                 <label asp-for="DesignationId" class="control-label"></label>
61.             </div>
62.             <div class="col-lg-4">
63.                 <select asp-for="DesignationId" asp-
items="@((new SelectList(ViewBag.Designations, "DesignationId", "DesignationName")))" class="form-
control">
64.                     <option value="">--Select--</option>
65.                 </select>
66.                 <span asp-validation-for="DesignationId" class="text-danger"></span>
67.             </div>
68.         </div>
69.
70.         <div class="form-row">
71.             <div class="col-lg-2">
72.                 <label asp-for="Salary" class="control-label"></label>
73.             </div>
74.             <div class="col-lg-4">
75.                 <input asp-for="Salary" class="form-control" />
76.                 <span asp-validation-for="Salary" class="text-danger"></span>
77.             </div>
78.         </div>
79.         <div class="col-lg-4">
80.
81.
82.         </div>

```

```

85. <div class="form-row">
86.     <div class="col-lg-2">
87.         <label asp-for="Address" class="control-label"></label>
88.     </div>
89.     <div class="col-lg-4">
90.         <input asp-for="Address" class="form-control" />
91.         <span asp-validation-for="Address" class="text-danger"></span>
92.     </div>
93.     <div class="col-lg-2">
94.         <label asp-for="State" class="control-label"></label>
95.     </div>
96.     <div class="col-lg-4">
97.         <input asp-for="State" class="form-control" />
98.         <span asp-validation-for="State" class="text-danger"></span>
99.     </div>
100. </div>
101.
102. <div class="form-row">
103.     <div class="col-lg-2">
104.         <label asp-for="City" class="control-label"></label>
105.     </div>
106.     <div class="col-lg-4">
107.         <input asp-for="City" class="form-control" />
108.         <span asp-validation-for="City" class="text-danger"></span>
109.     </div>
110.     <div class="col-lg-2">
111.         <label asp-for="ZipCode" class="control-label"></label>
112.     </div>
113.     <div class="col-lg-4">
114.         <input asp-for="ZipCode" class="form-control" />
115.         <span asp-validation-for="ZipCode" class="text-danger"></span>
116.     </div>
117. </div>
118.
119. </div>
120. </div>
121. <div class="form-row">
122.     <div class="col-lg-3">
123.         <div>

```



```

126.
127.         <a asp-action="Index" class="btn btn-secondary"> Post      Ask Question
128.         </div>
129.     </div>
130. </div>
131. </form>
132. </div>
133. </div>
134.
135. @section Scripts {
136.     @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
137. }

```

Step 4

Now, return back to the EmployeesController.cs file and write down the method to save the data in the database. Note that, in this method we need to use `HttpPost` verb so that it can be fired on Form submit event from UI and it takes Employee data as argument. After successfully saving the data, we will automatically redirect to the Index UI.

```

01. [HttpPost]
02.     public IActionResult Create(Employee model)
03.     {
04.         if (ModelState.IsValid)
05.         {
06.             _dbContext.Employees.Add(model);
07.             _dbContext.SaveChanges();
08.             return RedirectToAction("Index");
09.         }
10.         this.GetModelData();
11.         return View();
12.     }

```

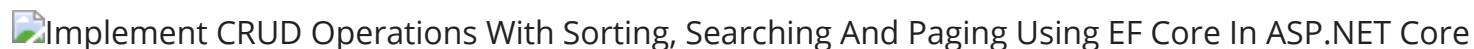
Now run the application and then click on **Create New Button** in the Employee Index UI.



After providing proper data in the form, click on the Submit button.

Step 7

Now, the Employee Index Page starts to display the information.



Step 8

In the Employee List, department and designation information is missing. But when we save the data, we provide the information for Department and Designation. This is because the Employee entity only contains DepartmentId and DesignationId. But we need to display a description related to those ids in the UI. For that purpose, we will need to make changes in the retrieval of Employee List in the Index() as below:

```
01. public IActionResult Index()
02.     {
03.         var employees = (from employee in this._dbContext.Employees
04.                         join design in this._dbContext.Designations on employee.DepartmentId equals design.DesignationId
05.                         join dept in this._dbContext.Departments on employee.DepartmentId equals dept.DepartmentId
06.                         select new Employee
07.                         {
08.                             Id = employee.Id,
09.                             EmployeeCode = employee.EmployeeCode,
10.                             EmployeeName = employee.EmployeeName,
11.                             Salary = employee.Salary,
12.                             Address = employee.Address,
13.                             State = employee.State,
14.                             City = employee.City,
```



```

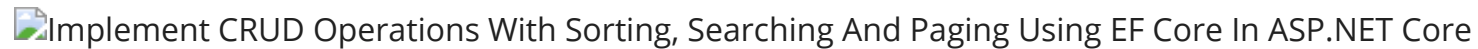
19.         DepartmentName = dept.DepartmentName,
20.         DesignationId = employee.DesignationId,
21.         DesignationName = desig.DesignationName
22.     }).ToList();
23.
24.     return View(employees);
25. }

```

[Post](#)
[Ask Question](#)

Step 9

Now, again run the application and check the Index UI of the Employee.



EDIT & DELETE EMPLOYEE DATA

Step 1

Now, we want to edit the existing employee data. For that, we have already created the "Edit" button in the Index UI. On click of the Edit, it will redirect to the Edit() of the controller. This method returns the same create.cshtml view.

```

01. public IActionResult Edit(int id)
02. {
03.     Employee data = this._dbContext.Employees.Where(p => p.Id == id).FirstOrDefault();
04.     this.GetModelData();
05.     return View("Create", data);
06. }

```

After making the modification on the existing data, we need to submit that data. For this, we need to call another `HttpPost` Method called `Edit()` in the controller.



```
03.         {  
04.             if (ModelState.IsValid)  
05.             {  
06.                 _dbContext.Employees.Update(model);  
07.                 _dbContext.SaveChanges();  
08.                 return RedirectToAction("Index");  
09.             }  
10.             this.GetModelData();  
11.             return View("Create", model);  
12.         }
```

Step 3

After submitting the change, related changes have been shown in the Index Page.

Step 4

Now, add another method called Delete() for deleting the existing data in the EmployeesController File.

```
01. public IActionResult Delete(int id)  
02.     {  
03.         Employee data = this._dbContext.Employees.Where(p => p.Id == id).FirstOrDefault();  
04.         if (data != null)  
05.         {  
06.             _dbContext.Employees.Remove(data);  
07.             _dbContext.SaveChanges();  
08.         }  
09.         return RedirectToAction("Index");  
10.     }
```

Step 5

Now, on clicking on the Delete button for any employee, that data will be deleted from the index list as well as the database.

Step 6



```
01. Create procedure uspGetEmployee
02. (
03.     @p_EmployeeId    INT = 0
04. )
05. As
06. BEGIN
07.     IF @p_EmployeeId > 0
08.     BEGIN
09.         Select * from DBO.Employee WHERE Id= @p_EmployeeId
10.     END
11.
12. END
13. GO
```

Step 7

Now, open the EFDataContext.cs file and add the below method to execute the stored procedure using Entity Framework.

```
01. public Employee GetEmployeeById(int employeeId)
02. {
03.     IQueryable<Employee> data = this.Employees.FromSql<Employee>(
04.         "Exec [dbo].uspGetEmployee " +
05.         "@p_EmployeeId", new SqlParameter("p_EmployeeId", employeeId));
06.
07.     if (data != null)
08.         return data.FirstOrDefault();
09.     else
10.         return new Employee();
11. }
```

Now, change the code of Edit() method as below.

```
01. public IActionResult Edit(int id)
02. {
```



```
05.         return View("Create", data);
06.     }
```

Step 9

Now, run the application and check all CRUD operations.

ADD SORTING IN EMPLOYEE LIST

Step 1

Now, to add sorting, we first need to make changes as below in the Index.cshtml pages.

```
01. <thead>
02.     <tr>
03.         <th style="width:10%;">
04.             @Html.ActionLink("Code", "Index", new { sortField = "EmployeeCode", currentSortField =
05.         </th>
06.         <th style="width:20%;">
07.             @Html.ActionLink("Employee Name", "Index", new { sortField = "EmployeeName", currentSortField =
08.         </th>
09.         <th style="width:10%;">
10.             @Html.ActionLink("Join Date", "Index", new { sortField = "JoinDate", currentSortField =
11.         </th>
12.         <th style="width:15%;">
13.             @Html.ActionLink("Department", "Index", new { sortField = "DepartmentName", currentSortField =
14.         </th>
15.         <th style="width:15%;">
16.             @Html.ActionLink("Designation", "Index", new { sortField = "DesignationName", currentSortField =
17.         </th>
18.         <th style="width:10%;">
19.             @Html.ActionLink("Salary", "Index", new { sortField = "Salary", currentSortField = View
20.         </th>
21.         <th style="width:10%;">
22.             @Html.ActionLink("City", "Index", new { sortField = "City", currentSortField = View
23.         </th>
24.     </tr>
25. </thead>
```

```
17.         </tr>
18.     </thead>
```

[Post](#)
[Ask Question](#)

Step 2

In the above code, we pass SortField, CurrentSortField and CurrentSortOrder to the Index Page so that we can sort the data in the controller and return to the view.

Step 3

So, we need to accept 3 arguments in the index method as below:

```
01. public IActionResult Index(string sortField, string currentSortField, string currentSortOrder)
02. {
03.     var employees = this.GetEmployeeList();
04.     return View(this.SortEmployeeData(employees, sortField, currentSortField, currentSortOrder));
05. }
06.
07. private List<Employee> SortEmployeeData(List<Employee> employees, string sortField, string currentSortOrder)
08. {
09.     if (string.IsNullOrEmpty(sortField))
10.     {
11.         ViewBag.SortField = "EmployeeCode";
12.         ViewBag.SortOrder = "Asc";
13.     }
14.     else
15.     {
16.         if (currentSortField == sortField)
17.         {
18.             ViewBag.SortOrder = currentSortOrder == "Asc" ? "Desc" : "Asc";
19.         }
20.         else
21.         {
22.             ViewBag.SortOrder = "Asc";
23.         }
24.         ViewBag.SortField = sortField;
```




```
27.     var propertyInfo = typeof(Employee).GetProperty(ViewBag.SortField);
28.     if (ViewBag.SortOrder == "Asc")
29.     {
30.         employees = employees.OrderBy(s => propertyInfo.GetValue(s, null)).ToList();
31.     }
32.     else
33.     {
34.         employees = employees.OrderByDescending(s => propertyInfo.GetValue(s, null)).ToList();
35.     }
36.     return employees;
37. }
```

Step 4

Now, in the above code, we create another method called SortEmployeeData() which stored the Current Sort Field and SortOrder in the View Bag. Also, this method is used to find the property name and then sort that data accordingly.

Step 5

Now, run the application and check the index page of employee.

 Implement CRUD Operations With Sorting, Searching And Paging Using EF Core In ASP.NET Core

ADD SEARCH BOX IN THE INDEX PAGE

Step 1

Now, for adding the search functionality, we need to add the below code in the Index.cshtml page just above the Index table.

```
01. <p>
02.     <a asp-action="Create">Create New</a>
03. </p>
04. @using (Html.BeginForm())
05. {
```

```
08.         <input type="submit" value="Search" />
09.     </p>
10. }
11. <table class="table">
12.     <thead>
13.         <tr>
14.             <th style="width:10%;">
15.                 @Html.ActionLink("Code", "Index", new { sortField = "EmployeeCode", currentSortField
16.                 @*@Html.DisplayNameFor(model => model.EmployeeCode)*@
17.             </th>
```


Step 2

Now make the below changes in the Index() of the EmployeesController.cs

```
01. public IActionResult Index(string sortField, string currentSortField, string currentSortOrder, string
02.     {
03.         var employees = this.GetEmployeeList();
04.         if (!String.IsNullOrEmpty(SearchString))
05.         {
06.             employees = employees.Where(s => s.EmployeeName.Contains(SearchString)).ToList();
07.         }
08.         return View(this.SortEmployeeData(employees, sortField, currentSortField, currentSortOrder
09.     }
```

Step 3

Now run the application and check

 Implement CRUD Operations With Sorting, Searching And Paging Using EF Core In ASP.NET Core

ADD PAGING IN THE INDEX PAGE



First Select the DataContext project and add a folder called Paging.

[Post](#)[Ask Question](#)

Step 2

In that folder, add a file called PagingList.cs file and add the below code

```
01. using Microsoft.EntityFrameworkCore;
02. using System;
03. using System.Collections.Generic;
04. using System.Linq;
05. using System.Text;
06. using System.Threading.Tasks;
07.
08. namespace DataContext.Paging
09. {
10.     public class PagingList<T> : List<T>
11.     {
12.         public int PageIndex { get; private set; }
13.         public int TotalPages { get; private set; }
14.
15.         public PagingList(List<T> items, int count, int pageIndex, int pageSize)
16.         {
17.             PageIndex = pageIndex;
18.             TotalPages = (int)Math.Ceiling(count / (double)pageSize);
19.
20.             this.AddRange(items);
21.         }
22.
23.         public bool HasPreviousPage
24.         {
25.             get
26.             {
27.                 return (PageIndex > 1);
28.             }
29.         }
30.
31.         public bool HasNextPage
```



```
34.         {
35.             return (PageIndex < TotalPages);
36.         }
37.     }
38.
39.     public int TotalPageNo
40.     {
41.         get
42.         {
43.             return TotalPages;
44.         }
45.     }
46.
47.     public static PagingList<T> CreateAsync(IQueryable<T> source, int pageIndex, int pageSize)
48.     {
49.         var count = source.Count();
50.         var items = source.Skip((pageIndex - 1) * pageSize).Take(pageSize).ToList();
51.         return new PagingList<T>(items, count, pageIndex, pageSize);
52.     }
53. }
54. }
```

Step 3

Now change the below code in the declaration section of the index.cshtml page

```
01. @model DataContext.Paging.PagingList<DataContext.Models.Employee>
02.
03. <link href="~/Content/PagedList.css" rel="stylesheet" type="text/css" />
04.
05. @{
06.     ViewData["Title"] = "Employee List";
07. }
```

Step 4

Now change the below code in the header part of the index.cshtml page

```

03.         <th style="width:10%;">
04.             @Html.ActionLink("Code", "Index", new { sortField = "Emp1 Post " Ask Question Field =
05.         </th>
06.         <th style="width:20%;">
07.             @Html.ActionLink("Employee Name", "Index", new { sortField = "EmployeeName", currentSc
08.         </th>
09.         <th style="width:10%;">
10.             @Html.ActionLink("Join Date", "Index", new { sortField = "JoinDate", currentSortField
11.         </th>
12.         <th style="width:15%;">
13.             @Html.ActionLink("Department", "Index", new { sortField = "DepartmentName", currentSor
14.         </th>
15.         <th style="width:15%;">
16.             @Html.ActionLink("Designation", "Index", new { sortField = "DesignationName", currentS
17.         </th>
18.         <th style="width:10%;">
19.             @Html.ActionLink("Salary", "Index", new { sortField = "Salary", currentSortField = Vie
20.         </th>
21.         <th style="width:10%;">
22.             @Html.ActionLink("City", "Index", new { sortField = "City", currentSortField = ViewBag
23.         </th>
24.         <th style="width:10%;"></th>
25.     </tr>
26. </thead>

```

Step 5

Now add the below code at the end of the Index.cshtml page.

```

01. <br />
02.
03. @{
04.     var prevDisabled = !Model.HasPreviousPage ? "disabled" : "";
05.     var nextDisabled = !Model.HasNextPage ? "disabled" : "";
06.     var totalPagesNo = Model.TotalPageNo;
07.     var currentPageNo = Model.PageIndex;
08. }

```




```

11.     asp-route-sortOrder="@ViewData["CurrentSort"]"
12.     asp-route-pageNo="@((Model.PageIndex - 1))"
13.     asp-route-currentFilter="@ViewData["CurrentFilter"]"
14.     class="btn btn-default @prevDisabled">
15.         Previous
16. </a>
17. <a asp-action="Index"
18.     asp-route-sortOrder="@ViewData["CurrentSort"]"
19.     asp-route-pageNo="@((Model.PageIndex + 1))"
20.     asp-route-currentFilter="@ViewData["CurrentFilter"]"
21.     class="btn btn-default @nextDisabled">
22.         Next
23. </a>
24. <span>
25.     Page No : @currentPageNo of @totalPageNo
26. </span>

```

Step 4


Now, make the below changes in the Index() in the EmployeesController.cs method

```

01. public IActionResult Index(string sortField, string currentSortField, string currentSortOrder, string
02.     int ? pageNo) {
03.     var employees = this.GetEmployeeList();
04.     if (SearchString != null) {
05.         pageNo = 1;
06.     } else {
07.         SearchString = currentFilter;
08.     }
09.     ViewData["CurrentSort"] = sortField;
10.     ViewBag.CurrentFilter = SearchString;
11.     if (!String.IsNullOrEmpty(SearchString)) {
12.         employees = employees.Where(s => s.EmployeeName.Contains(SearchString)).ToList();
13.     }
14.     employees = this.SortEmployeeData(employees, sortField, currentSortField, currentSortOrder);
15.     int pageSize = 10;
16.     return View(PagingList < Employee > .CreateAsync(employees.AsQueryable < Employee > (), pageNo ?
17. }

```

Now run the application.

 Implement CRUD Operations With Sorting, Searching And Paging Using EF Core In ASP.NET Core

Download Index List Data as Excel File

Now we need to download the Index List data as excel file. For that purpose, we first need to install a package called EPPlus 4.5.3 version from Nuget Package. Now insert the below code into Index.cshtml file -

```
01. @*@model IEnumerable<DataContext.Models.Employee>*@
02.
03. @model DataContext.Paging.PagingList<DataContext.Models.Employee>
04.
05. @{
06.     ViewData["Title"] = "Employee List";
07. }
08.
09. <h1>Index</h1>
10.
11. <p>
12.     <a asp-action="Create">Create New</a>
13. </p>
14. <div class="text-
15.     right" style="overflow:hidden;position:absolute;padding:1px 0px 0px 0px;right: 350px;">
16.     <button type="button" class="btn btn-
17.     info" onclick="location.href='@Url.Action("ExportToExcel", "Employees")'">Export To Excel</button>
18. </div>
19. @using (Html.BeginForm("Index", "Employees", FormMethod.Get))
20. {
21.     <p>
22.         Find by Employee Name: @Html.TextBox("SearchString", ViewBag.CurrentFilter as string)
23.         <input type="submit" value="Search" />
24.     </p>
25. }
```

```
01. public IActionResult ExportToExcel()
02.     {
03.         var employees = this.GetEmployeeList();
04.         byte[] fileContents;
05.
06.         ExcelPackage Ep = new ExcelPackage();
07.         ExcelWorksheet Sheet = Ep.Workbook.Worksheets.Add("EmployeeInfo");
08.         Sheet.Cells["A1"].Value = "Employee Code";
09.         Sheet.Cells["B1"].Value = "Employee Name";
10.         Sheet.Cells["C1"].Value = "Join Date";
11.         Sheet.Cells["D1"].Value = "Department";
12.         Sheet.Cells["E1"].Value = "Designation";
13.         Sheet.Cells["F1"].Value = "Salary";
14.         Sheet.Cells["G1"].Value = "City";
15.
16.         int row = 2;
17.         foreach (var item in employees)
18.         {
19.             Sheet.Cells[string.Format("A{0}", row)].Value = item.EmployeeCode;
20.             Sheet.Cells[string.Format("B{0}", row)].Value = item.EmployeeName;
21.             Sheet.Cells[string.Format("C{0}", row)].Value = item.JoinDate;
22.             Sheet.Cells[string.Format("D{0}", row)].Value = item.DepartmentName;
23.             Sheet.Cells[string.Format("E{0}", row)].Value = item.DesignationName;
24.             Sheet.Cells[string.Format("F{0}", row)].Value = item.Salary;
25.             Sheet.Cells[string.Format("G{0}", row)].Value = item.City;
26.             row++;
27.         }
28.
29.
30.         Sheet.Cells["A:AZ"].AutoFitColumns();
31.         Response.Clear();
32.         Response.ContentType = "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet";
33.         fileContents = Ep.GetAsByteArray();
34.
35.         if (fileContents == null || fileContents.Length == 0)
36.         {
37.             return NotFound();
```


[Come a member](#)
[Login](#)

```

40.         return File(
41.             fileContents: fileContents,
42.             contentType: "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",
43.             fileName: "Employee.xlsx"
44.         );
45.     }

```

[Post](#)
[Ask Question](#)

Now, run the result and check the output -

Conclusion

In this article, we discussed how to implement basic CRUD operations including sorting data and paging in the index page. I hope this will help the reader to understand how to use entity framework core in real life applications. Suggestions and feedback related to this article or any subject are most welcome. For any further query or clarification, ping me.

Next Recommended Article

[CRUD Operation In ASP.NET Core 2.0 Using Dapper ORM](#)

[ASP.NET Core](#)
[Data download as Excel File in .Net Core](#)
[EF Core In ASP.NET Core](#)
[Excel file download](#)
[Implement CRUD Operations](#)
[Implement CRUD Operations With Sorting](#)
[Searching And Paging](#)
[Searching And Paging Using EF Core](#)
[Sorting](#)

Debasis Saha *TOP 50*

Tech Lead | CSM | C# Corner MVP (5 Times) | DZone MVB | Technical Author | .Net | .Net Core | Angular | Full Stack | Angular Expert | Cloud Enthusiast | Speaker | Chapter Lead For any technical guidance, can contact me i... [Read more](#)


[Come a member](#)
[Login](#)

<https://github.com/debasis-saha>
[Post](#)
[Ask Question](#)

11 9.3m 5

12 21



Type your comment here and press Enter Key (Minimum 10 characters)



Thanks are very useful information. Please design the cascading dropdown list article in Angular with web api core a and database sql server 2008

[Ramzanali Momin](#)

1356 550 584

Nov 06, 2019

1 1 Reply



Definitely Ramzanali, I will release an article related that very soon.

[Debasis Saha](#)

11 49.8k 9.3m

Nov 06, 2019

0



Very useful article....

[Chittaranjan Swain](#)

70 23.3k 194.6k

Oct 31, 2019

1 0 Reply



Thanx Debassis Saha for guide me to export the Index view to Microsoft Excel. I believe many others will also benefit from this. Thanx very much.

[Maurice Aboagye](#)

1610 290 289

Oct 25, 2019

2 0 Reply



Hello Debassis Saha. PLease guide me to export the results of the Index View In This Project to Microsoft Excel. Thanx in advance.

[Maurice Aboagye](#)

1610 290 289

Oct 10, 2019

2 1 Reply

As per ur requirement i will make the change and update u very shortly.


[Come a member](#)
[Login](#)
[Post](#)
[Ask Question](#)


Thanx vey much Debassis Saha. You ve shed the light

[Maurice Aboagye](#)

1610 290 289

Sep 16, 2019

2 1 Reply



Most welcome Maurice...

[Debasis Saha](#)

11 49.8k 9.3m

Sep 16, 2019

1



Hello, I tried to launch your project but it gives me HTTP Error 502.5 - Process Failure

[Paolo Scotti](#)

1896 4 0

Aug 30, 2019

1 1 Reply



I think you need to run the database related migration command first. for that Select the DataContext Project in Package Console and then run the migration command.

[Debasis Saha](#)

11 49.8k 9.3m

Aug 30, 2019

1



Can you provide a function for search box witch if no result a modal open and ask if you want to add? ex : search for a city and if no data found ..result "This city not exist .Do you want to crate? Yes / No.and if press yes open modal with create city or any else ?

[alex z](#)

1770 130 0

Aug 29, 2019

2 1 Reply



Good idea Alex. Definitely, I will do that and update the article.

[Debasis Saha](#)

11 49.8k 9.3m

Aug 29, 2019

1



Thank you very much for your help graciassss

[Heràclides Morales](#)

1833 67 207

May 03, 2019

2 1 Reply



Most welcome....

[Debasis Saha](#)

11 49.8k 9.3m

May 03



Heràclides Morales

1833 67 207

:come a member

Login

Post

Ask Question

May 03, 2019

Reply 4



Select Data Project in the Default Project Drop Down

Debasis Saha

11 49.8k 9.3m

May 03, 2019

2



Then Run add migration migration1

Debasis Saha

11 49.8k 9.3m

May 03, 2019

2



Then run update database

Debasis Saha

11 49.8k 9.3m

May 03, 2019

2



Muy buen ejemplo. por favor me podrian ayudar con este error No DbContext was found in assembly 'EF_Crud_Samples'. Ensure that you're using the correct assembly and that the type is neither abstract nor generic.

Heràclides Morales

1833 67 207

May 03, 2019

1 1 Reply



Select the DataContext Project in Package Console and then run the migration command.

Debasis Saha

11 49.8k 9.3m

May 03, 2019

1

FEATURED ARTICLES

Agile Or Scrum - Which One To Choose And Why?

Database Operations in ASP.NET Core Web API Using ADO.NET

Search Data Between Two Dates Using Web API And Angular 9

Build Low-Code and No-Code Solutions With PowerApps

[Come a member](#)[Login](#)[Post](#)[Ask Question](#)[View All](#)

TRENDING UP

- 01 Agile Or Scrum - Which One To Choose And Why?
- 02 Performing Update and Delete Operations in ASP.NET Core 3 Razor Pages Using Microsoft SQL Server
- 03 Getting Started With .NET 5.0
- 04 Inserting Data into SQL Server Database Using ASP.NET Core 3 Razor Pages
- 05 Dependency Injection Lifetimes In ASP.NET CORE
- 06 Getting Started with ML.NET Model Builder for Using in ASP.NET Core
- 07 ASP.NET Core API Analyzer
- 08 Entity Framework Core And Its Data Modelling Approaches
- 09 How To Use Async Pipe In Angular 8
- 10 Introduction To Building ASP.NET Core And Angular Web Application

[View All](#)[About Us](#) [Contact Us](#) [Privacy Policy](#) [Terms](#) [Media Kit](#) [Sitemap](#) [Report a Bug](#) [FAQ](#) [Partners](#)[C# Tutorials](#) [Common Interview Questions](#) [Stories](#) [Consultants](#) [Ideas](#) [Certifications](#)

©2020 C# Corner. All contents are copyright of their authors.