CRUD Operations In ASP.NET Core MVC (.NET 5.0)

Achyuta Jha Updated date Jul 20, 2021 82.3k 3 9

ASP.NETCORE5.0CRUD.zip
Download Free .NET & JAVA Files API

Introduction

In this article, we will learn CRUD Operations in ASP.NET Core 5.0. We will use Entity Framework Core 5.0 to interact with sql-server database and for performing CRUD operations. Before moving ahead, i am going to give you a brief introduction to .NET 5.0

What is .NET 5.0 ?

The .NET 5.0 is the major release of .NET Core after .Net Core 3.1. We can say that .NET 5 = .NET Core vNext. In .NET 5 lot of new .NET APIs, runtime capabilities and language features has been added.

The main reason behind presenting .NET 5 is to produce a single .NET runtime and framework that can be used everywhere and that has uniform runtime behaviors.

But the question is why .NET 5.0, why not .NET Core 4.0 ?

There are two main reasons:

1. Microsoft skipped version numbers 4.x to avoid confusion with .NET Framework 4.x.
2. Microsoft dropped the word "Core" from the name to emphasize that this is the main implementation of .NET going forward.

To Avoid Confusion:

1. ASP.NET Core 5.0 is based on .NET 5.0 but retains the name "Core" to avoid confusing it with ASP.NET MVC 5.
2. Entity Framework Core 5.0 retains the name "Core" to avoid confusing it with Entity

Framework 5.

So we can say that, there is no .NET Core 5.0, now everything falls under one umbrella, which is .NET 5. The whole idea is to bring all .NET runtimes into a single .NET platform with unified base class libraries (BCL) for all kinds of applications like ASP.NET Core, Windows Forms, WPF, Blazor etc.

I Will Cover The Following Points In This Article:

1. Basic prerequisites to run .Net core 5.0 project.
2. How to create ASP.NET CORE 5.0 project in visual studio.
3. ASP.NET CORE 5.0 project structure.
4. How to install all the necessory packages from Nuget.
5. Create Database and required tables for CRUD operation.
6. Create model and context class from an existing database.
7. Create Employee controller.
8. Miscellaneous Configuration to run ASP.NET CORE 5.0 project.
9. Implement ASP.NET Core MVC CRUD Operations.
10. Validations in ASP.NET Core MVC.
11. Run the application.
12. Instructions to download and run the project.
13. Summary.

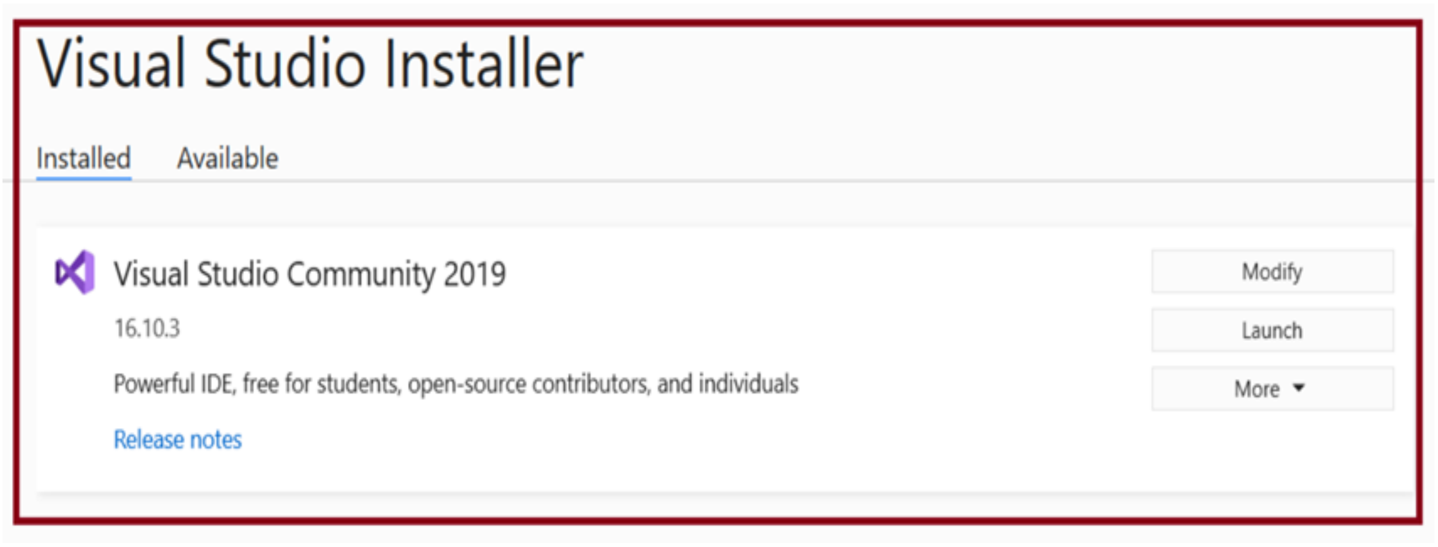
Step 1 - Prerequisites

1. Visual Studio 2019 latest version(at least 16.6.0).

2. .NET SDK 5.0 or later.

3. Sql-Server 2017 or later version.

<u>Note</u>
In case your visual studio version is lower then mentioned above, you can upgrade using visual studio installer.

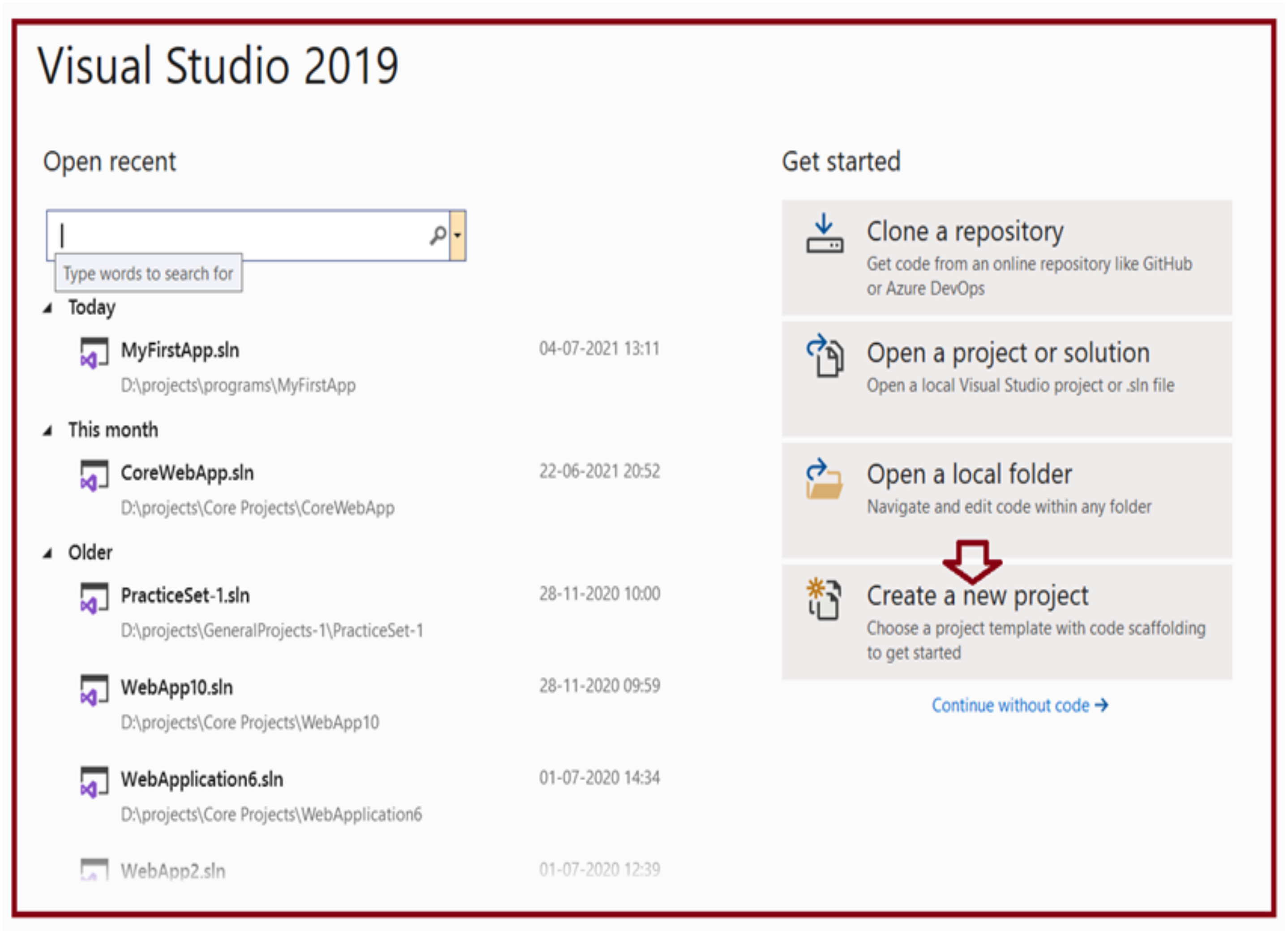

Visual Studio Installer

Visual Studio Community 2019
16.10.3
Powerful IDE, free for students, open-source contributors, and individuals

You will have the option to update in case it is not up to date.

Step 2 - Create ASP.NET Core 5.0 Project.

Open Visual Studio and click on **"Create a new project".**


Visual Studio 2019

Open recent

Today

MyFirstApp.sln D:\projects\programs\MyFirstApp	04-07-2021 13:11

This month

CoreWebApp.sln D:\projects\Core Projects\CoreWebApp	22-06-2021 20:52

Older

PracticeSet-1.sln D:\projects\GeneralProjects-1\PracticeSet-1	28-11-2020 10:00
WebApp10.sln D:\projects\Core Projects\WebApp10	28-11-2020 09:59
WebApplication6.sln D:\projects\Core Projects\WebApplication6	01-07-2020 14:34
WebApp2.sln	01-07-2020 12:39

Get started

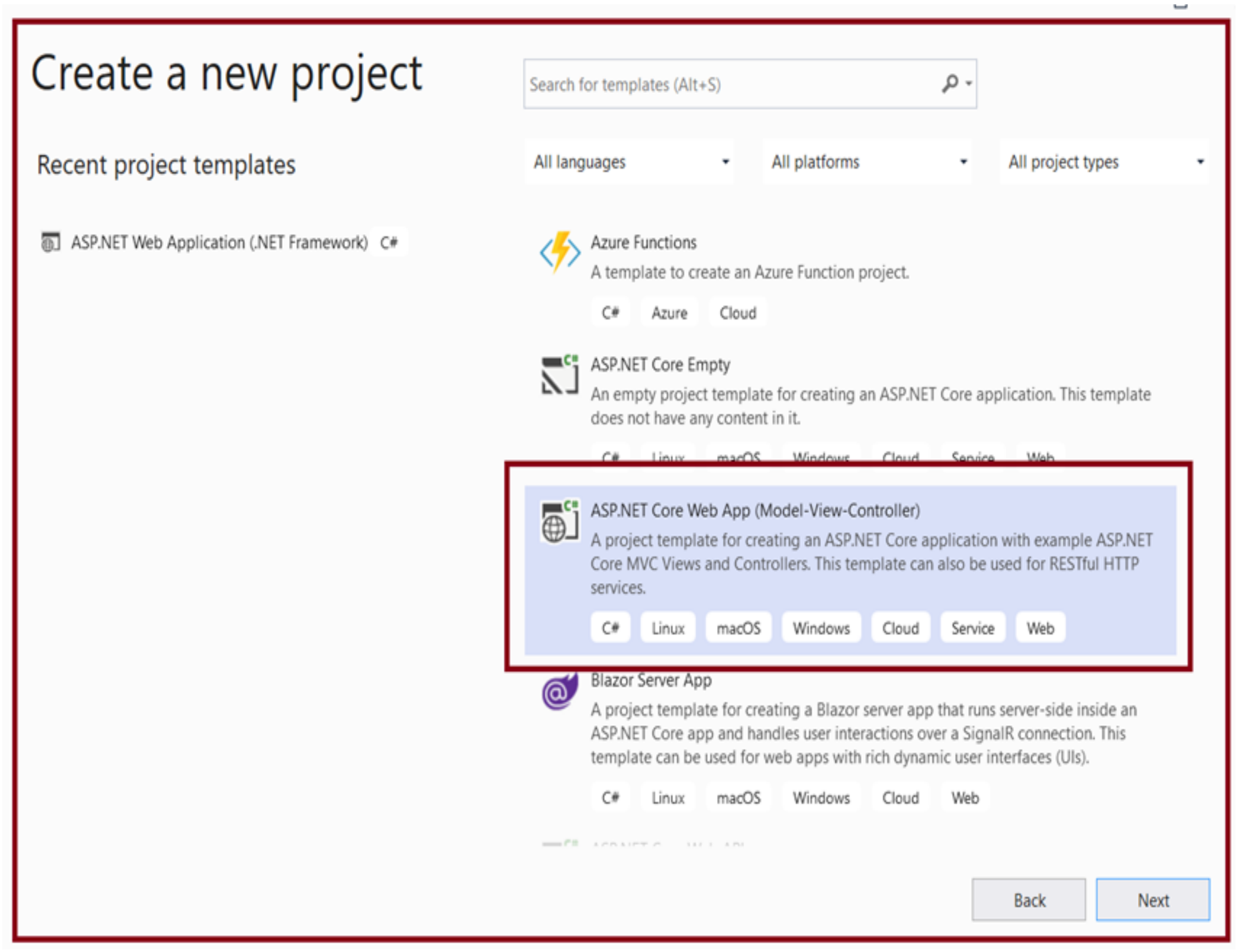
Clone a repository
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Select the ASP.NET Core Web App(Model-View-Controller) as a project template and click **Next**.



Create a new project

Search for templates (Alt+S)

Recent project templates

ASP.NET Web Application (.NET Framework) C#

All languages | All platforms | All project types

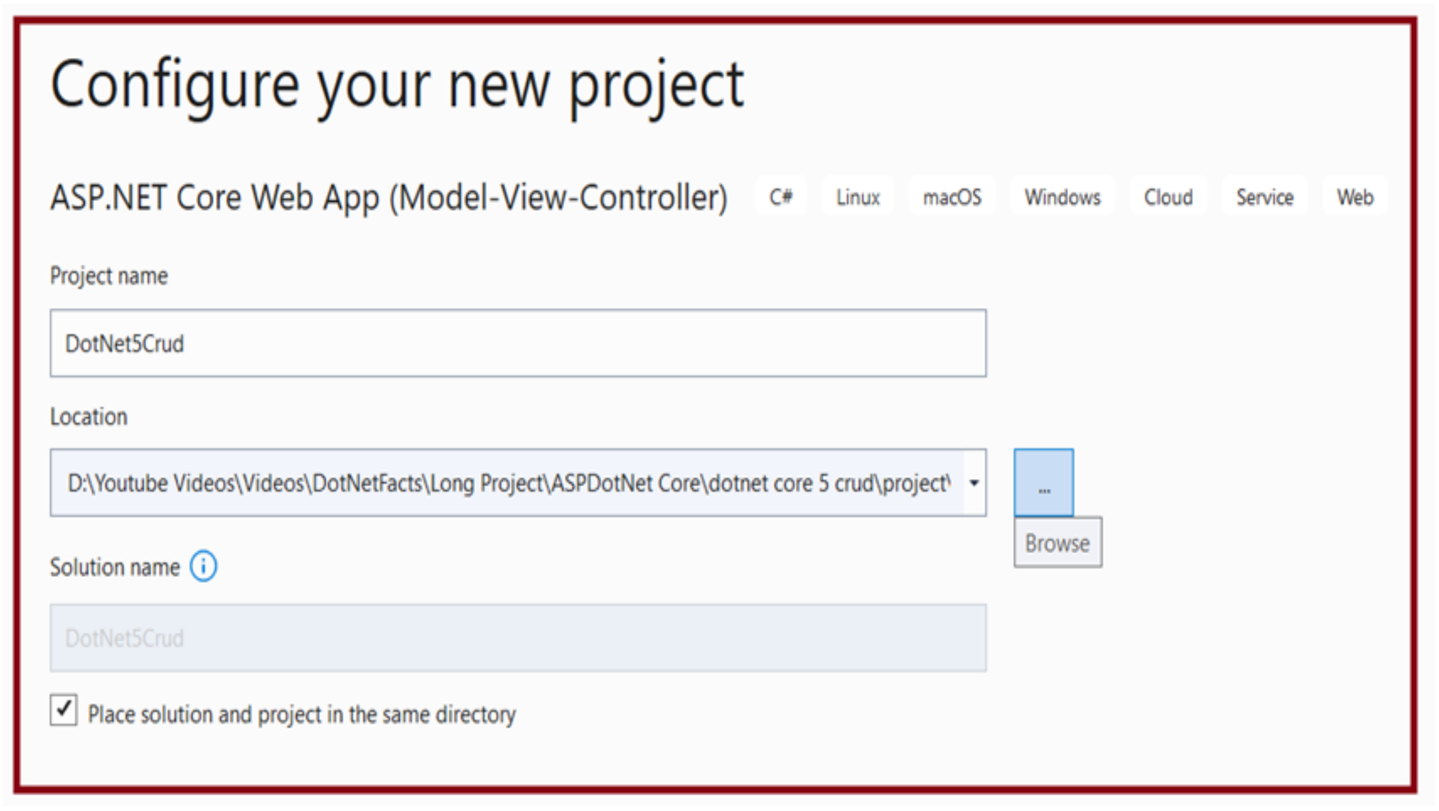
Azure Functions
A template to create an Azure Function project.
C# Azure Cloud

ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App (Model-View-Controller)
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.
C# Linux macOS Windows Cloud Service Web

Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

Enter the Project name and click **Next.**


Configure your new project
ASP.NET Core Web App (Model-View-Controller)
C#
Linux
macOS
Windows
Cloud
Service
Web
Browse

Project name

DotNet5Crud

Location

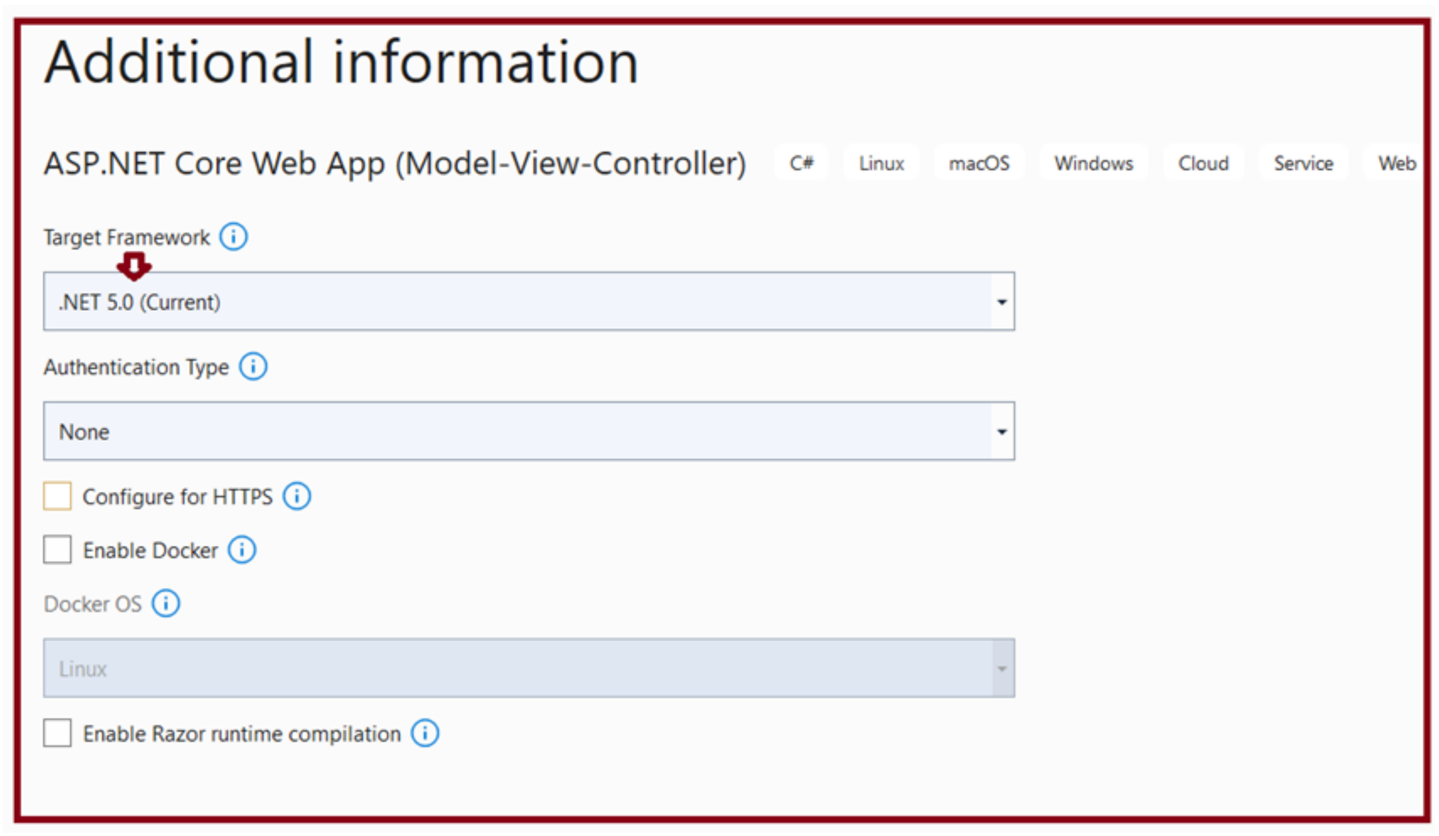
D:\Youtube Videos\Videos\DotNetFacts\Long Project\ASPDotNet Core\dotnet core 5 crud\project\

Solution name

DotNet5Crud

[✓] Place solution and project in the same directory

In additional information, select the fields as configured below and click on **Create**.



Target Framework

.NET 5.0 (Current)

Authentication Type

None

[ ] Configure for HTTPS

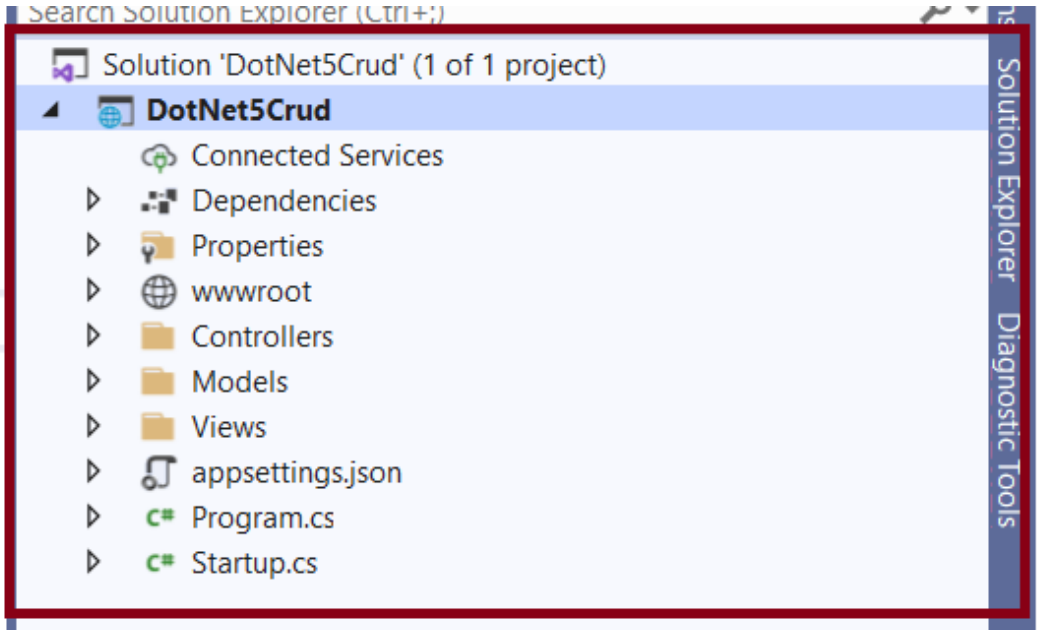
[ ] Enable Docker

Docker OS

Linux

[ ] Enable Razor runtime compilation

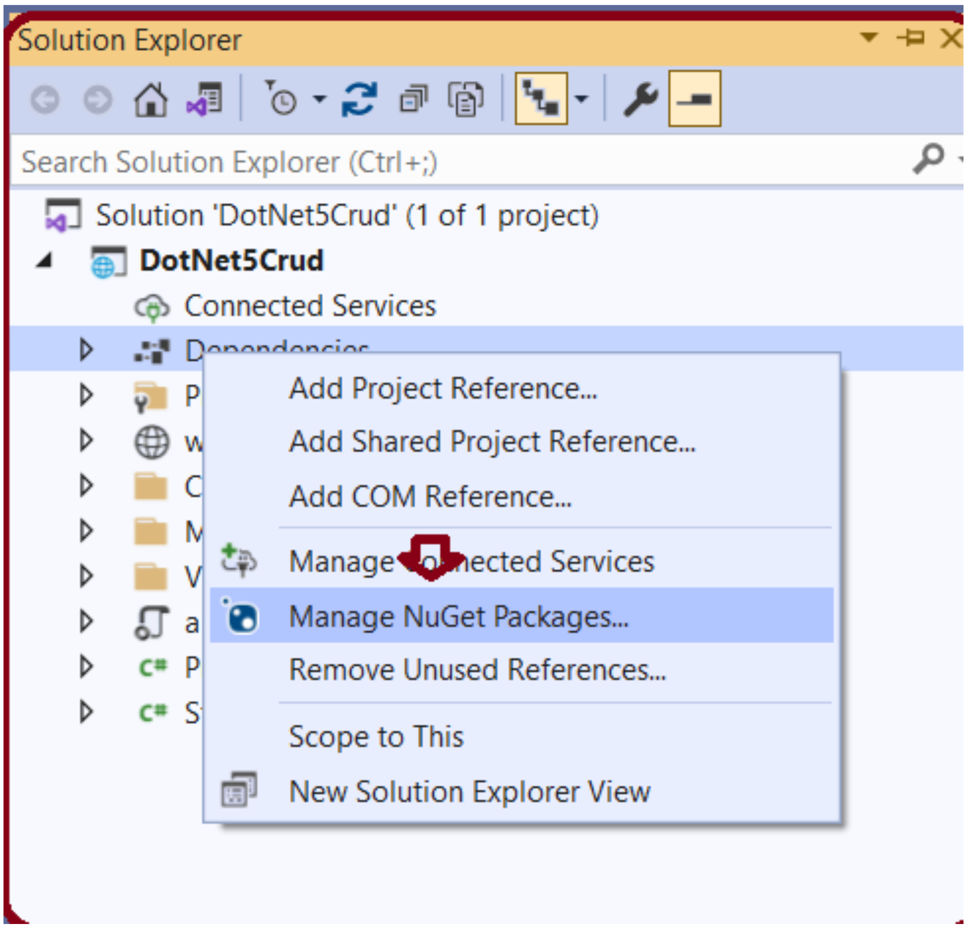
Step 3 - ASP.NET CORE 5.0 Project Structure



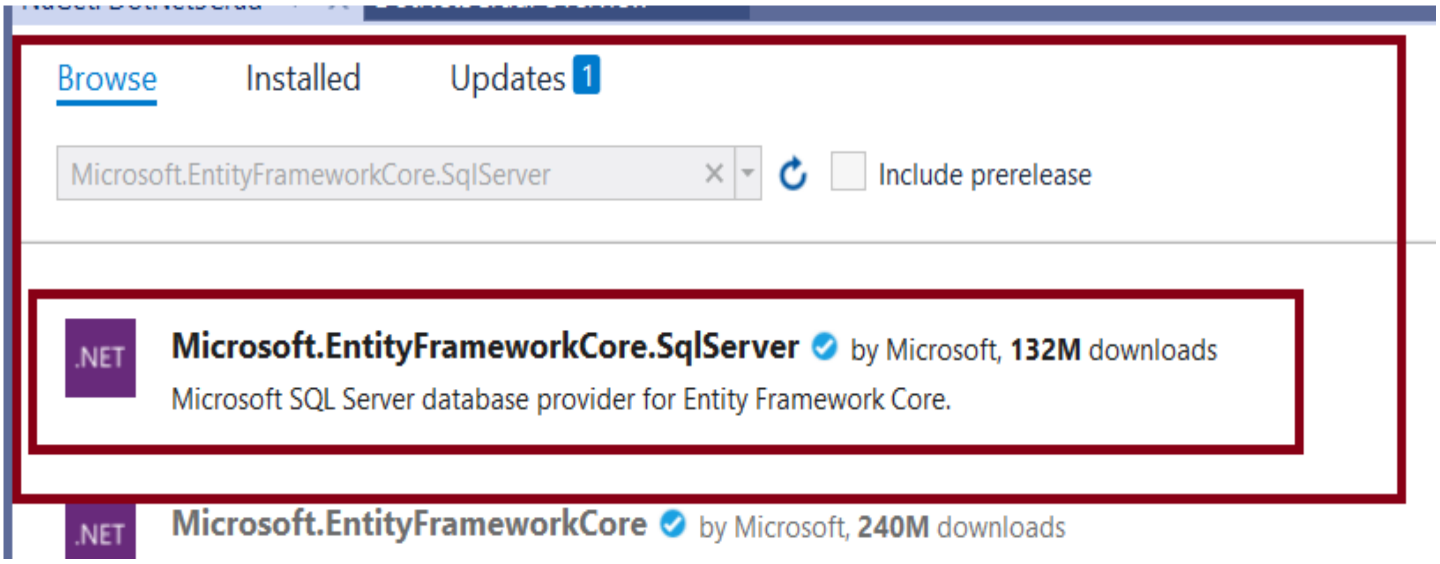
- **Dependencies:** It contains all the installed NuGet packages. We can manage the NuGet packages by right clicking on it.
- **Properties:** It contains launchSettings.json file which has visual studio profiles, iis and debug setting.
- **wwwroot folder:** It is the web root folder of asp.net core application where we can put all the static files such as javascript , css , images.
- **Controllers:** It contails all the controller class we create in our asp.net core mvc application.
- **Models:** We can put all the model or view model classes inside this folder.
- **Views:** We can add views for certain actions methods inside view folder. There will be seperate folder for each view we create inside Views folder.
- **appsettings.json:** It is the application configuration file which is used to store configuration settings i.e connections strings of the database, global variables etc.
- **Program.cs :** Initially asp.net core application starts as a console application. In the **Main** method it calls the **CreateWebHostBuilder()** method that configures the asp.net core setting and launch it as asp.net core application.
- **Startup.cs:** It contains the ConfigureServices() and Configure methods. As the name implies ConfigureServices() method configures all the services which are going to used by the application. Configure method take care of all the request processing pipelines.

Step 4 - Install All Necessary Packages From NuGet

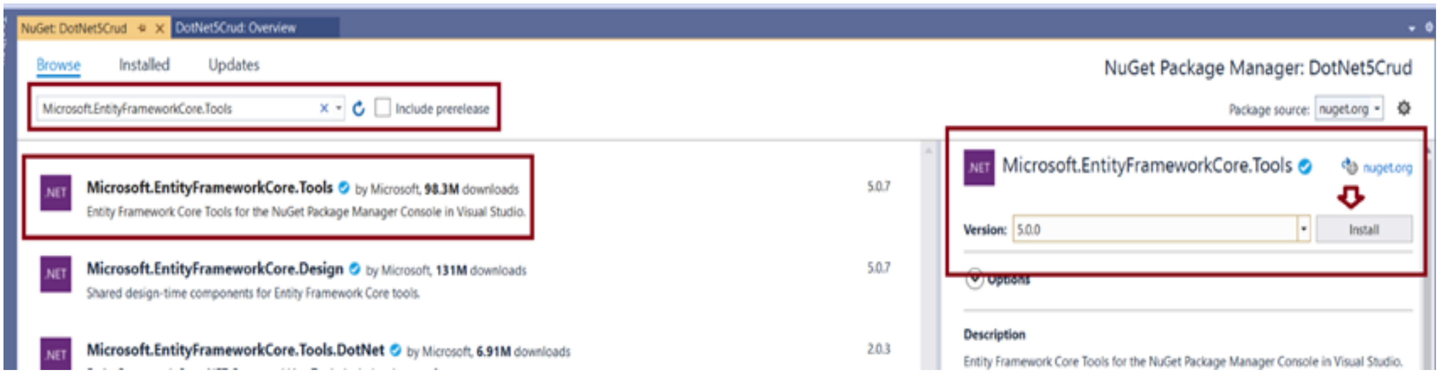
Right-click on **Dependencies** and then **Manage NuGet Package.**


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DotNet5Crud' (1 of 1 project)
DotNet5Crud
Connected Services
Dependencies
Add Project Reference...
Add Shared Project Reference...
Add COM Reference...
Manage Connected Services
Manage NuGet Packages...
Remove Unused References...
Scope to This
New Solution Explorer View

In order to access the MS SQL Server database, we need to install the below provider. Search **"Microsoft.EntityFrameworkCore.SqlServer"** as below and **install.**


Browse
Installed
Updates 1
Microsoft.EntityFrameworkCore.SqlServer
Include prerelease
Microsoft.EntityFrameworkCore.SqlServer by Microsoft, 132M downloads
Microsoft SQL Server database provider for Entity Framework Core.
Microsoft.EntityFrameworkCore by Microsoft, 240M downloads

To execute EF Core commands we will require EF-Core tools. Search "**Microsoft.EntityFrameworkCore.Tools**" as below and **install**.



Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.Tools by Microsoft, 98.3M downloads 5.0.7
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Microsoft.EntityFrameworkCore.Design by Microsoft, 131M downloads 5.0.7
Shared design-time components for Entity Framework Core tools.

Microsoft.EntityFrameworkCore.Tools.DotNet by Microsoft, 6.91M downloads 2.0.3

Microsoft.EntityFrameworkCore.Tools nuget.org
Version: 5.0.0

Description
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

This package will allow us to execute scaffold , migration commands from **package manager console** (PMC). It will also help you to create database context and the model classes.

Step 5 - Create Database And Employee Table

Create a new database named CompanyDB in sql-server and execute the below SQL query to create employee table.

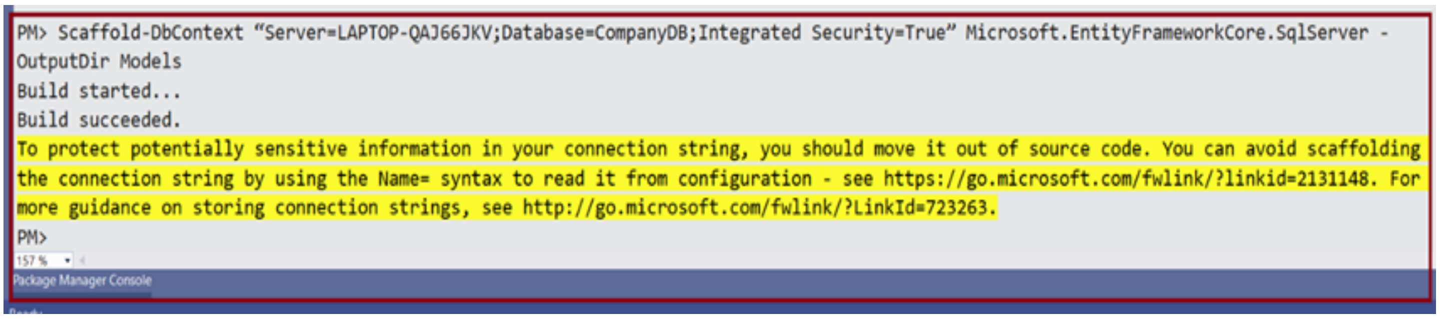
```sql
CREATE TABLE [dbo].[Employees](
[EmployeeId] [int] IDENTITY(1,1) NOT NULL,
[Name] [varchar](50) NOT NULL,
[Address] [varchar](250) NULL,
[Designation] [varchar](50) NULL,
[Salary] [decimal](18, 0) NOT NULL,
[JoiningDate] [datetime] NOT NULL,
PRIMARY KEY CLUSTERED
(
[EmployeeId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Employees] ADD  DEFAULT (getdate()) FOR [JoiningDa
GO
```

Step 6 - Create Model And Context Class From An Existing Database.

Creating model and context class from an existing database is also called Database-First approach. So to reverse engineer we need to execute **Scaffold-DbContext** command. This scaffold command will create models and context classes based on the database schema.

Run the below scaffold command after replacing server name, database name with your applications connection setting.

```
Scaffold-DbContext “Server=******;Database=ComapnyDB;Integrated Secur
```



```
PM> Scaffold-DbContext "Server=LAPTOP-QAJ66JKV;Database=CompanyDB;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
Build started...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723263.
PM>
```

Above scaffold, commands have three parameters.

First parameter of the scaffold command have server name, database name and integrated security information.

```
Server=******;Database=ComapnyDB;Integrated Security=True
```

Second parameter have information about provider. we are using sql-server so provider will be Microsoft.EntityFrameworkCore.SqlServer.

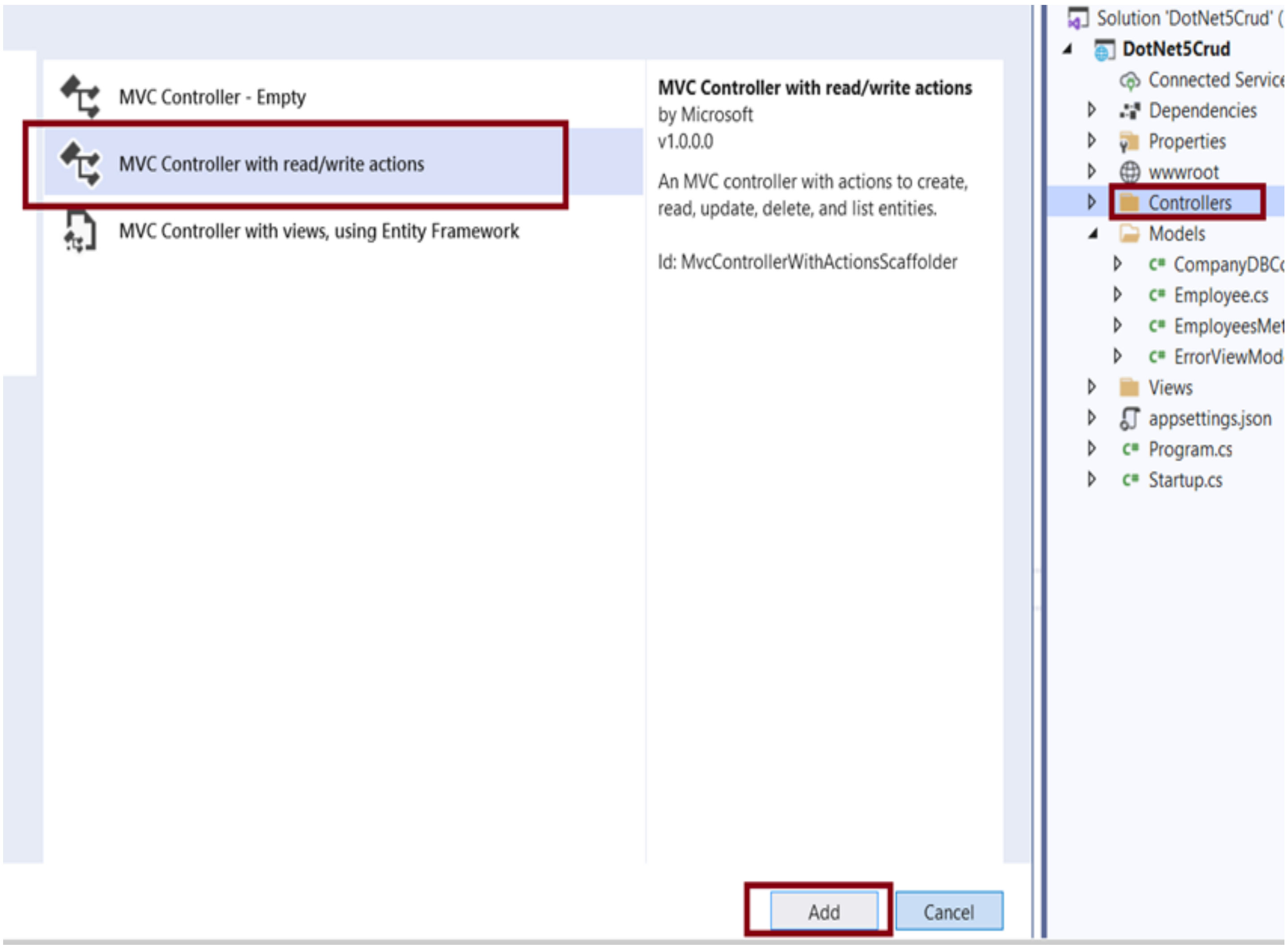
```
Microsoft.EntityFrameworkCore.SqlServer
```

Third parameter i.e **-OutputDir** is use to specify the location where we want to generate model classes. In our cases it is **Models** folder.

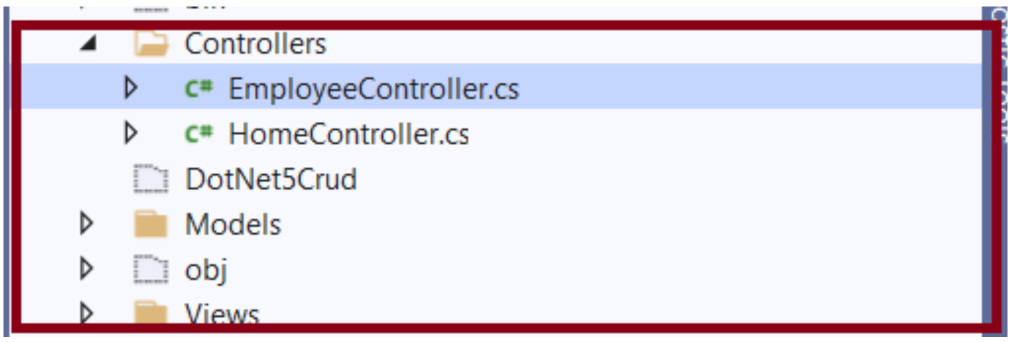
```
-OutputDir Models
```

Step 7 - Create Employee Controller

On the controller folder, Right-click and then **Add > Controller**. Select the controller template as highlighted below and Click **Add.** In the dialog, provided next type name of the controller, in our case it is **EmployeeController.cs** and click **Add** to create **EmployeeController** class under controller folder.


MVC Controller - Empty
MVC Controller with read/write actions
MVC Controller with views, using Entity Framework
MVC Controller with read/write actions
by Microsoft
v1.0.0.0
An MVC controller with actions to create, read, update, delete, and list entities.
Id: MvcControllerWithActionsScaffolder
Add
Cancel
Solution 'DotNet5Crud' (
DotNet5Crud
Connected Service
Dependencies
Properties
wwwroot
Controllers
Models
CompanyDBC
Employee.cs
EmployeesMet
ErrorViewMod
Views
appsettings.json
Program.cs
Startup.cs

Finally the employee controller class has been created with basic auto generated code for crud operation.



Step 8 - Miscellaneous Configuration

Store connection string inside **appsettings.json** and remove auto generated **OnConfiguring()** method from dbcontext class as it is not good practice to have connection string inside OnConfiguring() method.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CompanyDB": "Server=******;Database=CompanyDB;Integrated Securit
  }
}
```

In the **Startup.cs** class add **CompanyDBContext** as a service inside **ConfigureService()** method as below. We will retrieve the connection string value from **appsettings.json** file through **IConfiguration** object's **GetConnectionString()** method.

```
public Startup(IConfiguration configuration)
{
  Configuration = configuration;
}
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add ser
public void ConfigureServices(IServiceCollection services)
{
  var connectionString = Configuration.GetConnectionString("CompanyDI
  services.AddDbContextPool<CompanyDBContext>(option =>
  option.UseSqlServer(connectionString));
  services.AddControllersWithViews();
}
```

In order load Employee Controller's **Index** view on application start, we simply need to change controller name to **Employee** inside **Configure()** method.

```
app.UseEndpoints(endpoints =>
{
  endpoints.MapControllerRoute(
  name: "default",
  pattern: "{controller=Employee}/{action=Index}/{id?}");
});
```

Inject **CompnyDBContext** object in the Employees controller's constructor using **dependency injection**.

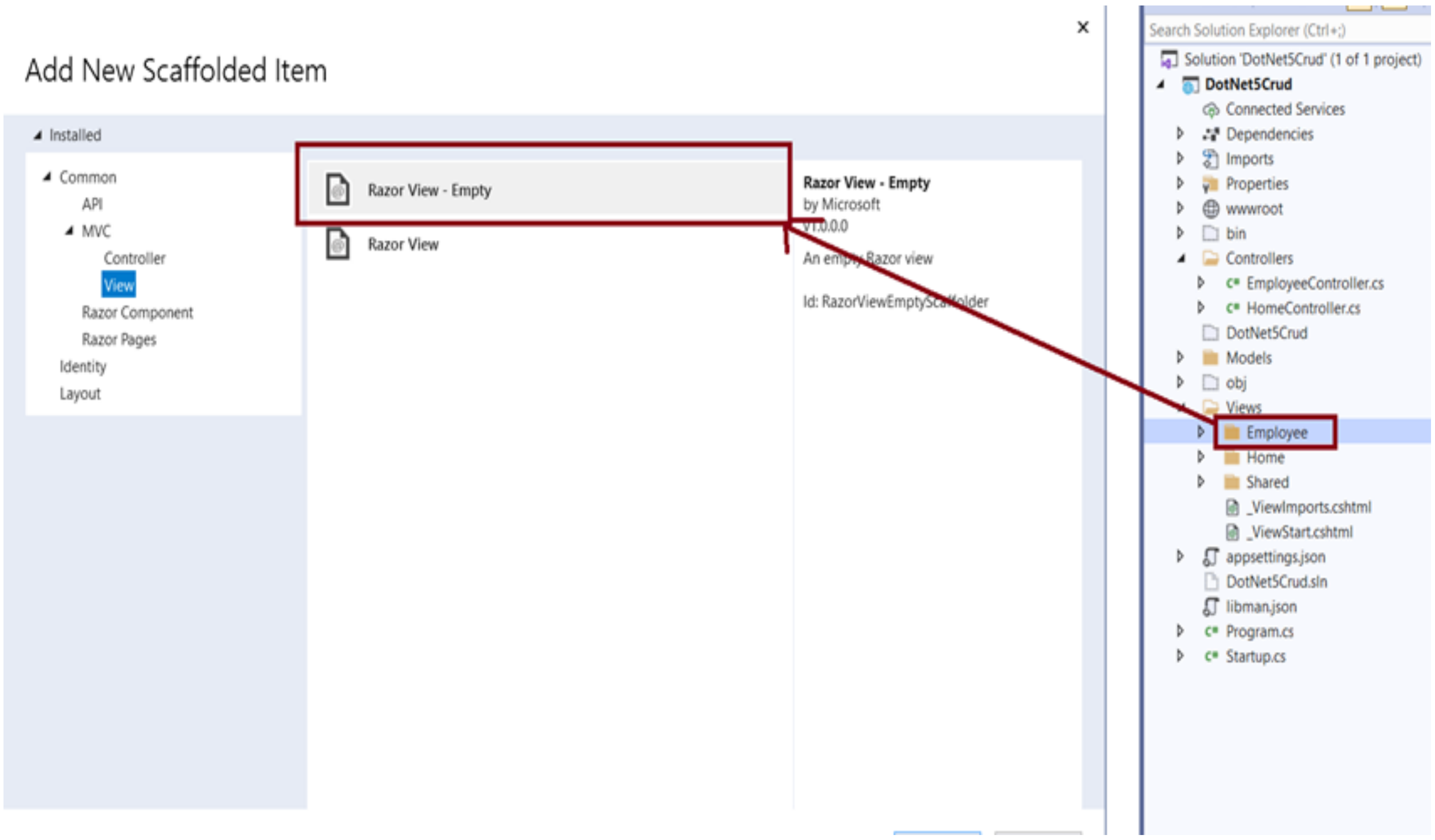
```
public class EmployeeController : Controller
{
  private readonly CompanyDBContext _context;

  public EmployeeController(CompanyDBContext context)
  {
    _context = context;
  }
------
-------
}
```

Step 9 - ASP.NET Core MVC CRUD Operations

General Instructions To Add Views,

Under **Views folder,** Create a new folder named **Employee**. Click on the **Employee** folder and click **Add**.

Then Click on **View** and click on the **Razor View – Empty** template and then click on **Add** to create the view.



After adding all the views for CRUD operation, the Employee folder will have the following views file.

CRUD Operations In ASP.NET Core MVC (.NET 5.0)

Index Action Method

Replace your **Index** action method with the below code inside the employee controller.

```
public async Task<IActionResult> Index()
{
    var employees = await _context.Employees.ToListAsync();
    return View(employees);
}
```

Replace auto generated **Index View** code with below code,

```
@model IEnumerable<DotNet5Crud.Models.Employee>

@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Employee List</h1>

<p style="text-align:right;margin-right:20px;">
    <a class="btn btn-outline-primary" asp-action="AddOrEdit">Create
</p>
<div class="table-responsive">
    <table class="table">
        <thead>
            <tr>
                <th>
                    @Html.DisplayNameFor(model => model.Name)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.Designation)
                </th>
                    <th>
                        @Html.DisplayNameFor(model => model.Address)
                    </th>
                    <th>
                        @Html.DisplayNameFor(model => model.Salary)
                    </th>
                    <th>
                        @Html.DisplayNameFor(model => model.JoiningDate)
                    </th>
                    <th></th>
                    <th>Edit Action</th>
                    <th>Details Action</th>
                    <th>Delete Action</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.Name)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Designation)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Address)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Salary)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.JoiningDate)
                    </td>
                    <td>
                    <td class="text-center">
                        <a asp-action="AddOrEdit" class="btn btn-outline-
                    </td>
                    <td class="text-center">
                        <a asp-action="Details" class="btn btn-outline-ir
                    </td>
```

```
                    <td class="text-center">
                        <a asp-action="Delete" class="btn btn-outline-dar
                    </td>
                </tr>
            }
        </tbody>
    </table>
    </div>
```

Index

It returns all the employees from employee table and presents it to the index view.

AddOrEdit Action Method

Replace **Create** and **Edit** action methods with **AddOrEdit** single action method below,

```
//AddOrEdit Get Method
public async Task<IActionResult> AddOrEdit(int? employeeId)
{
    ViewBag.PageName = employeeId == null ? "Create Employee" : "Edit
    ViewBag.IsEdit = employeeId == null ? false : true;
    if (employeeId == null)
    {
        return View();
    }
    else
    {
        var employee = await _context.Employees.FindAsync(employeeId)

        if (employee == null)
        {
            return NotFound();
        }
        return View(employee);
    }
}

//AddOrEdit Post Method
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> AddOrEdit(int employeeId, [Bind("Emp
Employee employeeData)
```

```
{
    bool IsEmployeeExist = false;

    Employee employee = await _context.Employees.FindAsync(employeeI

    if (employee != null)
    {
        IsEmployeeExist = true;
    }
    else
    {
        employee = new Employee();
    }

    if (ModelState.IsValid)
    {
        try
        {
            employee.Name = employeeData.Name;
            employee.Designation = employeeData.Designation;
            employee.Address = employeeData.Address;
            employee.Salary = employeeData.Salary;
            employee.JoiningDate = employeeData.JoiningDate;

           if(IsEmployeeExist)
            {
                _context.Update(employee);
            }
            else
            {
                _context.Add(employee);
            }
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            throw;
        }
        return RedirectToAction(nameof(Index));
    }
    return View(employeeData);
}
```

Replace below code in **AddOrEdit** view,

```
@model DotNet5Crud.Models.Employee

@{
    ViewData["Title"] = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="container p-3 my-3 border" >
    <h1> @ViewBag.PageName</h1>

    <div class="row">
        <div class="col-sm-6">
            <hr />
            <form asp-action="AddOrEdit">
                <div asp-validation-summary="ModelOnly" class="text-d
                @if (@ViewBag.IsEdit)
                {
                    <input type="hidden" asp-for="EmployeeId" />
                }
                <div class="form-group">
                    <label asp-for="Name" class="control-label"></lab
                    <input asp-for="Name" class="form-control" />
                    <span asp-validation-for="Name" class="text-dange
                </div>
                <div class="form-group">
                    <label asp-for="Designation" class="control-label
                    <input asp-for="Designation" class="form-control"
                    <span asp-validation-for="Designation" class="tex
                </div>
                <div class="form-group">
                    <label asp-for="Address" class="control-label"></
                    <input asp-for="Address" class="form-control" />
                    <span asp-validation-for="Address" class="text-da
                </div>
                <div class="form-group">
                    <label asp-for="Salary" class="control-label"></l
                    <input asp-for="Salary" class="form-control" />
                    <span asp-validation-for="Salary" class="text-dar
                    </div>
                    <div class="form-group">
                        <label asp-for="JoiningDate" class="control-label
                        <input asp-for="JoiningDate" class="form-control'
                        <span asp-validation-for="JoiningDate" class="tex
                    </div>
                    <div class="form-group">
                        <input class="btn btn-primary" type="submit" valu
                        <a class="btn btn-danger" asp-action="Index">Back
                    </div>
                </form>
            </div>
        </div>
    </div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Create

It takes employee details as input and creates a new employee record in the employee table.

Edit

It takes employee details as input and updates the new details in the employee table.

Details Action Method

Replace **Details** action method code with below code snippet.

```
// Employee Details
public async Task<IActionResult> Details(int? employeeId)
{
    if (employeeId == null)
    {
        return NotFound();
    }
    var employee = await _context.Employees.FirstOrDefaultAsync(m =>
    if (employee == null)
    {
        return NotFound();
    }
    return View(employee);
}
```

Replace below code in **Details** view,

```
@model DotNet5Crud.Models.Employee
@{
    ViewData["Title"] = "Details";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="container p-3 my-3 border">
    <h1>Employee Details</h1>
    <hr />
    <dl class="row">
        <dt class="col-sm-3">
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd class="col-sm-9">
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt class="col-sm-3">
            @Html.DisplayNameFor(model => model.Designation)
        </dt>
        <dd class="col-sm-9">
            @Html.DisplayFor(model => model.Designation)
        </dd>
        <dt class="col-sm-3">
            @Html.DisplayNameFor(model => model.Address)
        </dt>
        <dd class="col-sm-9">
            @Html.DisplayFor(model => model.Address)
        </dd>
        <dt class="col-sm-3">
            @Html.DisplayNameFor(model => model.Salary)
        </dt>
        <dd class="col-sm-9">
            @Html.DisplayFor(model => model.Salary)
        </dd>
        <dt class="col-sm-3">
            @Html.DisplayNameFor(model => model.JoiningDate)
        </dt>
        <dd class="col-sm-9">
            @Html.DisplayFor(model => model.JoiningDate)
            </dd>
        </dl>
        <div>
            <a class="btn btn-primary" asp-action="AddOrEdit" asp-route-e
            <a class="btn btn-danger" asp-action="Index">Back</a>
        </div>
</div>
```

Details

It returns the employee details from the employee table by employee ID.

Delete Action Method

Replace **Delete** action method code with **the** below code snippet.

```
// GET: Employees/Delete/1
public async Task<IActionResult> Delete(int? employeeId)
{
    if (employeeId == null)
    {
        return NotFound();
    }
    var employee = await _context.Employees.FirstOrDefaultAsync(m =>

    return View(employee);
}

// POST: Employees/Delete/1
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Delete(int employeeId)
{
    var employee = await _context.Employees.FindAsync(employeeId);
    _context.Employees.Remove(employee);
    await _context.SaveChangesAsync();

    return RedirectToAction(nameof(Index));
}
```

Replace below code in **Delete** view:

```
@model DotNet5Crud.Models.Employee
```

```

@{
    ViewData["Title"] = "Delete";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Deleting Employee</h2>

<h3>Would you like to continue?</h3>
<div class="container p-3 my-3 border">
    <hr />
        <dl class="row">
            <dt class="col-sm-3">
                @Html.DisplayNameFor(model => model.Name)
            </dt>
            <dd class="col-sm-9">
                @Html.DisplayFor(model => model.Name)
            </dd>
            <dt class="col-sm-3">
                @Html.DisplayNameFor(model => model.Designation)
            </dt>
            <dd class="col-sm-9">
                @Html.DisplayFor(model => model.Designation)
            </dd>
            <dt class="col-sm-3">
                @Html.DisplayNameFor(model => model.Address)
            </dt>
            <dd class="col-sm-9">
                @Html.DisplayFor(model => model.Address)
            </dd>
            <dt class="col-sm-3">
                @Html.DisplayNameFor(model => model.Salary)
            </dt>
            <dd class="col-sm-9">
                @Html.DisplayFor(model => model.Salary)
            </dd>
            <dt class="col-sm-3">
                @Html.DisplayNameFor(model => model.JoiningDate)
            </dt>
            <dd class="col-sm-9">
                @Html.DisplayFor(model => model.JoiningDate)
                </dd>
            </dl>

                <form asp-action="Delete">
                    <input type="hidden" asp-for="EmployeeId" />
                    <input class="btn btn-primary" type="submit" value="Y
                    <a class="btn btn-danger" asp-action="Index">No</a>
                </form>

    </div>
```

Delete: it takes the employee ID as input, and after confirm delete popup, deletes the employee from employee table.

Step 10 - Validations In ASP.NET Core

We need to check whether the provided input is correct or not before submitting the data. We perform Client side as well as Server side validation for this purpose.

Model Validation

To perform model validation, we need to use vaidation attributes presents in the System.ComponentModel.DataAnnotations namespace. These attributes decides the validation rules for the model properties.

Below are the few importent built-in model attributes,

- **[Required] -** its validates that the input field should not be null or empty.
- **[Compare] -** it validate that the two model fields match or not.
- **[Range] -** it validate that the provided input is in the specified range or not.
- **[StringLength] -** it validate that the provided string input should not exceed the specified limit.
- **[ValidateNever] -** if any model property is decorated with **VlaidateNever** attribues, it mean this property is excluded from validation.

In our crud operation, we have created EmployeeValidator class with properties that need to perform model validation. In the name property we set Required and MaxLenghth attribute to validate against for condition, means name should not be empty and it should not exceed maxlength specified.

```
public class EmployeeValidator
 {
     [Required]
     [MaxLength(50)]
     [Display(Name = "Employee Name")]
```

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

        [Required]
        [Display(Name = "Employee Salary")]
        public decimal Salary { get; set; }

        [Required]
        [Display(Name = "Joining Date")]
        public decimal JoiningDate { get; set; }
    }

    [ModelMetadataType(typeof(EmployeeValidator))]
    public partial class Employee
    {
    }
```

Server Side Validation

Server side validation performs using **ModelState** property of ControllerBase class. Generally ModelState handle errors that comes from model binding and model validation. At the execution od the controller action, **ModelState.IsValid** checks whether any error present or not. If it is true then it will allow to proceed otherwise return the view.

Client Side Validation

Client side validation will not allow the page to submit until the form is valid. It prevent the unnecessary round trip to server.

The below script references support client side validation:

In Layout.cshtml

```
<script src="~/lib/jquery/dist/jquery.min.js"></script>
```

In _ValidationScriptsPartial.cshtml

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></s
<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unob
```

Step 11 - Run The Application

When we **Run** the application. By default index view will be loaded.

CRUD Operations In ASP.NET Core MVC (.NET 5.0)

Click on **Create New** button to create new employee**.**

CRUD Operations In ASP.NET Core MVC (.NET 5.0)

Click on **Edit button** to Edit an employee**.**

CRUD Operations In ASP.NET Core MVC (.NET 5.0)

Click **on Details button** to see the employee detail.

CRUD Operations In ASP.NET Core MVC (.NET 5.0)

Click on **Delete button** to delete the employee.

CRUD Operations In ASP.NET Core MVC (.NET 5.0)

In order to download and run the application, please follow the below steps,

Step 1

Download the solution files from the attachement of this article.

Step 2

Open the solution file in Visual Studio 2019 or later.

Step 3

In order to restore the required NuGet packages, rebuild the solution.

Step 4

Change the connection string in the **appsettings.json** file with your SQL Server connection string.

Step 5

Run the application.

Summary

In this article, we learned step by step process to create ASP.NET Core 5.0 application and performed CRUD operation using Entity Framework Core 5.0. We learned how to create controlller class and write business logic for crud action methods. We applied both client side as well as server side validation for create and edit views. We used scaffold command to reverse engineer database and create dbcontext and model class.

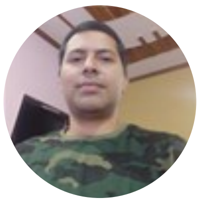
Please share your **valuable feedback** in the **comments section** . Let me know if I missed anything.

In case you have any queries or concerns about this article, please write to me in the comments section below.

Thank You

.NET 5.0 .NET Core .NET Core MVC CRUD ASP.NET Core MVC
CRUD Operations In ASP.NET Core MVC

Next Recommended Reading
CRUD Operations Using ASP.NET Core And ADO.NET



Achyuta Jha

Senior Software Engineer | C# | ASP.NET Core | MVC | Angular | JavaScript | SQL-Server | Entity Framework Core | .NET Core API ...etc.

https://www.youtube.com/channel/UCJI26aghfqPGbF2WU78oWwg

1730 84k

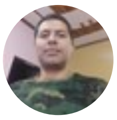
9 3

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

Its working Flawlessly for me .....Thank You...

BHUSHAN SHEJWAL Jan 24, 2022

2088 14 0 1 1 Reply



Thanks Bhushan ....

Achyuta Jha Jan 31, 2022

1730 372 84k 0

It is not good, because it don't have ViewModel

hadi hhmmn Nov 13, 2021

2035 67 1 0 0 Reply

FEATURED ARTICLES

Azure Duration Functions - How To Use And Implement It

Easily Use Flurl Testable HttpClient

Legacy Classes And Legacy Interface Of Collections API

It's Not About How You Inject Your Services, It's About How You Test Them

SPFx Form Customizer Extension To Customize SharePoint New/Edit/Display Form Of List/Libraries

TRENDING UP

01 Azure Durable Functions - An Overview

02 How To Upload Files Into Azure Blog Storage Using Azure Functions In C#

03 Change Data Capture - Another Way To Implement The Incremental Load

04 Azure Duration Functions - How To Use And Implement It

05 Rockin' The Code World with dotNetDave ft. Khalid Abuhakmeh Ep. 52

06 Creating Search Feature In Blazor Server Grid

07 Rockin' The Code World with dotNetDave ft. Steve Jones Ep. 53

08 Growth Mindset Show Ep. 11 - 2022

09 How To Handle Nullable Reference In .NET 6

10 Distributed Transaction in C# Microservices using SAGA Pattern

Microsoft® SQL Server®
Reporting Services

Learn SSRS In 11 Hours

CHALLENGE YOURSELF

HTML

GET CERTIFIED

Python Developer