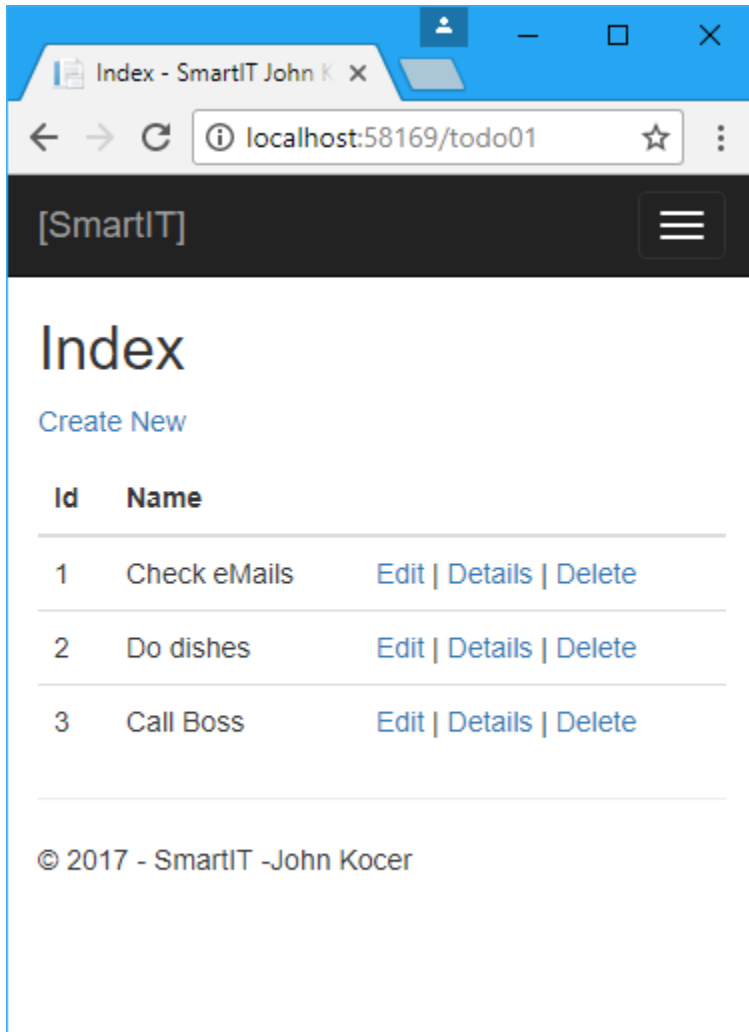# How to write simple Todo CRUD ASP.NET MVC Application.

Visual Studio 2017, ASP.NET Core 2.0

By John kocer- SmartIT



## Download color pdf version of this article with pictures.

https://github.com/SmartITAz/TodoMvcSolution/blob/master/TodoAspMvcCrud.pdf

NOTE: If you need to copy and paste the code download the pdf file locally.

## Introduction

In this session, we will learn

-How to consume Todo inmemory database using TodoRepository.
-How to create custom ASP.NET MVC custom controller with CRUD operations.
-List of Objects
-Create a new insert object via View
-Update and object via Edit View
-Delete an Item via Delete View.
-Write HttpPost Create API method.

-Write Httppost Edit API method.
-Write HttpPost Delete API method.
-SmartIT DebugTraceHelper tool
- A lab exercise for you to demonstrate what have you learned from this training material to create your own Employee CRUD operation using EmployeeRepository included in this training material.

Download source code from GitHub: https://github.com/SmartITAz/TodoMvcSolution



# UI -User Interface

We will have 5 views
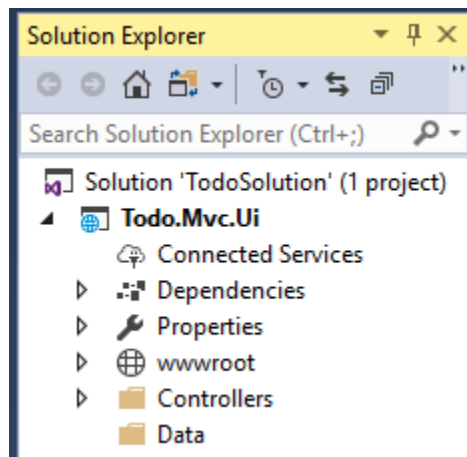
-Index
-Create
-Edit
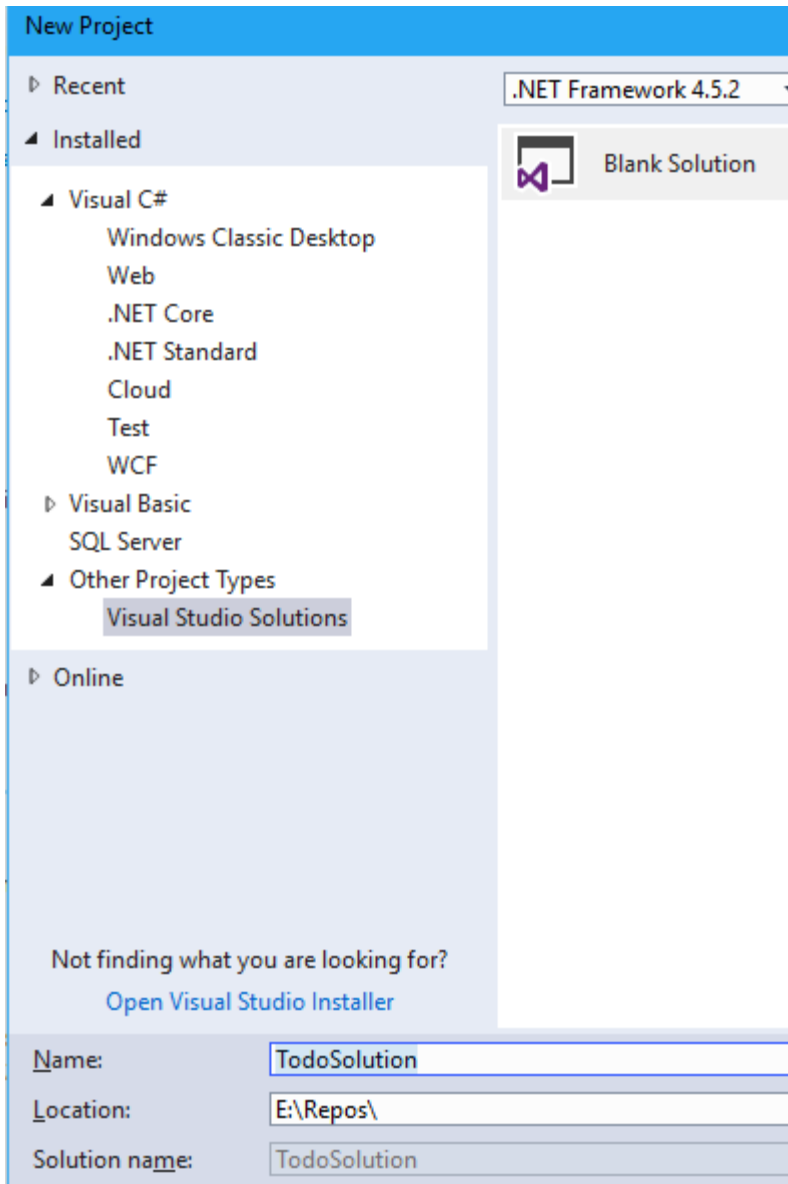-Detail
-Delete

# Controller

TodoController
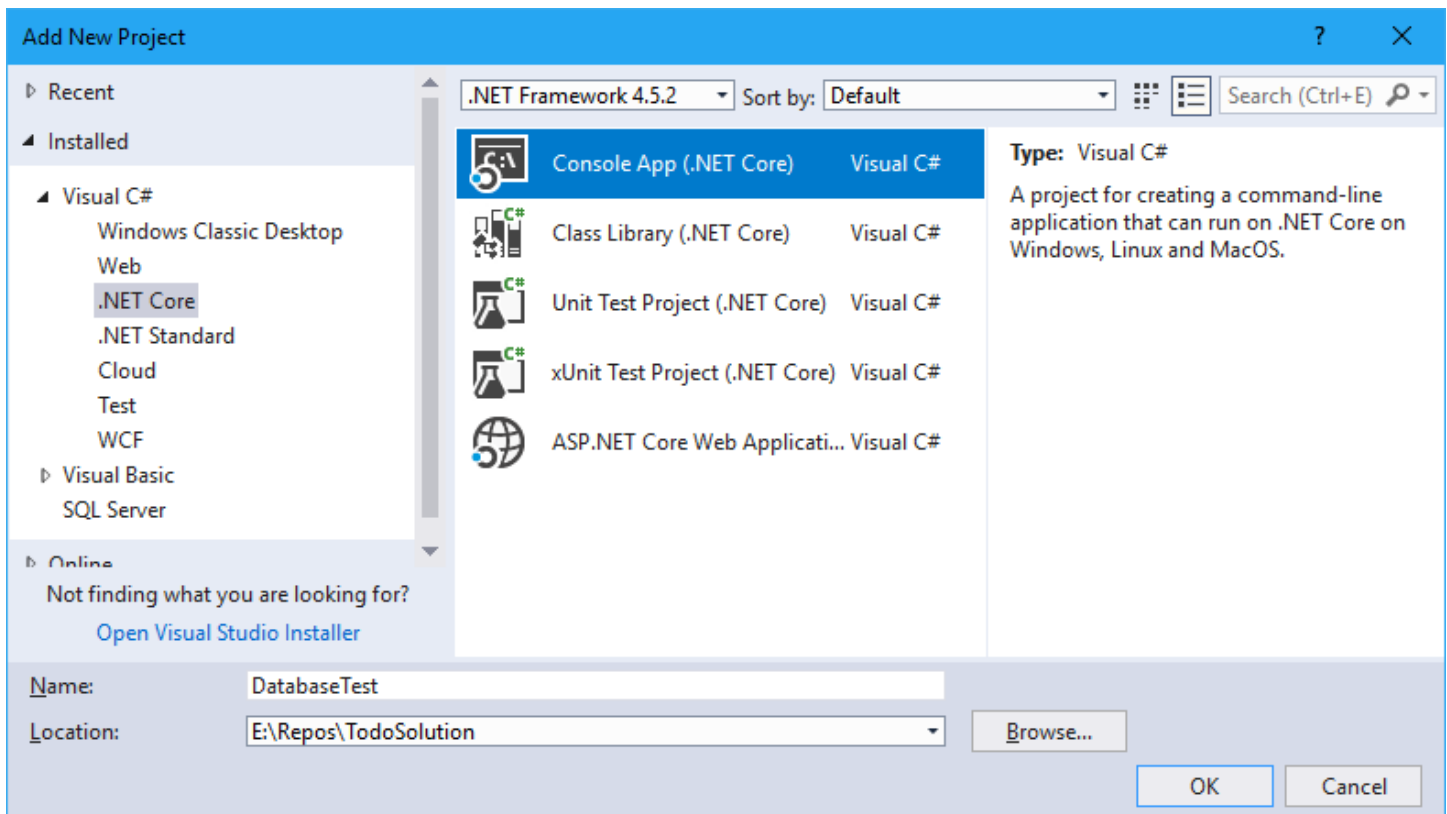Wire the controller to Database and Views

# Database

We will be using in memory Todo repository to simplify database.



-Create a new solution and name it **TodoSolution**

-Add a new .NETCore Console  application called **DatabaseTest**

-Install the **SmartIT.DebugTraceHelper** package latest version from nugget.org and

-Install the `SmartIT.Employee.MockDB 1.0.4` package latest version  from nugget.org

-Let test from MockDB lest get the Toto list

NuGet: DatabaseTest

Browse   Installed   Updates

NuGet Package Manager: DatabaseTest

SmartIT                                    Include prerelease

Package source: nuget.org

**SmartIT.Employee.MockDB** by SmartIT by John Kocer, **80** downlo   v1.0.4
Development Employee Mock Database, Fast API Development/Test
Training

**SmartIT.DebugTaceHelper** by SmartIT -John Kocer   v1.0.1
SmartIT Debug Trace Helper -John Kocer

**DebugTraceHelper** by SmartIT -John Kocer, **35** downloads   v1.0.0
SmartIT Debug Trace Helper -John Kocer

**DebugTaceHelper** by SmartIT -John Kocer, **25** downloads   v1.0.0
SmartIT Debug Trace Helper -John Kocer

**SmartLifeLtd** by SmartLife-Solutions Corporation, **182** downloads   v1.0.1.2
SmartLife Solutions (SLS) specialized in innovative, interactive and smart
solutions that provides clients with a fully integrated and professional r...

**SmartLifeLtdWeb** by SmartLife-Solutions Corporation, **66** downloa   v1.0.0.2
SmartLife Solutions (SLS) specialized in innovative, interactive and smart
solutions that provides clients with a fully integrated and professional r...

**Usa.Smart.Resolver** by machi_pon, **1.73K** downloads   v1.3.0
Smart container library for .NET

**angular-smart-table** by RENARD Laurent, **27.5K** downloads   v2.1.8
angular smart table package

### SmartIT.DebugTaceHelper

Version: Latest stable 1.0.1 ▾   Install

⊙ Options

**Description**
Debug helper logging utility

**Version:**          1.0.1
**Author(s):**       SmartIT -John Kocer
**Date published:**  Friday, September 15, 2017
(9/15/2017)
**Report Abuse:**    https://www.nuget.org/
packages/
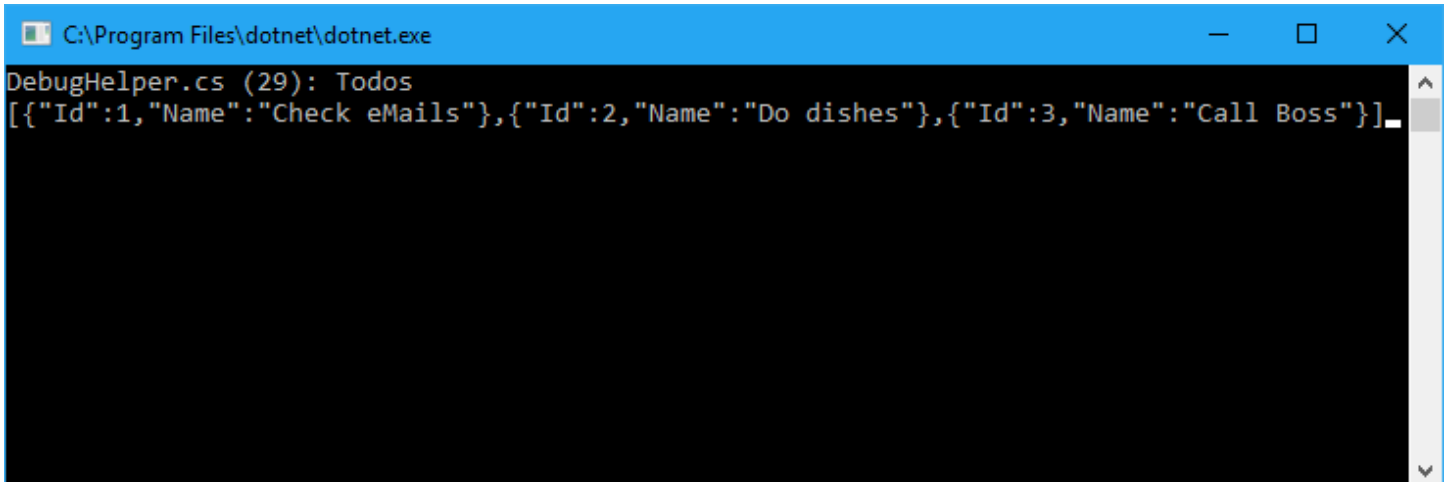SmartIT.DebugTaceHelper/1.0.1/
ReportAbuse

**Tags:**            logger, logs, Debug, trace,
logging

**Dependencies**

.NETCoreApp,Version=v2.0
Newtonsoft.Json (>= 10.0.3)

```csharp
using SmartIT.DebugHelper;
using SmartIT.Employee.MockDB;
using System;

namespace DatabaseTest
{
  class Program
  {
    static void Main(string[] args)
    {
      TodoRepository _todoRepository = new TodoRepository();
      var todoList=_todoRepository.GetAll();
      todoList.CDump("Todos");
      Console.ReadLine();
    }
  }
}
```

You will see SmartIT.DebugHelper object extension CDump() method wrote Todo list as Jason formatted to the console below.
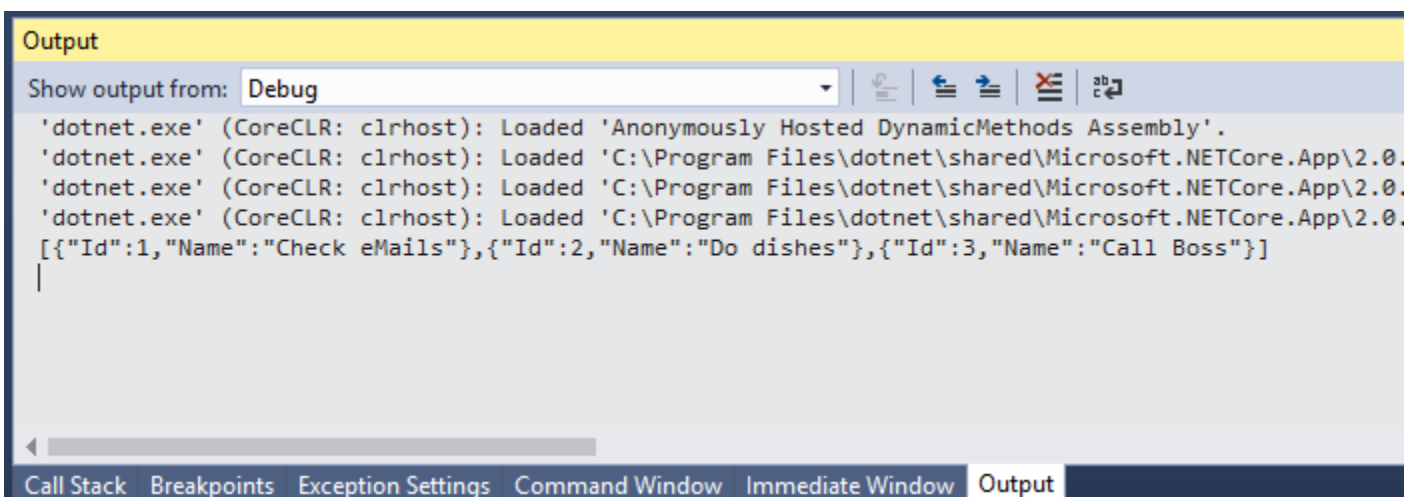


```
C:\Program Files\dotnet\dotnet.exe                                    —  □  ✕
DebugHelper.cs (29): Todos
[{"Id":1,"Name":"Check eMails"},{"Id":2,"Name":"Do dishes"},{"Id":3,"Name":"Call Boss"}]_
```

Note: SmartIT.DebugHelper also has DDump() object extension method write Output debug window.

```csharp
namespace SmartIT.DebugHelper
{
  public static class DebugHelper
  {
    public static void CDump(this object obj, string message);
    public static void DDump(this object obj, string message);
  }
}

class Program
  {
    static void Main(string[] args)
    {
      TodoRepository _todoRepository = new TodoRepository();
      var todoList=_todoRepository.GetAll();
      todoList.DDump("Todos");

      Console.ReadLine();
    }
  }
```



```
Output
Show output from: Debug                                    ▾ | ⚹ | ⬆ ⬇ | ⤮ | ᵃᵇ⤶
 'dotnet.exe' (CoreCLR: clrhost): Loaded 'Anonymously Hosted DynamicMethods Assembly'.
 'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.
 'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.
 'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\2.0.
 [{"Id":1,"Name":"Check eMails"},{"Id":2,"Name":"Do dishes"},{"Id":3,"Name":"Call Boss"}]
 |

Call Stack  Breakpoints  Exception Settings  Command Window  Immediate Window  Output
```

SmartIT.DebugHelper DDump() extension method is useful when you are using other project type such as below project types, because other projects may not have console output functionally.
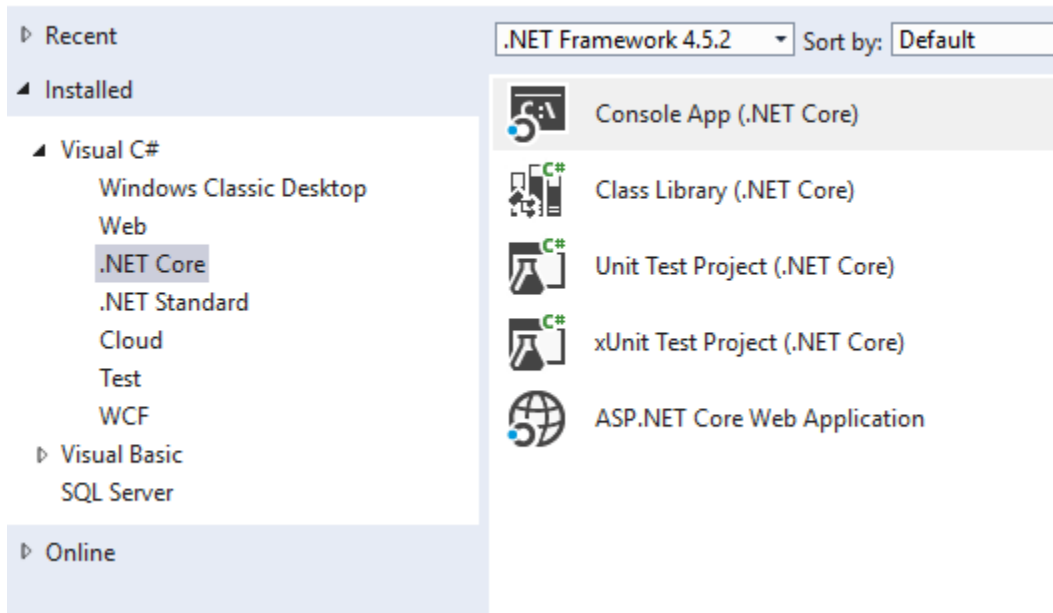


-A Database CRUD operation we need to use below TodoRepository functionalities.

```
_todoRepository.GetAll();
    _todoRepository.FindById()
    _todoRepository.Add()
    _todoRepository.Add(new SmartIT.Employee.MockDB.Todo() { Name = collection["Name"] });
    _todoRepository.Update(newTodo);
    _todoRepository.Delete(deleteTodo)
```
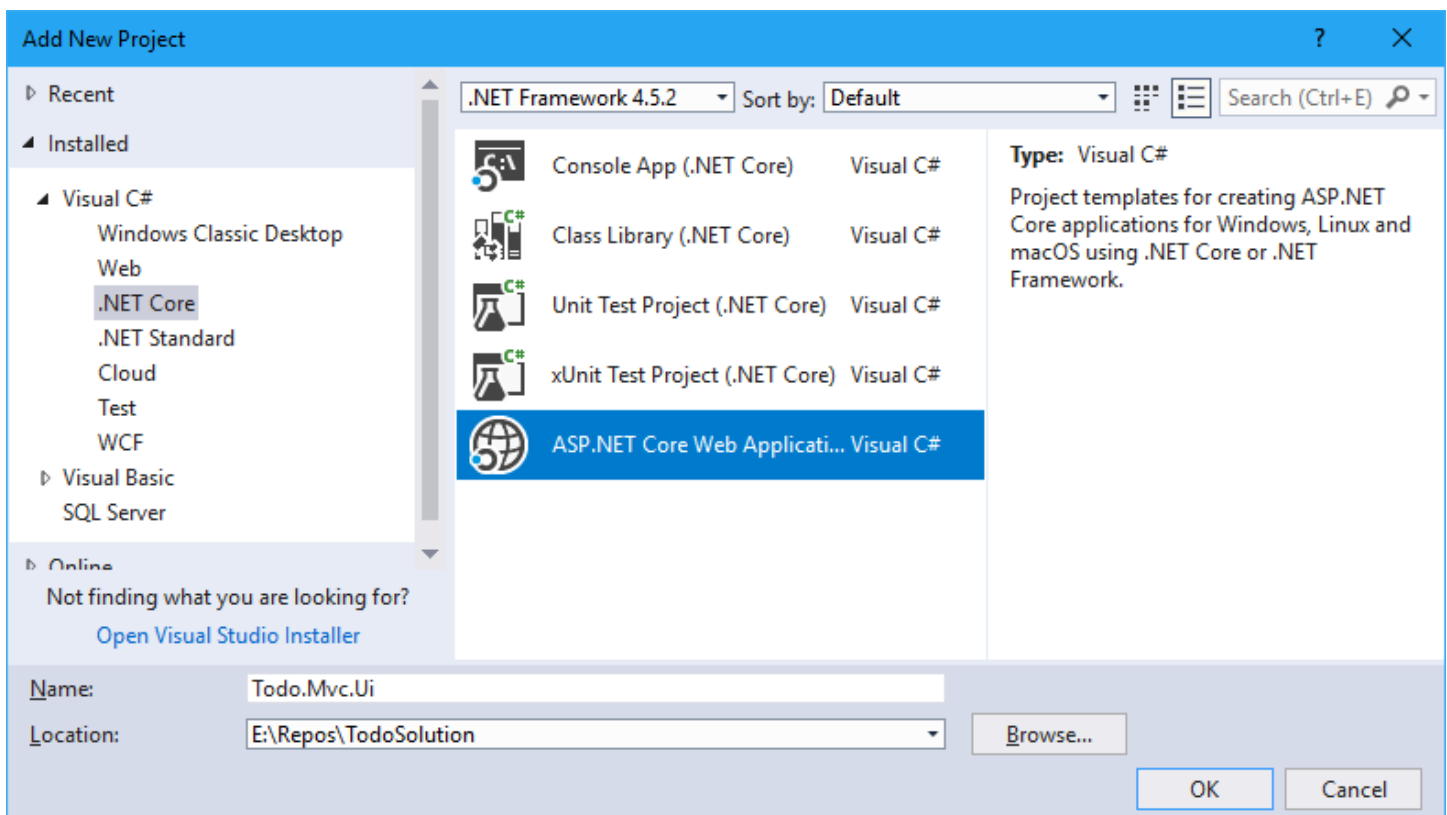
-Lets test this functionalities before we use it.
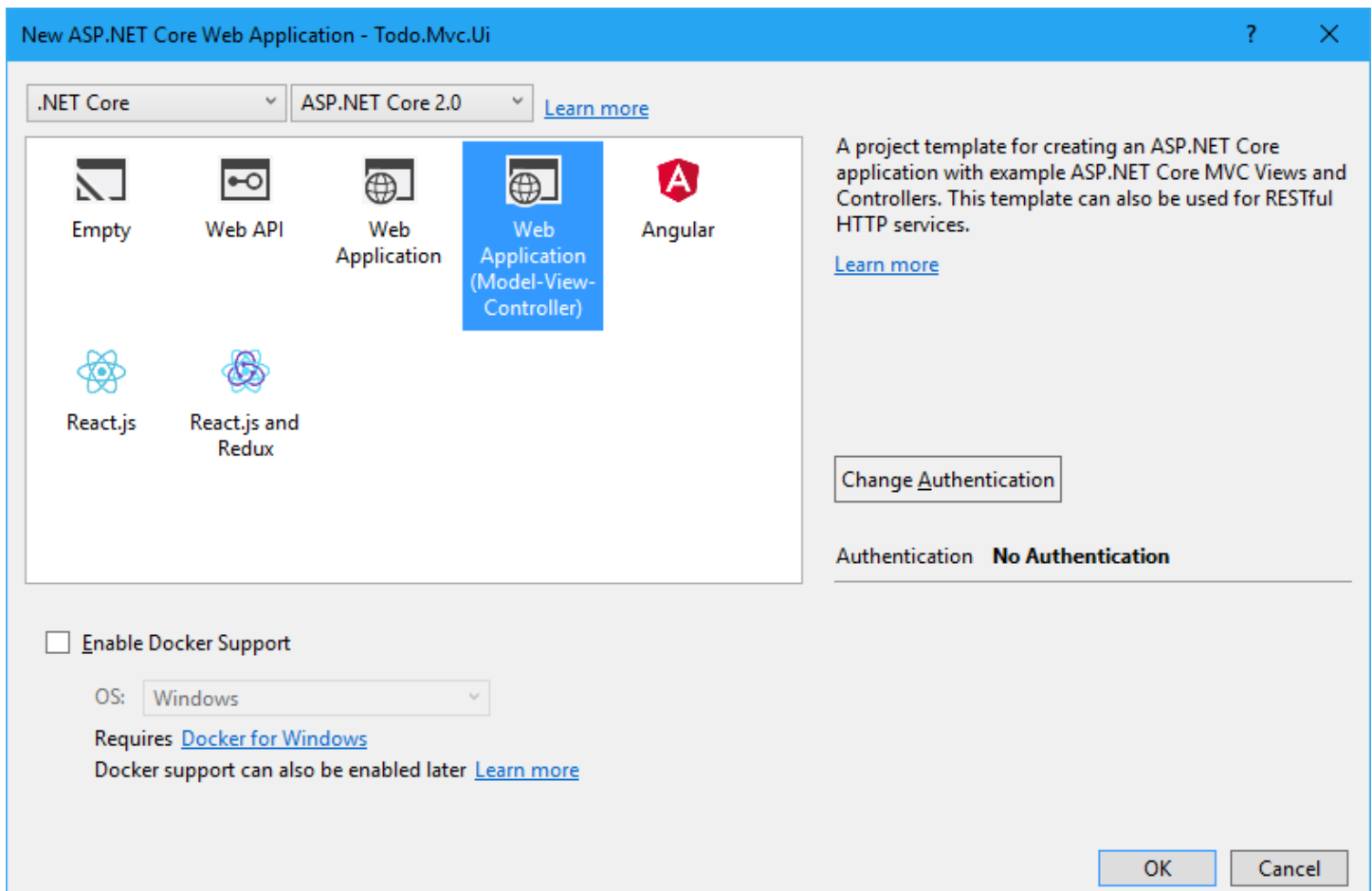
```
static void Main(string[] args)
    {
        TodoRepository _todoRepository = new TodoRepository();
        var todoList = _todoRepository.GetAll();
        todoList.CDump("_todoRepository.GetAll()");
        var findById = _todoRepository.FindById(2);
        findById.CDump("_todoRepository.FindById(2)");
        var newTodo = _todoRepository.Add(new Todo { Name = "Call a friend" });
        _todoRepository.GetAll().CDump("Check if Call a friend todo added?");
        newTodo.Name = newTodo.Name + " Updated";
        _todoRepository.Update(newTodo);
        _todoRepository.GetAll().CDump("Check if Call a friend todo updated with Updated?");
        _todoRepository.Delete(_todoRepository.FindById(1));
        _todoRepository.GetAll().CDump("Check if Id=1 todo is Deleted?");
```

We can see from below output that all Todo functionality was passed for GetAll, FindById, Update and Delete.

```
C:\Program Files\dotnet\dotnet.exe                          —   □   X

DebugHelper.cs (29): _todoRepository.GetAll()
[{"Id":1,"Name":"Check eMails"},{"Id":2,"Name":"Do dishes"},{"Id":3,"Name":"Call Boss"}]
DebugHelper.cs (29): _todoRepository.FindById(2)
{"Id":2,"Name":"Do dishes"}
DebugHelper.cs (29): Check if Call a friend todo added?
[{"Id":1,"Name":"Check eMails"},{"Id":2,"Name":"Do dishes"},{"Id":3,"Name":"Call Boss"},{"Id":4,"Name":"Call a friend"}]

DebugHelper.cs (29): Check if Call a friend todo updated with Updated?
[{"Id":1,"Name":"Check eMails"},{"Id":2,"Name":"Do dishes"},{"Id":3,"Name":"Call Boss"},{"Id":4,"Name":"Call a friend Up
dated"}]
DebugHelper.cs (29): Check if Id=1 todo is Deleted?
[{"Id":2,"Name":"Do dishes"},{"Id":3,"Name":"Call Boss"},{"Id":4,"Name":"Call a friend Updated"}]
```

Part 2: Add a new **Todo.Mvc.Ui** ASP.NET Core Web Application project

New ASP.NET Core Web Application - Todo.Mvc.Ui

.NET Core | ASP.NET Core 2.0 | Learn more

Empty    Web API    Web Application    Web Application (Model-View-Controller)    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

Change Authentication

Authentication   No Authentication

☐ Enable Docker Support

OS:   Windows

Requires Docker for Windows
Docker support can also be enabled later Learn more

OK    Cancel

-Add nugget packages to newly created **Todo.Mvc.Ui** project.

-Install the **SmartIT.DebugTraceHelper** package latest version from nugget.org and

-Install the `SmartIT.Employee.MockDB 1.0.4` package latest version  from nugget.org

## TodoSolution - NuGet - Solution

**NuGet - Solution**

Browse | **Installed** | Updates | Consolidate

Search (Ctrl+E)

☐ Include prerelease

**Manage Packages for Solution**

Package source: nuget.org

---

.NET ✓ **Microsoft.AspNetCore.All** by Microsoft    v2.0.0
Microsoft.AspNetCore.All

.NET ✓ **Microsoft.NETCore.App** by Microsoft    v2.0.0
A set of .NET API's that are included in the default .NET Core application model.

✓ **SmartIT.DebugTaceHelper** by SmartIT -John Kocer    v1.0.1
Debug helper logging utility

✓ **SmartIT.Employee.MockDB** by SmartIT by John Kocer    v1.0.4
SmartIT.Employee.MockDB
Development Employee Mock Database, Fast API Development/Test Training

---

**SmartIT.Employee.MockDB**

Version(s) - 1

| Project |
|---|
| ☐ DatabaseTest |
| ✓ Todo.Mvc.Ui |

**Installed:** 1.0.4    [Uninstall]

**Version:** Latest stable 1.0.4    [Install]

⌄ Options

**Description**

SmartIT.Employee.MockDB
Development Employee Mock Database, Fast API Development/Test Training

---

Solution 'TodoSolution' (2 projects)
▷ C# DatabaseTest
▲ Todo.Mvc.Ui
    ☁ Connected Services
  ▲ ⋅⋮⋅ Dependencies
   ▷ Analyzers
   ▲ NuGet
     ▷ Microsoft.AspNetCore.All (2.0.0)
     ▷ SmartIT.DebugTaceHelper (1.0.1)
     ▷ SmartIT.Employee.MockDB (1.0.4)
   ▷ SDK
  ▷ Properties
  ▷ ⊕ wwwroot
  ▷ Controllers
  ▷ Models
  ▷ Views
  ▷ appsettings.json
  ▷ bower.json
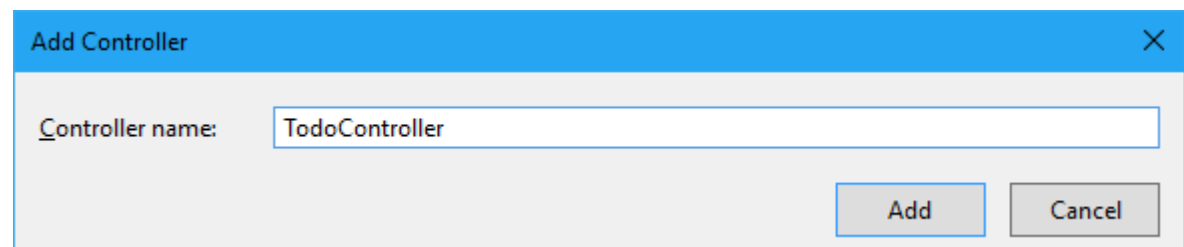   bundleconfig.json
  ▷ C# Program.cs
  ▷ C# Startup.cs

-Rebuild the **Todo.Mvc.Ui** project.

-Right mouse click on the controllers directory and select add controller
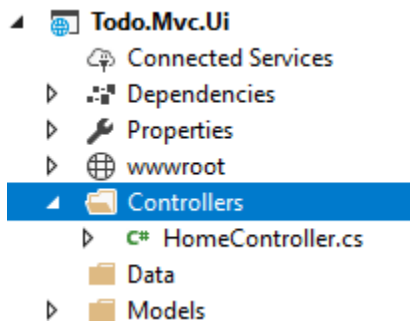


-Select MVC Controller with read/write actions like below.
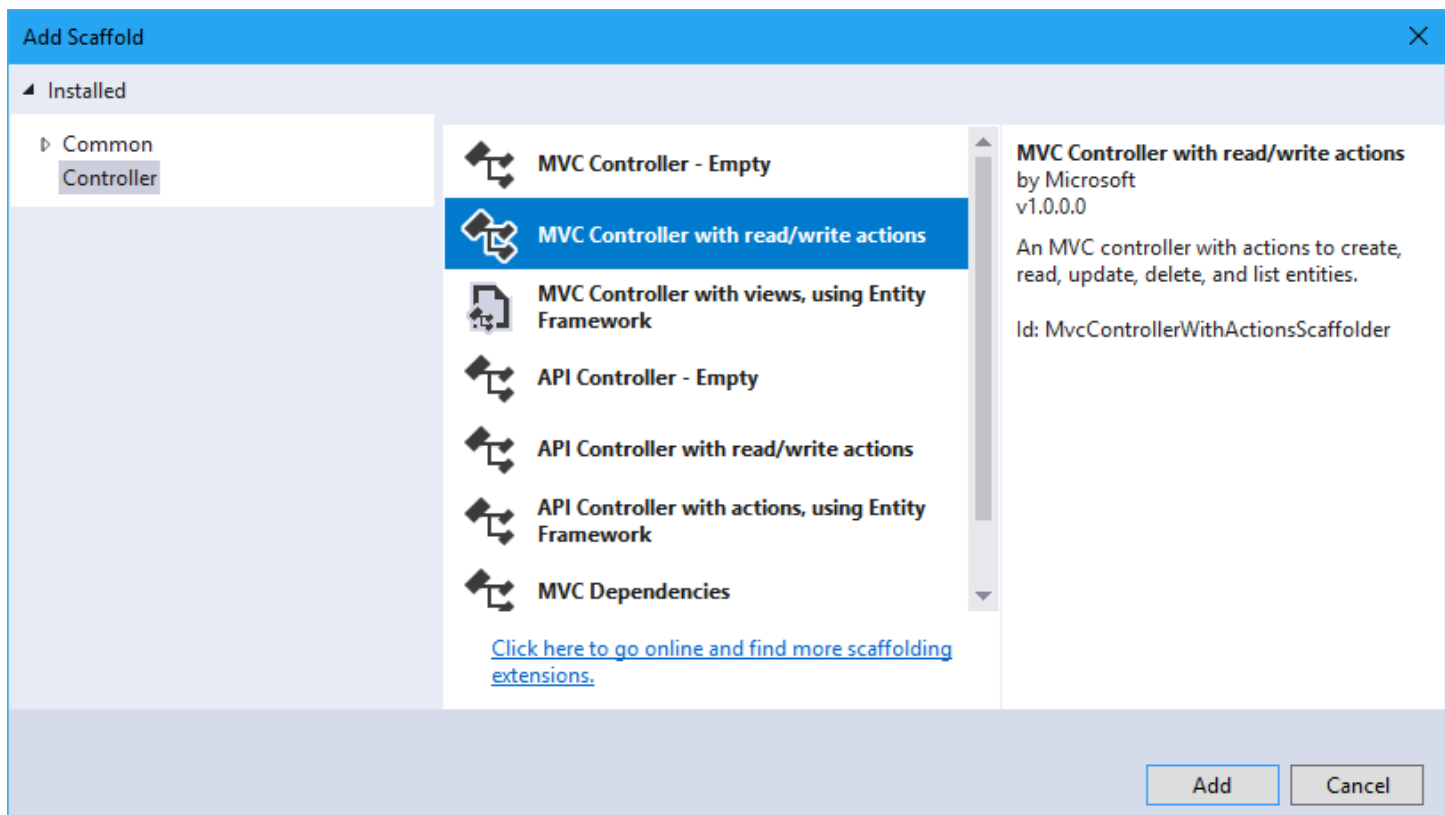


-Name the controller as  TodoController



-Add the namespaces to the TodoController file

```
using SmartIT.DebugHelper;
using SmartIT.Employee.MockDB;
```

- Add todo repository top of the `TodoController class as private member.`

`TodoRepository _todoRepository = new TodoRepository();`
-Update the `public ActionResult Index()` method return value with all todo list calling GetAll() method
 `return View(_todoRepository.GetAll());`
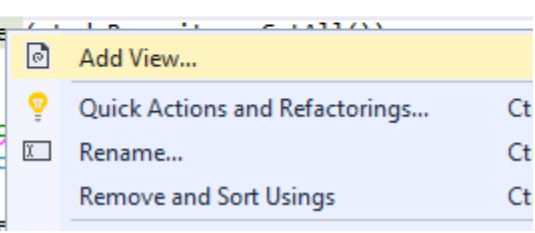
-Here you can see the changes in below code section.

```
public class TodoController : Controller
  {
     TodoRepository _todoRepository = new TodoRepository();
     // GET: Todo
     public ActionResult Index()
     {

        return View(_todoRepository.GetAll());
     }
```

-Right mouse click on the View in Index() method and select Add View like below screen capture.



-Choose default View name as Index

Our Todo class is not in **Model class** dropdown list.  So we need a work around.

**Work around:**

-Inside Models directory add a new Workaround class.
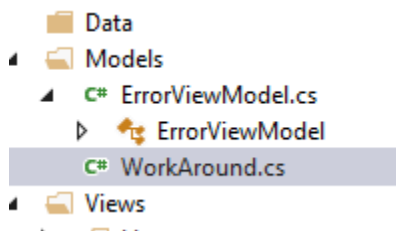


-And temporary write a new Todo class that inherit from `SmartIT.Employee.MockDB.Todo class.`

```
namespace Todo.Mvc.Ui.Models
{
    public class Todo : SmartIT.Employee.MockDB.Todo{}
}
```

-Now add a view to Index and chose Todo model class

## Add View

| | |
|---|---|
| View name: | Index |
| Template: | List ▾ |
| Model class: | Todo (Todo.Mvc.Ui.Models) ▾ |

Options:

☐ Create as a partial view

☑ Reference script libraries

☑ Use a layout page:

[                                                    ] [ ... ]

(Leave empty if it is set in a Razor _viewstart file)

[ Add ]  [ Cancel ]

-Index.html page change model namespace to SmartIT.Employee.MockDB.

-Lest change Route tempale from  template: "{controller=Home}/{action=Index}/{id?}"); to todo

```
app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Todo}/{action=Index}/{id?}");
        });
```

-And run the project
-From Index page click the Create New Link
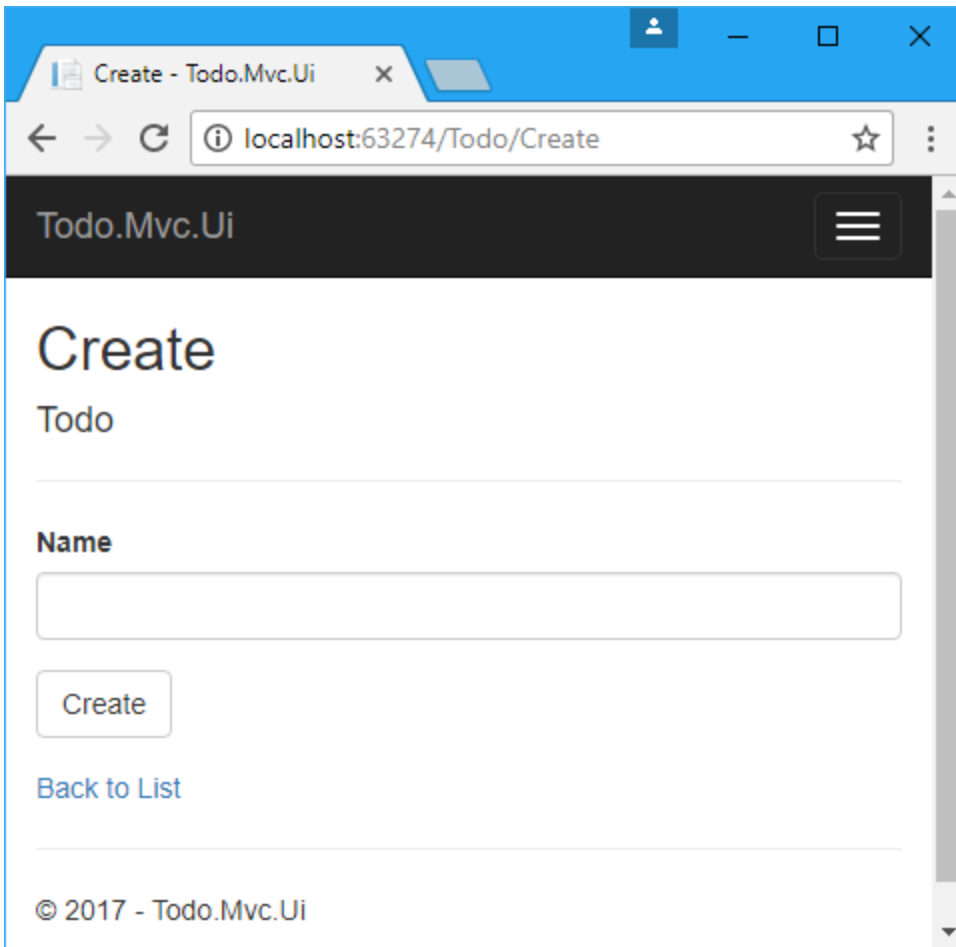
-Update the HttpPost Create method like below

```
[HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(IFormCollection collection)
    {
      try
      {
        _todoRepository.Add(new SmartIT.Employee.MockDB.Todo() { Name=collection["Name"]});

        return RedirectToAction(nameof(Index));
      }
      catch
      {
        return View();
      }
    }
```

Add a new todo
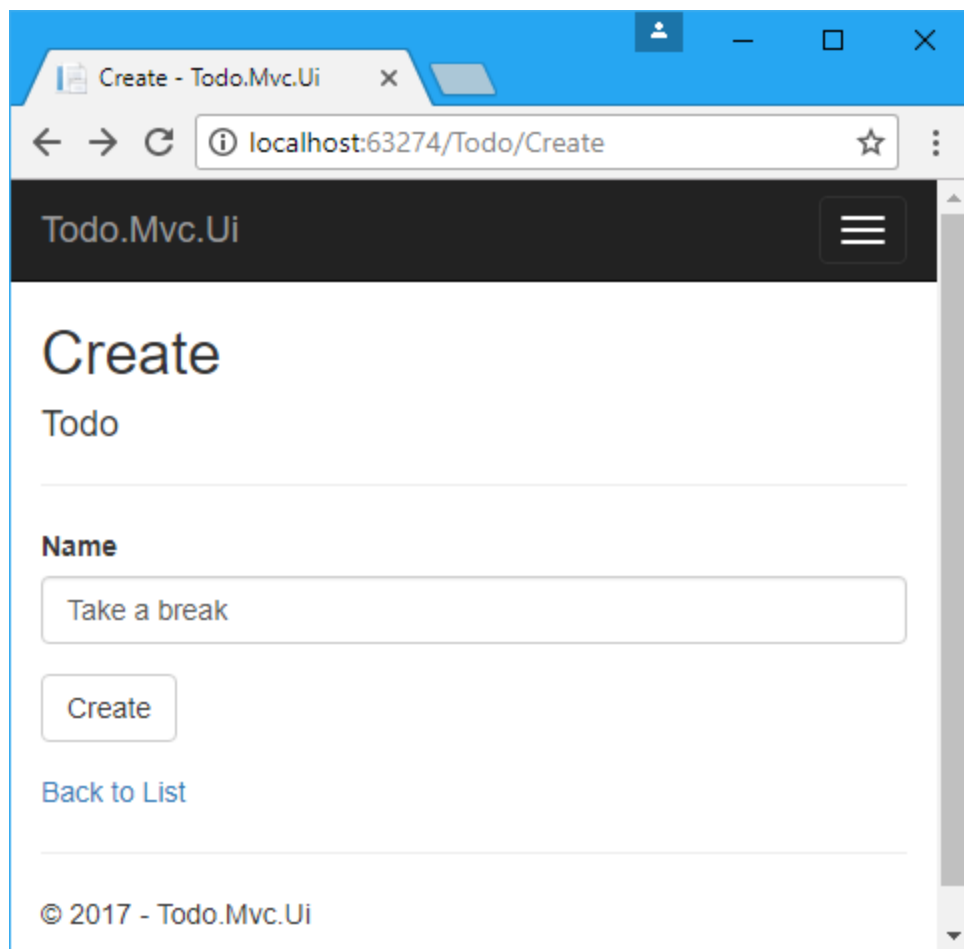
# Index - Todo.Mvc.Ui

localhost:63274

## Todo.Mvc.Ui

# Index

Create New

| Id | Name |  |
|----|------|--|
| 1 | Check eMails | Edit \| Details \| Delete |
| 2 | Do dishes | Edit \| Details \| Delete |
| 3 | Call Boss | Edit \| Details \| Delete |
| 4 | Take a break | Edit \| Details \| Delete |

© 2017 - Todo.Mvc.Ui

```razor
@model IEnumerable<SmartIT.Employee.MockDB.Todo>
@{
  ViewData["Title"] = "Index";
}
<h2>Index</h2>
<p>
  <a asp-action="Create">Create New</a>
</p>
<table class="table">
  <thead>
    <tr>
      <th>
        @Html.DisplayNameFor(model => model.Id)
      </th>
      <th>
        @Html.DisplayNameFor(model => model.Name)
      </th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in Model)
    {
      <tr>
        <td>
          @Html.DisplayFor(modelItem => item.Id)
        </td>
        <td>
          @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
          @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
          @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
          @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
        </td>
      </tr>
    }
  </tbody>
</table>
```

-Press F5 and run the project and go to todo like below.

**http://localhost:63274/todo**
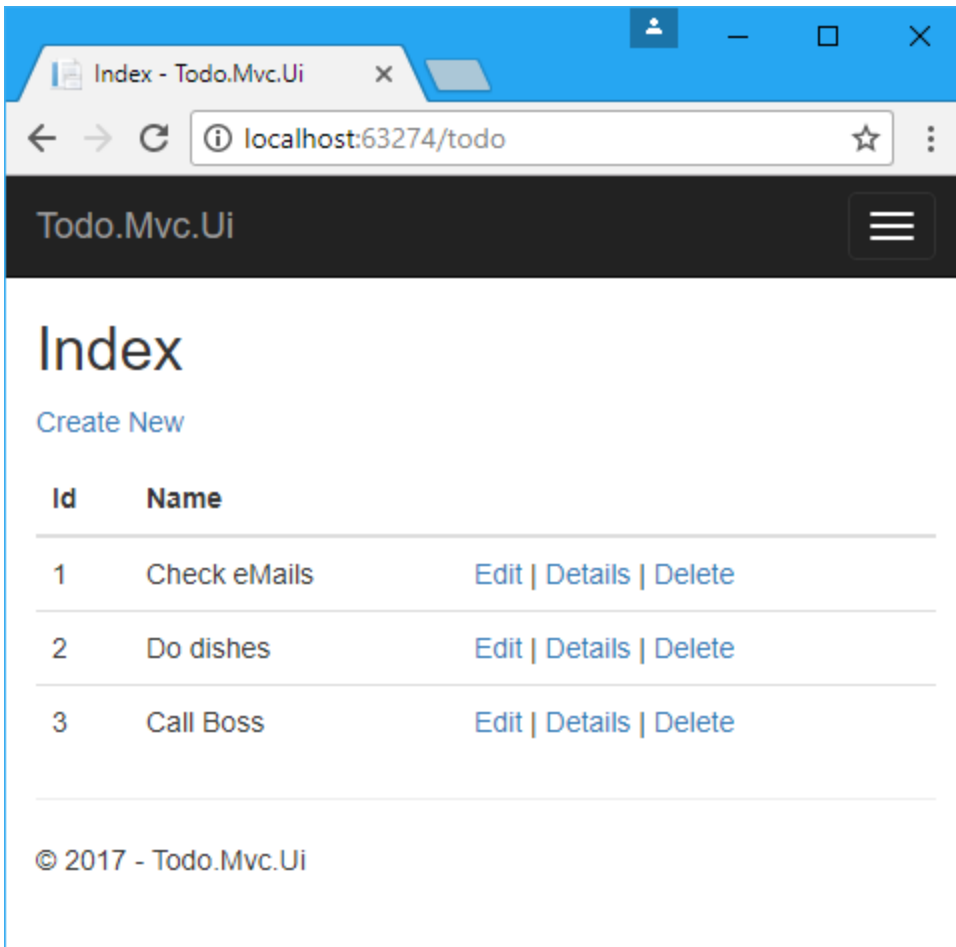
-Add a Create page

Go to `TodoController Create() Method` and write mouse click in the method and select add View

```
// GET: Todo/Create
public ActionResult Create()
{
    return View(
}

// POST: Todo/
[HttpPost]
[ValidateAntiF
public ActionF
{
    try
    {
        // TODO: A

        return Red
```

| | Add View... | |
|---|---|---|
| | Quick Actions and Refactorings... | Ctrl+. |
| | Rename... | Ctrl+R, Ctrl+R |
| | Remove and Sort Usings | Ctrl+R, Ctrl+G |
| | Peek Definition | Alt+F12 |
| | Go To Definition | F12 |
| | Go To Implementation | Ctrl+F12 |
| | Find All References | |
| | View Call Hierarchy | Ctrl+K, Ctrl+T |
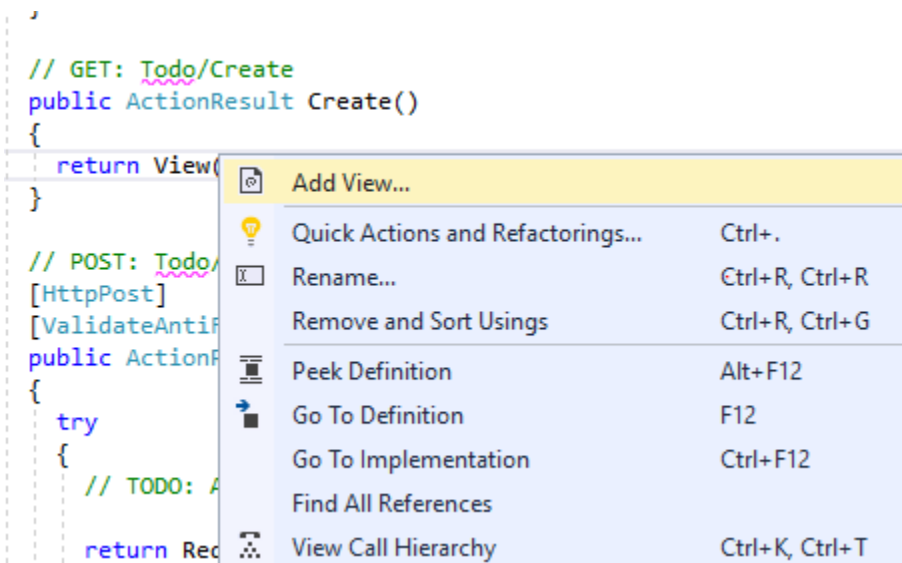
-Select View name and view model like below picture and click to the Add button.

## Add View

| View name: | Create |
|---|---|
| Template: | Create ⌄ |
| Model class: | Todo (Todo.Mvc.Ui.Models) ⌄ |

Options:

☐ Create as a partial view

☑ Reference script libraries

☑ Use a layout page:

|  | ... |
|---|---|

(Leave empty if it is set in a Razor _viewstart file)

Add    Cancel

Inside Create.html change to first line from `Todo.Mvc.Ui.Models.Todo` to `SmartIT.Employee.MockDB.Todo`

-Delete the the id section below

```
<div class="form-group">
            <label asp-for="Id" class="control-label"></label>
            <input asp-for="Id" class="form-control" />
            <span asp-validation-for="Id" class="text-danger"></span>
 </div>
```

Because the Id is automatically added by the TodoRepository.

```razor
@model SmartIT.Employee.MockDB.Todo

@{
    ViewData["Title"] = "Create";
}

<h2>Create</h2>

<h4>Todo</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
              <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

-Add an Edit View page

-Go to the Index.cshtml page

```razor
<td>
        @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
        @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
        @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
  </td>
```

-update the Edit, Details and Delete links to

```razor
@Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
@Html.ActionLink("Details", "Details", new { id=item.Id }) |
@Html.ActionLink("Delete", "Delete", new { id=item.Id })
```
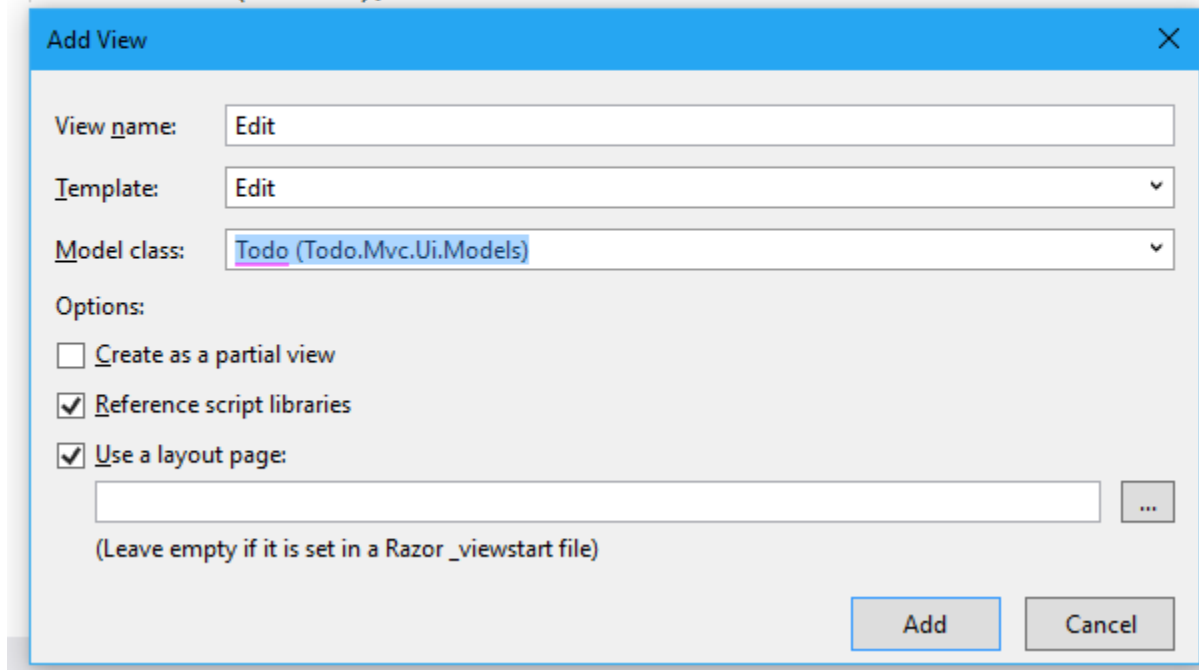
-Go to the TodoController page and update `public ActionResult Edit(int id)` method and update like below and pass the findTodo to the View.

```
// GET: Todo/Edit/5
    public ActionResult Edit(int id)
    {
       var findTodo = _todoRepository.FindById(id);
       return View(findTodo);
    }
```

-Add a new Edit view to the Edit method like below, **Template is Edit and Model class is Todo.**

**-**Click **Add** button**.**



-This will create the **Edit.cshtml** page like below

```
@model Todo.Mvc.Ui.Models.Todo
@{
    ViewData["Title"] = "Edit";
}
<h2>Edit</h2>
<h4>Todo</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Id" class="control-label"></label>
                <input asp-for="Id" class="form-control" />
                <span asp-validation-for="Id" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

-Delete the Id section, because we don't want to update the primary key Id.

```
<div class="form-group">
        <label asp-for="Id" class="control-label"></label>
        <input asp-for="Id" class="form-control" />
        <span asp-validation-for="Id" class="text-danger"></span>
 </div>
```

-Update Todo reference from @model Todo.Mvc.Ui.Models.Todo to SmartIT.Employee.MockDB.Todo

-Below is the to final updated version of Edit View

```
@model SmartIT.Employee.MockDB.Todo
@{
    ViewData["Title"] = "Edit";
}
<h2>Edit</h2>
<h4>Todo</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </form>
    </div>
</div>
<div>
    <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```
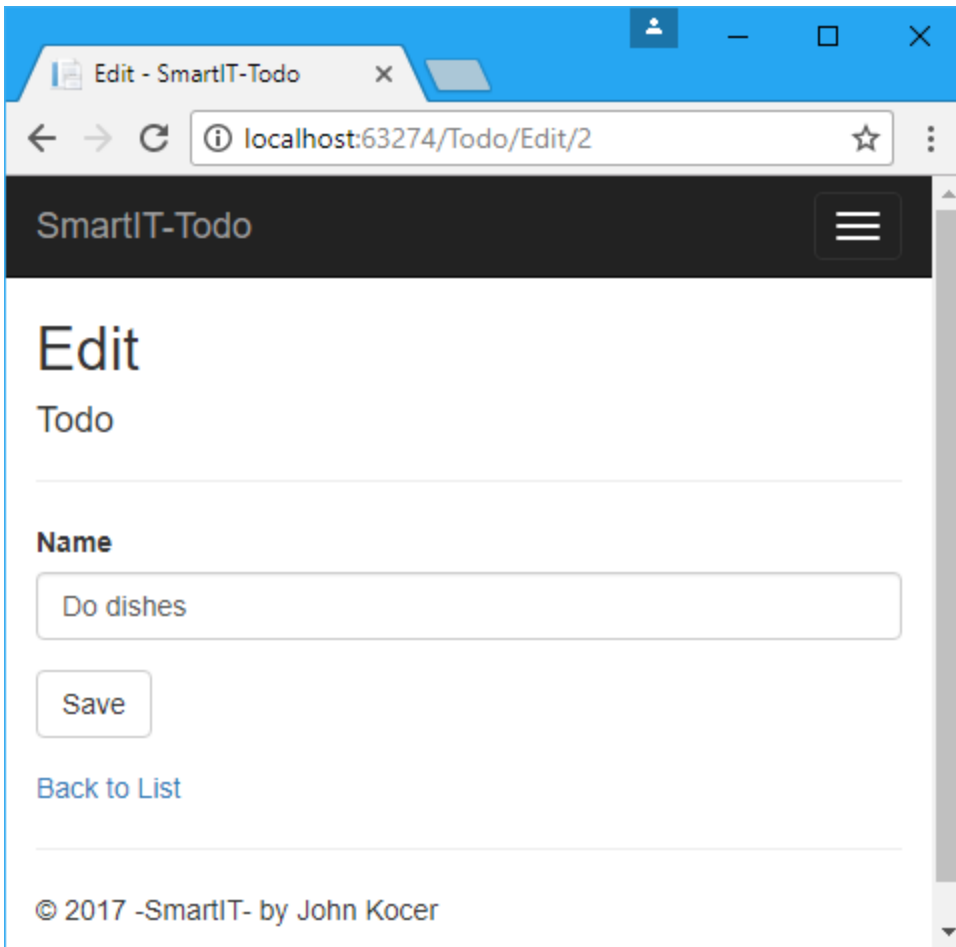
-Press F5 and run the project. From the Index page click Edit link.

-Lets wire the Edit Save button.

-Go to back to TodoController HttpPost Edit method below and update the `// TODO: Add update logic here` section.

```
[HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit(int id, IFormCollection collection)
    {
      try
      {
        // TODO: Add update logic here

        return RedirectToAction(nameof(Index));
      }
      catch
      {
        return View();
      }
    }
```

-Here is the updated edit section below. Using id we find the todo item and using form collection with name key
`findTodo.Name = collection["Name"];`
We can update the Name value.

```
[HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit(int id, IFormCollection collection)
    {
      try
      {
        var findTodo = _todoRepository.FindById(id);
        findTodo.Name = collection["Name"];

        return RedirectToAction(nameof(Index));
      }
      catch
      {
        return View();
      }
    }
```

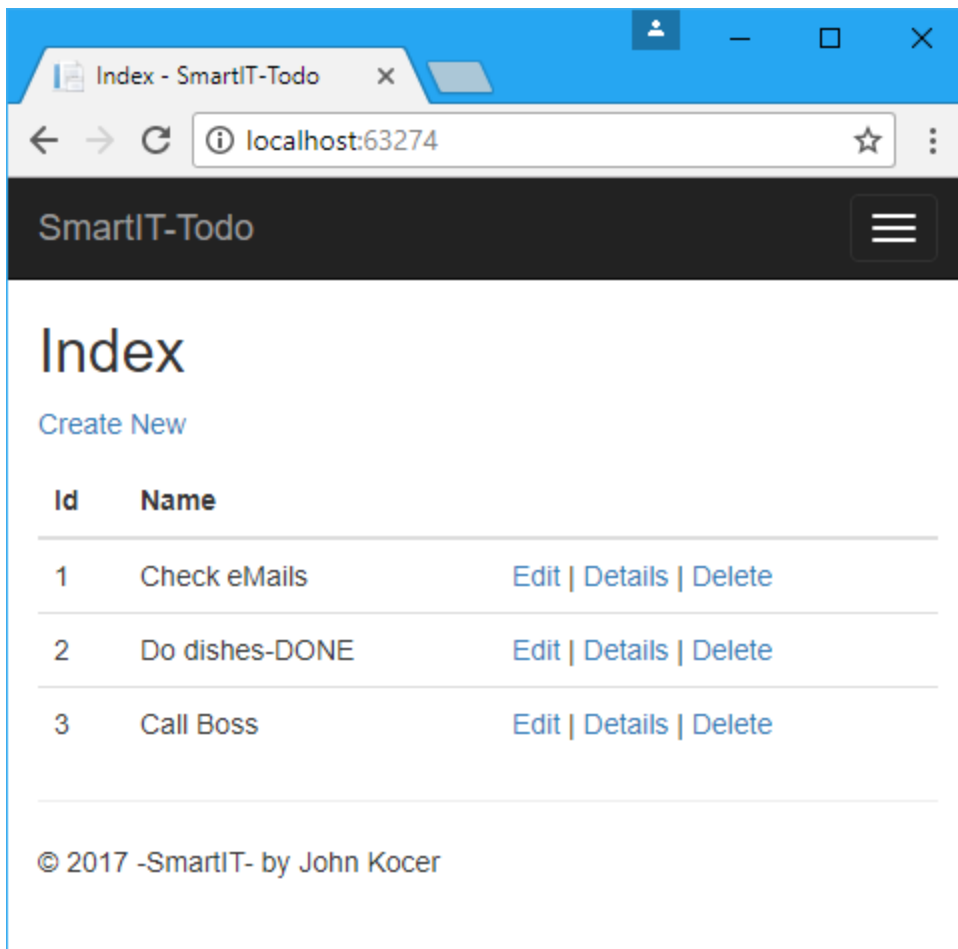-Press F5 and run the project. From the Index page click Edit link.

-Click on the Id=2 Do Dishes Edit link.
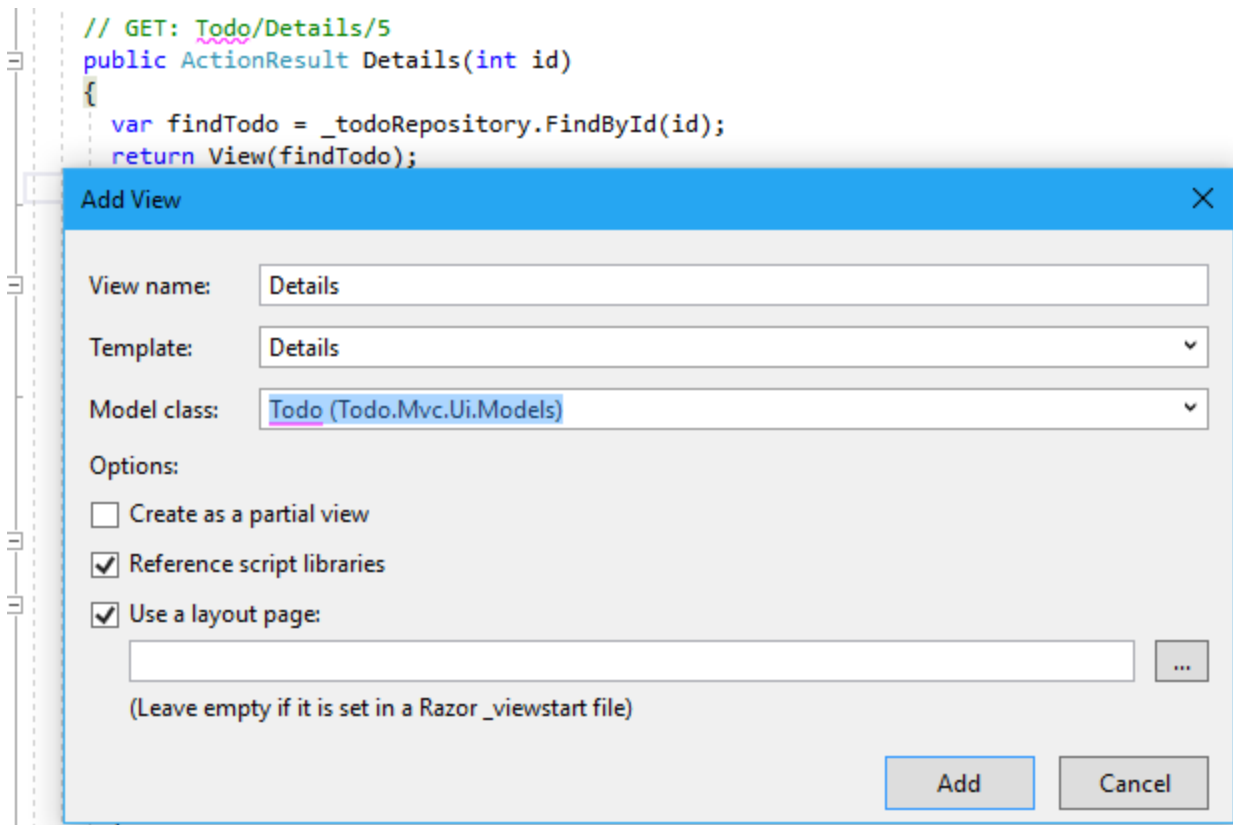
-Update Name to **Do dishes-DONE** .



-Click Save button and we can see the value is updated on the Index view.

-Lets Create the Details View.

-Go to TodoController Details method below.

-Add a View name as Details, select template as Details and Model class to Todo like below screen capture.

```
// GET: Todo/Details/5
public ActionResult Details(int id)
{
    var findTodo = _todoRepository.FindById(id);
    return View(findTodo);
```

**Add View**                                                    ✕

View name:     Details

Template:      Details                                          ⌄

Model class:   Todo (Todo.Mvc.Ui.Models)                        ⌄

Options:
   ☐ Create as a partial view
   ☑ Reference script libraries
   ☑ Use a layout page:

   [                                              ]    [ ... ]

   (Leave empty if it is set in a Razor _viewstart file)

                                        [ Add ]    [ Cancel ]

-Click **Add** button and Detail.cshtml will be created like below

**Detail.cshtml**

```
@model Todo.Mvc.Ui.Models.Todo
@{
    ViewData["Title"] = "Details";
}
<h2>Details</h2>
<div>
    <h4>Todo</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Id)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Id)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
    </dl>
</div>
<div>
    @Html.ActionLink("Edit", "Edit", new { /* id = Model.PrimaryKey */ }) |
    <a asp-action="Index">Back to List</a>
</div>
```

-As usual update our workaround change from Todo.Mvc.Ui.Models.Todo name space to SmartIT.Employee.MockDB.Todo

-Update Edit link id from    @Html.ActionLink("Edit", "Edit", new { /* id = Model.PrimaryKey */ })  to

new { id = Model.Id })

-Here is the updated **Details View**

```
@model SmartIT.Employee.MockDB.Todo
@{
    ViewData["Title"] = "Details";
}
<h2>Details</h2>
<div>
    <h4>Todo</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Id)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Id)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
    </dl>
</div>
<div>
    @Html.ActionLink("Edit", "Edit", new { id = Model.Id }) |
    <a asp-action="Index">Back to List</a>
</div>
```

-Lets update the Details method that returns the selected Todo item to the Details view.

```
// GET: Todo/Details/5
public ActionResult Details(int id)
{
  return View();
}
```
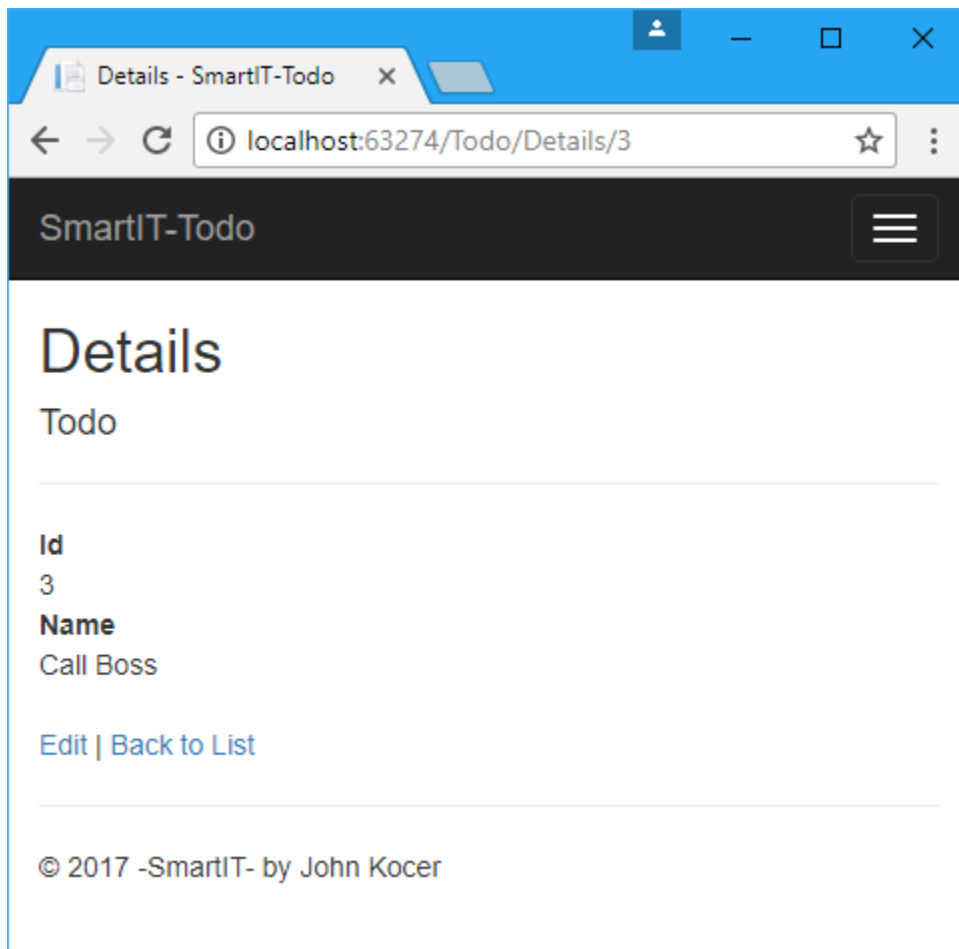
-Using the id Details method parameter find the Todo item and sent to the view.

-Here is the updated Details method.

```
public ActionResult Details(int id)
{
  var findTodo = _todoRepository.FindById(id);
  return View(findTodo);
}
```

-Press F5 and run the project.

-Pick Call Boss Todo item and see the detail view like below.



-Add Delete View

-Go to TodoController and Update the Delete method from
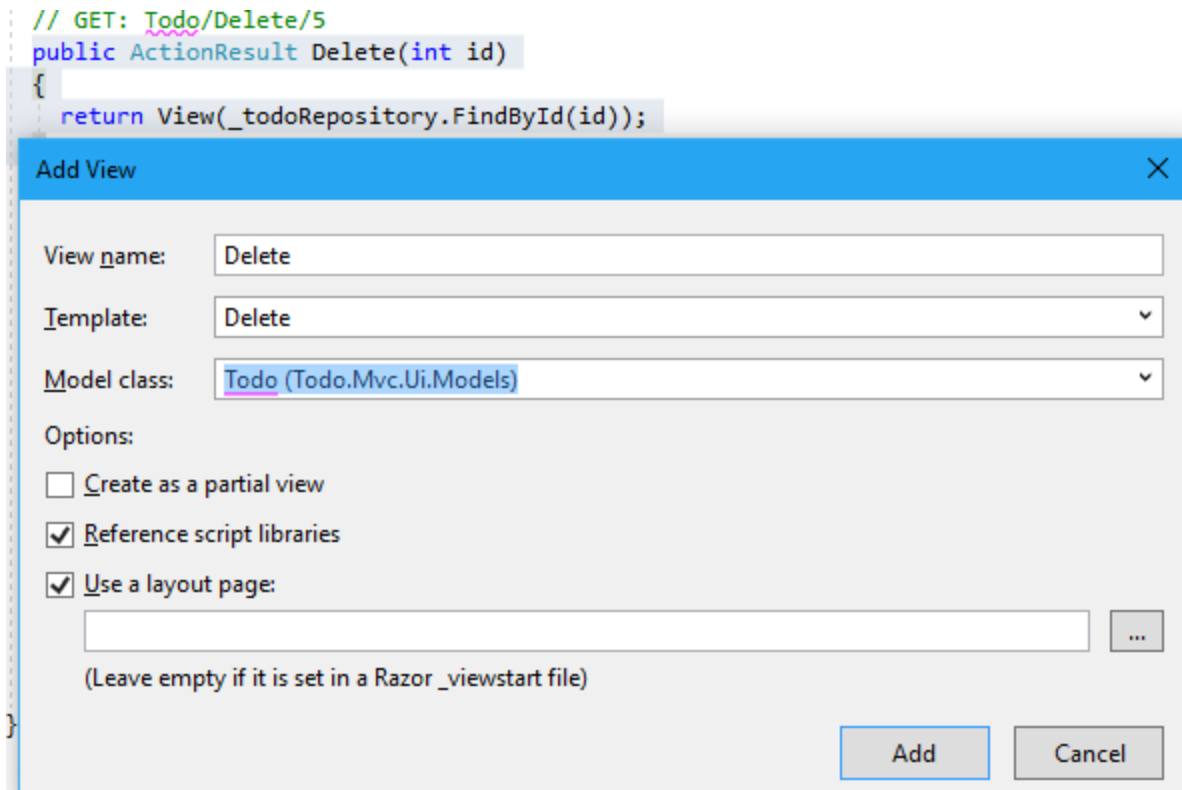
```
// GET: Todo/Delete/5
public ActionResult Delete(int id)
{
    return View();
}
```

To find delete Todo item using passed id number and pass to the View as argument like below.

```
public ActionResult Delete(int id)
{
    return View(_todoRepository.FindById(id));
}
```

-Add Delete View like below selecting View name as Delete, Template as Delete and Model class as Todo.

```
// GET: Todo/Delete/5
public ActionResult Delete(int id)
{
    return View(_todoRepository.FindById(id));
```

**Add View**                                                                    ✕

View name:        Delete

Template:         Delete                                                      ˅

Model class:      Todo (Todo.Mvc.Ui.Models)                                   ˅

Options:

☐ Create as a partial view

☑ Reference script libraries

☑ Use a layout page:

[                                                              ]  [ ... ]

(Leave empty if it is set in a Razor _viewstart file)

                                              [   Add   ]   [  Cancel  ]

-Click the Add button.

-Using out workaround update Todo name space from @model Todo.Mvc.Ui.Models.Todo to SmartIT.Employee.MockDB.Todo

-Here is the updated **Delete.cshtml** view page.

```
@model SmartIT.Employee.MockDB.Todo
@{
    ViewData["Title"] = "Delete";
}
<h2>Delete</h2>
<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Todo</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Id)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Id)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
    </dl>

    <form asp-action="Delete">
        <input type="submit" value="Delete" class="btn btn-default" /> |
        <a asp-action="Index">Back to List</a>
    </form>
</div>
```

-Update HttpPost Delete method  section // TODO: Add delete logic here

```csharp
// POST: Todo/Delete/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Delete(int id, IFormCollection collection)
    {
      try
      {
        // TODO: Add delete logic here

        return RedirectToAction(nameof(Index));
      }
      catch
      {
        return View();
      }
    }
```
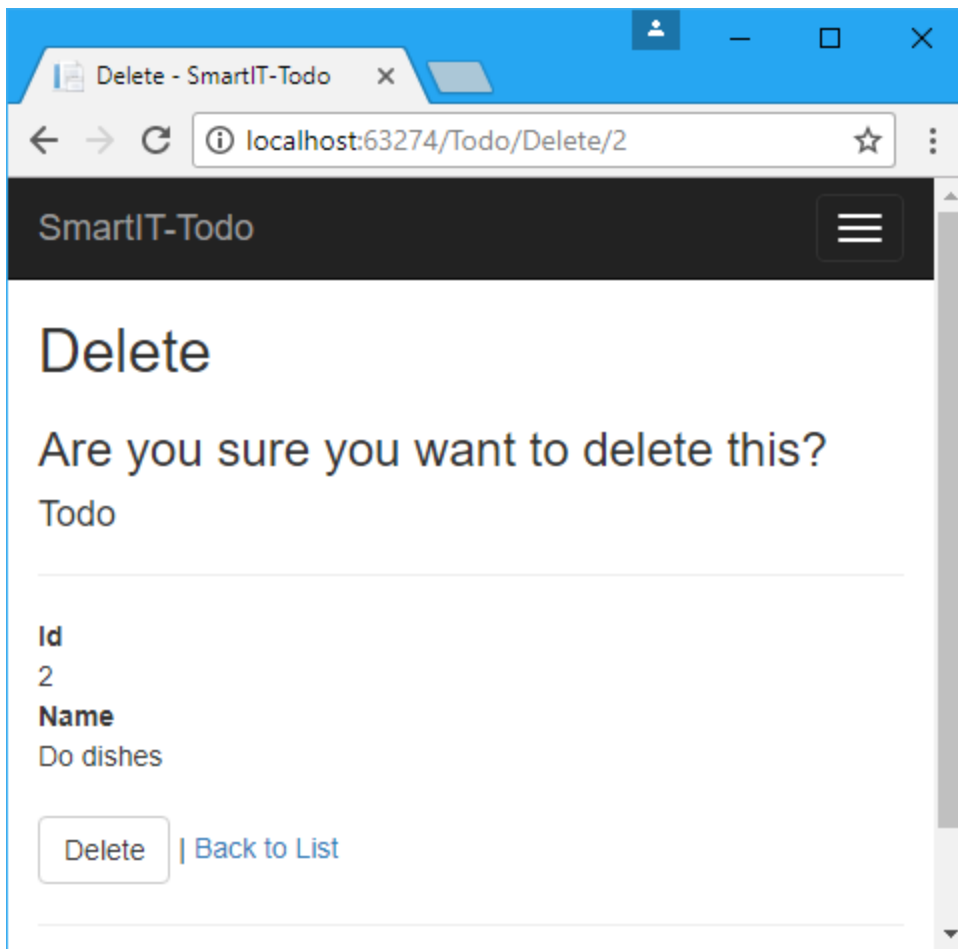
To below updated Delete HttpPost Delete method.

```csharp
public ActionResult Delete(int id, IFormCollection collection)
    {
      try
      {
        var findTodo = _todoRepository.FindById(id);
        _todoRepository.Delete(findTodo);

        return RedirectToAction(nameof(Index));
      }
      catch
      {
        return View();
      }
    }
```
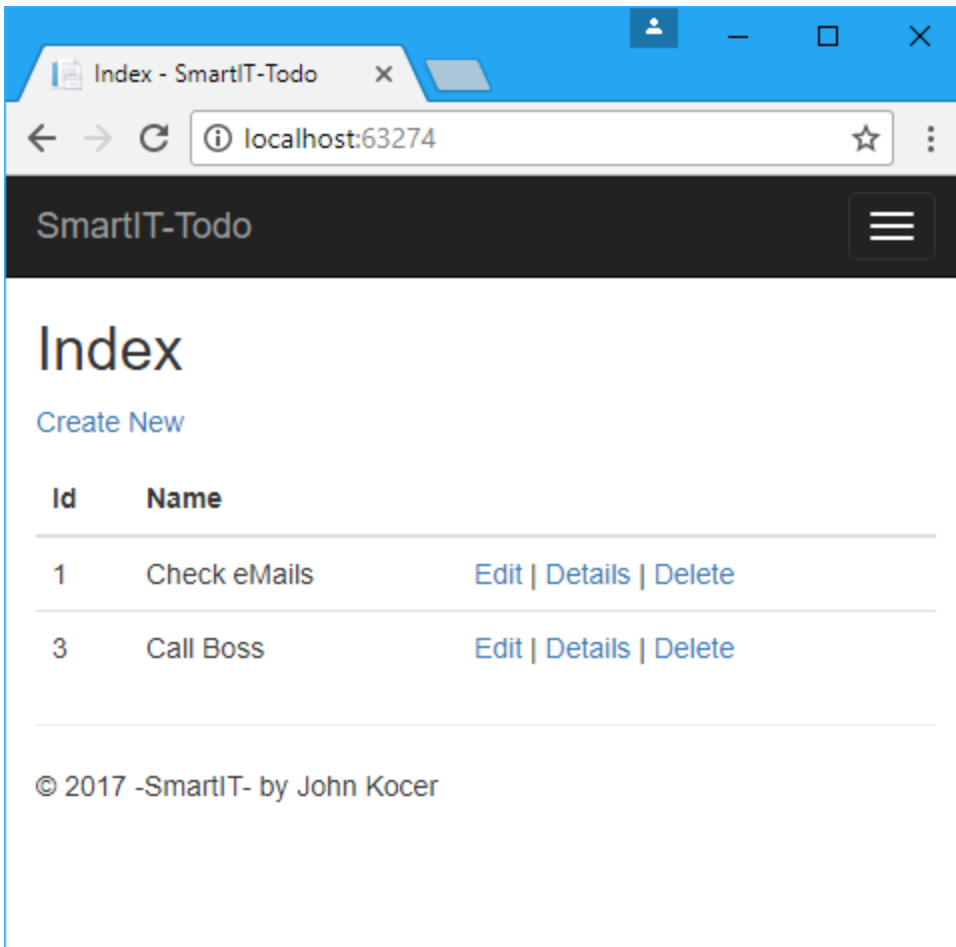
-Press F5 and run the project again.

-Select Do Dishes link and below delete view shows.

-Click Delete button.

-We can see that Do Dishes Todo item has been deleted.

-Finally comment out  workaround class we don't need any more.

## Summary

In this article, we have learned

-How to consume Todo inmemory database using TodoRepository.
-How to create custom ASP.NET MVC custom controller with CRUD operations.
-List of Objects
-Create a new insert object via View
-Update and object via Edit View
-Delete an Item via Delete View.
-Write HttpPost Create API method.
-Write Httppost Edit API method.
-Write HttpPost Delete API method.
-SmartIT DebugTraceHelper tool


Download source code from GitHub: https://github.com/SmartITAz/TodoMvcSolution


**Lab Exercise**

A lab exercise for you to demonstrate what have you learned from this training material to create your own Employee CRUD operation using EmployeeRepository included in this training material.

-You can follow above steps to create your own Employee CRUD ASP.NET MVC application.

```csharp
    //Use below Employee repository to created your CRUD operation
    EmployeeRepository _employeeRepository= new EmployeeRepository();
    _employeeRepository.Add(new Employee() { Name = "Mat Stone", Gender = "Male", DepartmentId = 2,
Salary = 8000 });
    _employeeRepository.CDump("Employees");
 //  DebugHelper.cs(29): Employees
 //{
"Items":[{"Id":1,"Name":"Mike","Gender":"Male","Salary":8000,"DepartmentId":1,"Department":null},
 //{"Id":2,"Name":"Adam","Gender":"Male","Salary":5000,"DepartmentId":1,"Department":null},
 //{"Id":3,"Name":"Jacky","Gender":"Female","Salary":9000,"DepartmentId":1,"Department":null},
 //{"Id":4,"Name":"Mat
Stone","Gender":"Male","Salary":8000,"DepartmentId":2,"Department":null}],"Count":4}
```