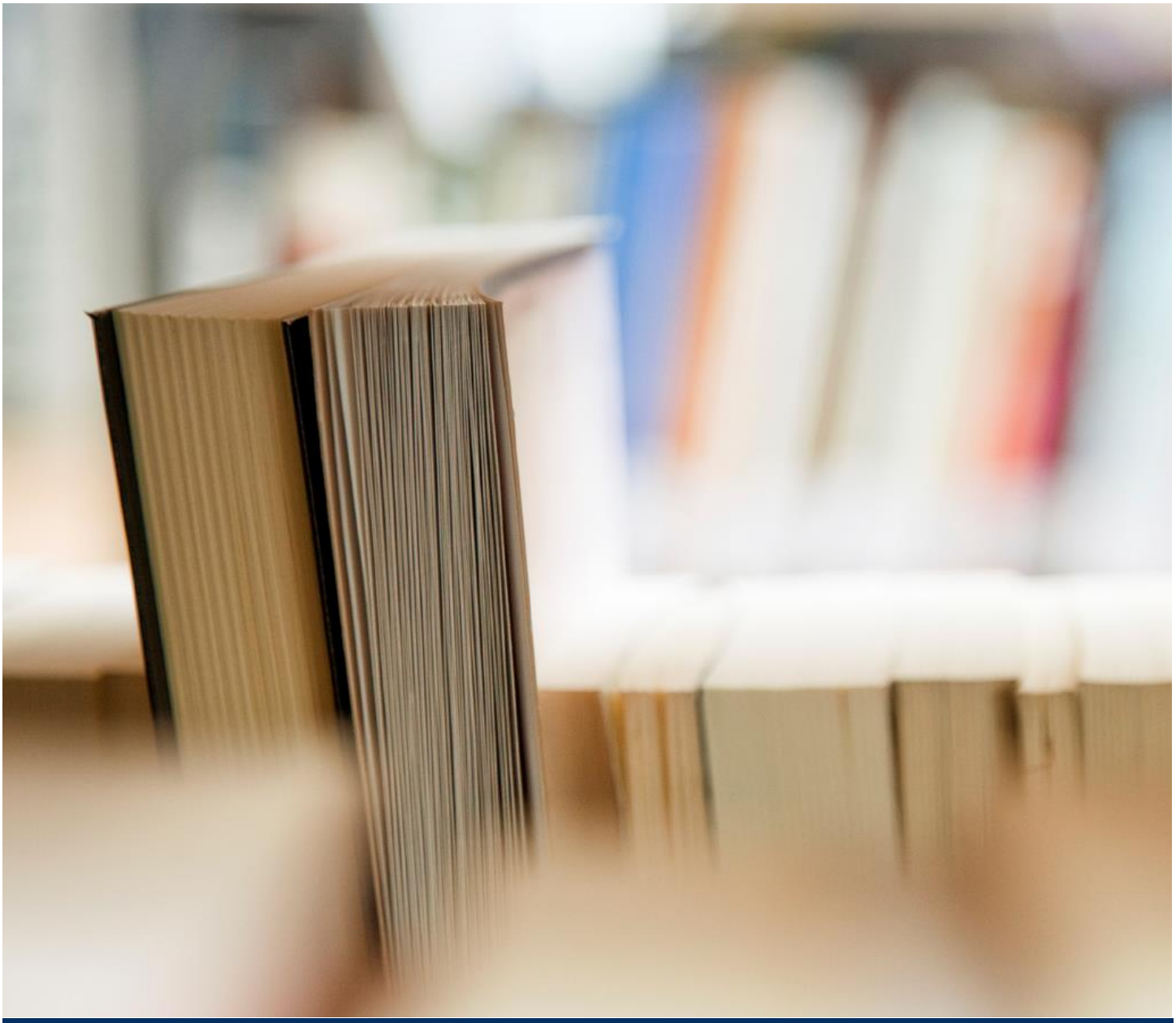


IronPdf Guide with .NET Core

Version 2019.8.9

IronSoftware

Ahmed Aboelmagd

Contents

Introduction

Adding Pdf file generation in ASP.Net MVC project is a cumbersome task; also converting MVC views, HTML file, and online web pages to Pdf is a very hard and complex problem,

<-- Text in introduction part -->

And you can visit http://www.ironpdf.com for more information.

You can download sample project from GitHub (https://github.com/magedo93/IronSoftware.git)

Document Organization

- Chapter 1 Install IronPdf: this part describes How to install IronPdf to existing project.
- Chapter 2 Convert to pdf: this part describes different methods to create Pdf from different sources like (URL, HTML, MVC views) and different advanced options that we can use for different output pdf settings,
  Also how to deploy your project to different images (Linux, windows)
- Chapter 3 Working with Pdf Document: this part describes how to different manipulation capabilities on created pdf files like adding headers or footers, merge files, add STAMP and other features.
- Summery
  brief conclusion about what we have learned in this document
- Appendix (A) Dockers: this part describes what is dockers and how to use it
- About author
  brief about document author

Chapter 1: Install IronPdf

IronPdf can be installed and used on all of .NET projects type like windows application, ASP.NET MVC and .Net Core Application.

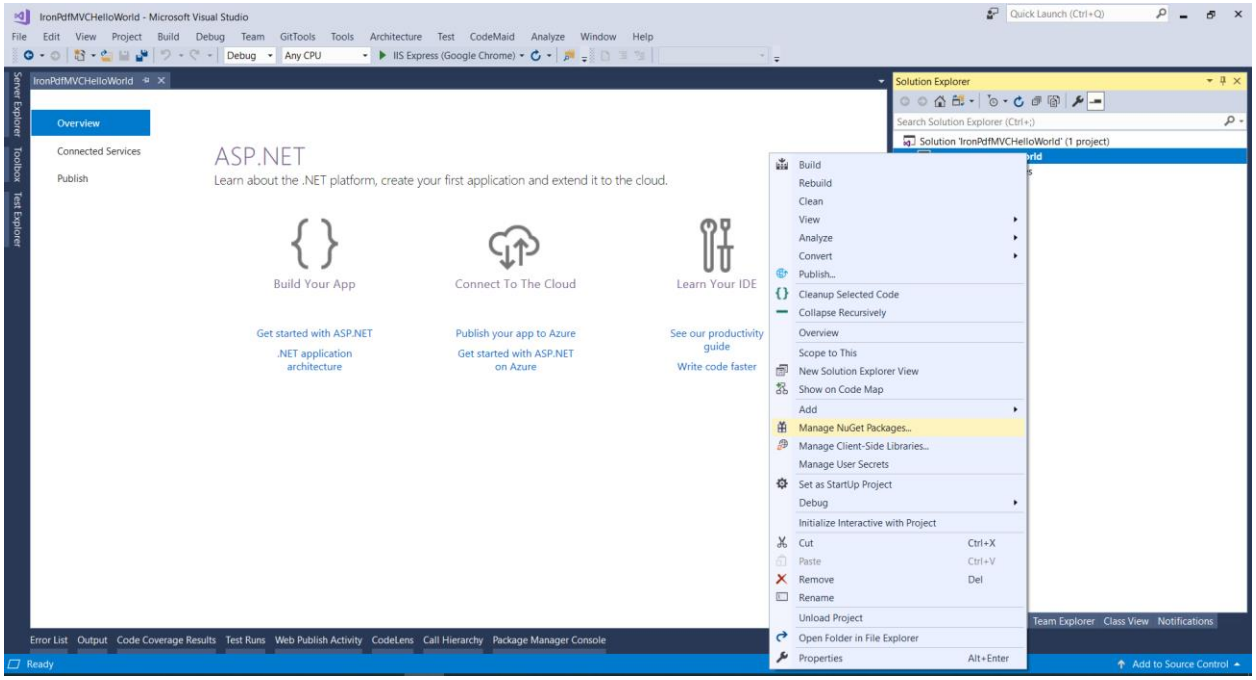
To add IronPdf library to the project we have two ways, from Visual studio editor install using NuGet or command line using package console manager as following: -

Install using NuGet

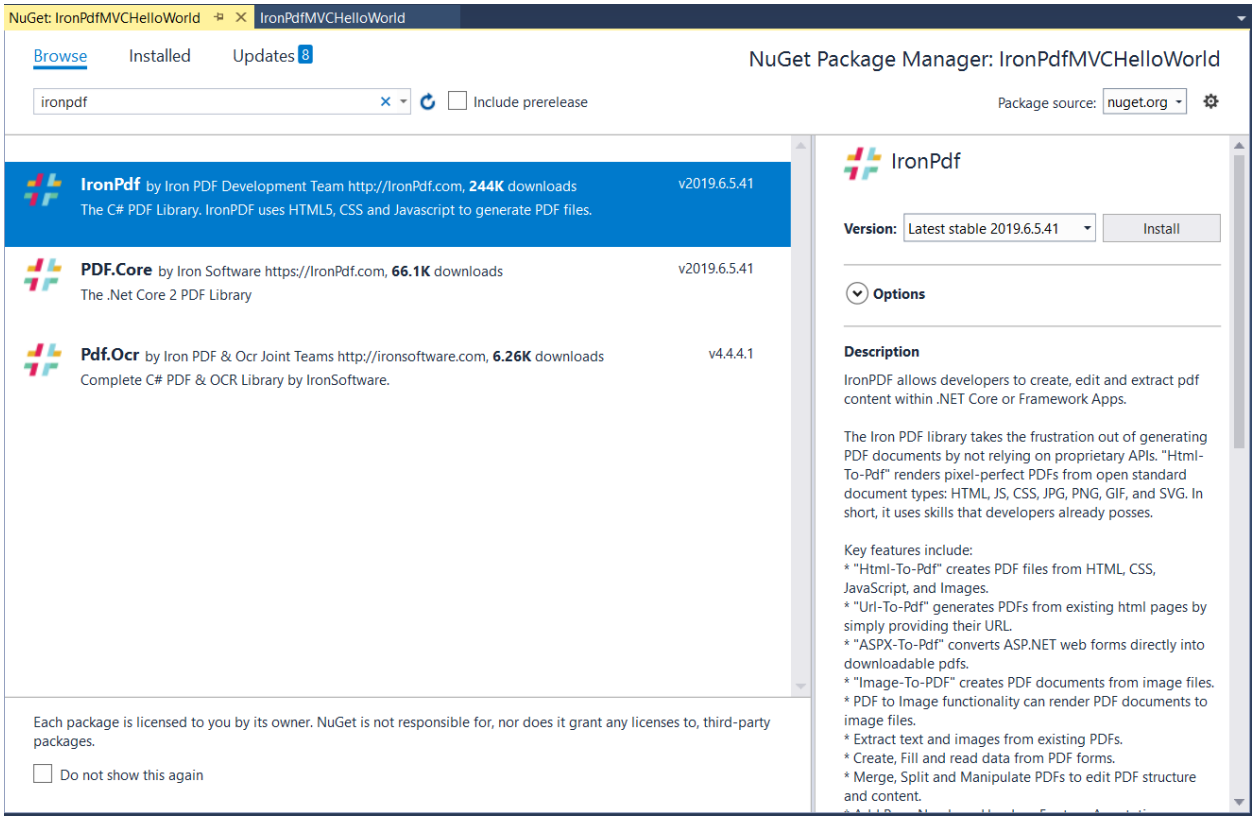
To add IronPdf library to our project using NuGet we can do it using visualized interface (NuGet Package Manager) or by command using Package Manager Console as following: -

Using NuGet Package Manager

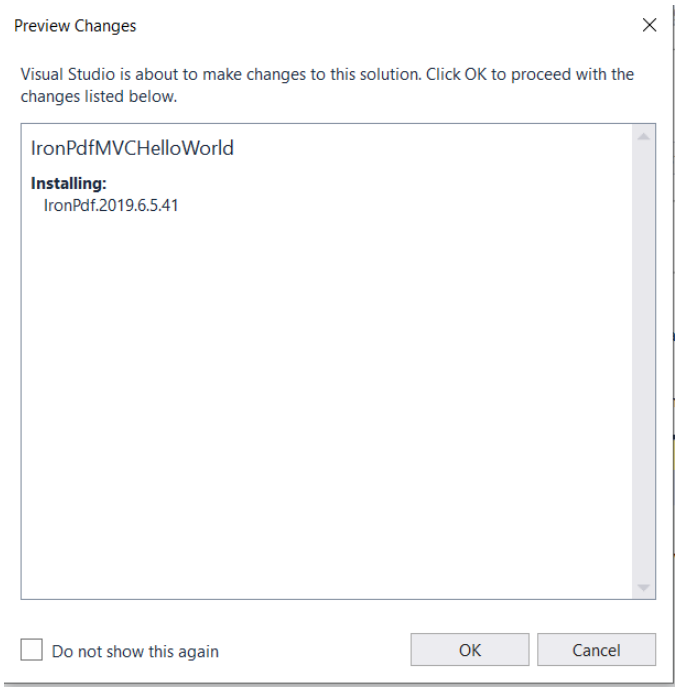
1- Using mouse -> right click on project name -> Select manage NuGet Package



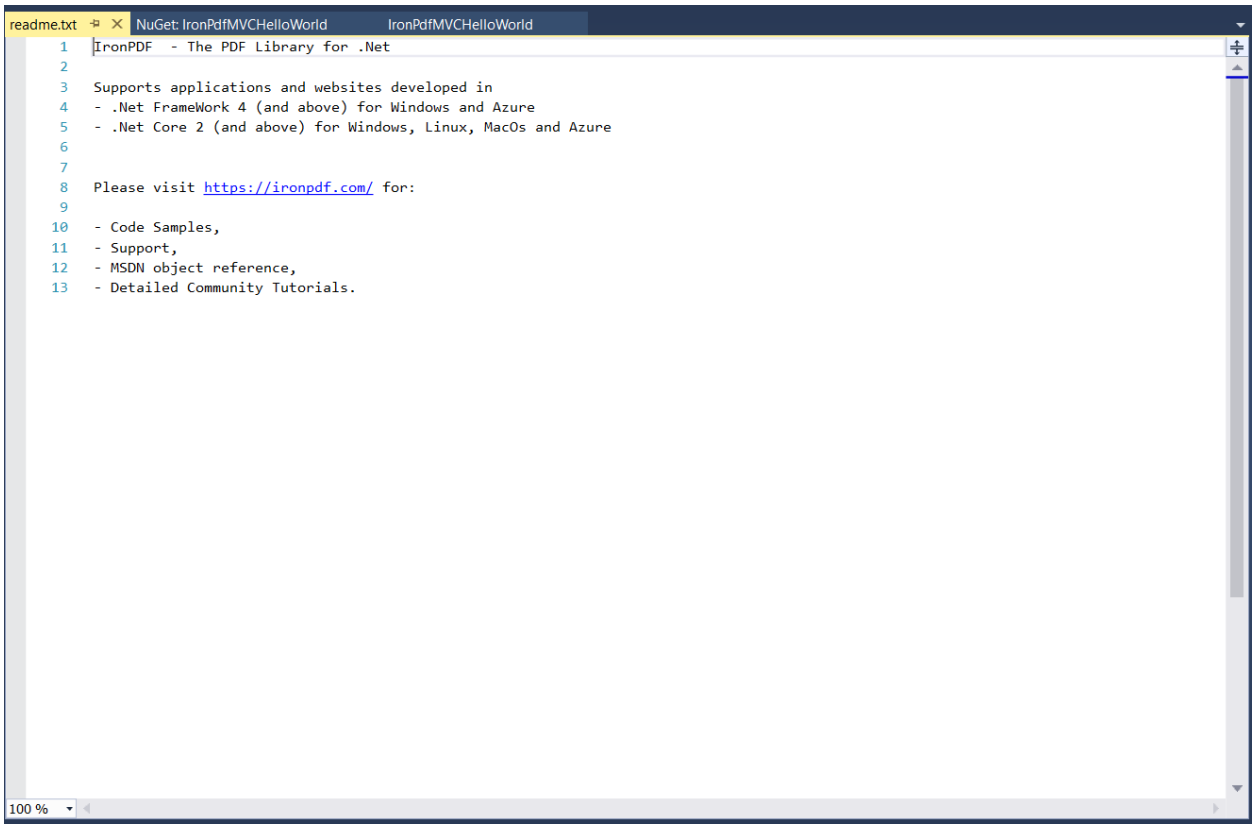
2- From brows tab -> search for IronPdf -> Install



3- Click Ok

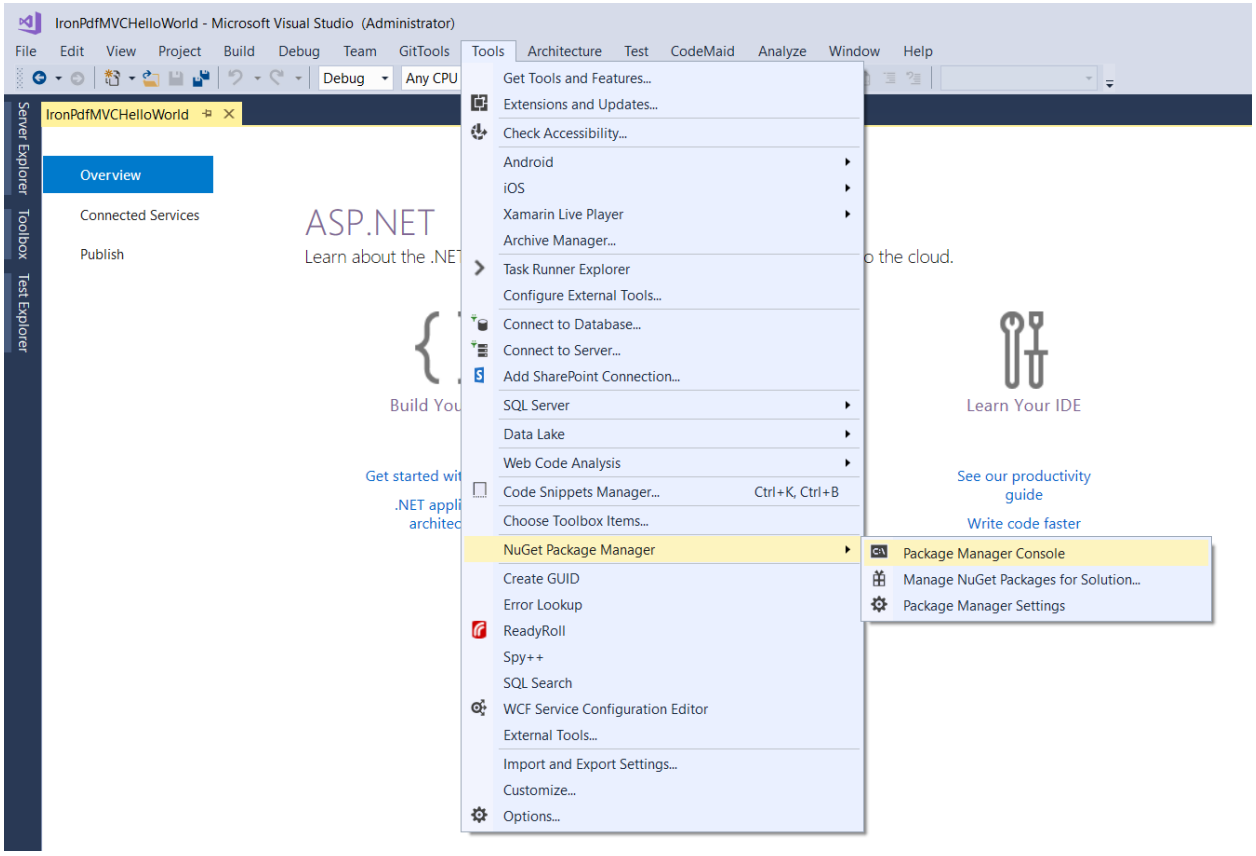


4- And we are Done

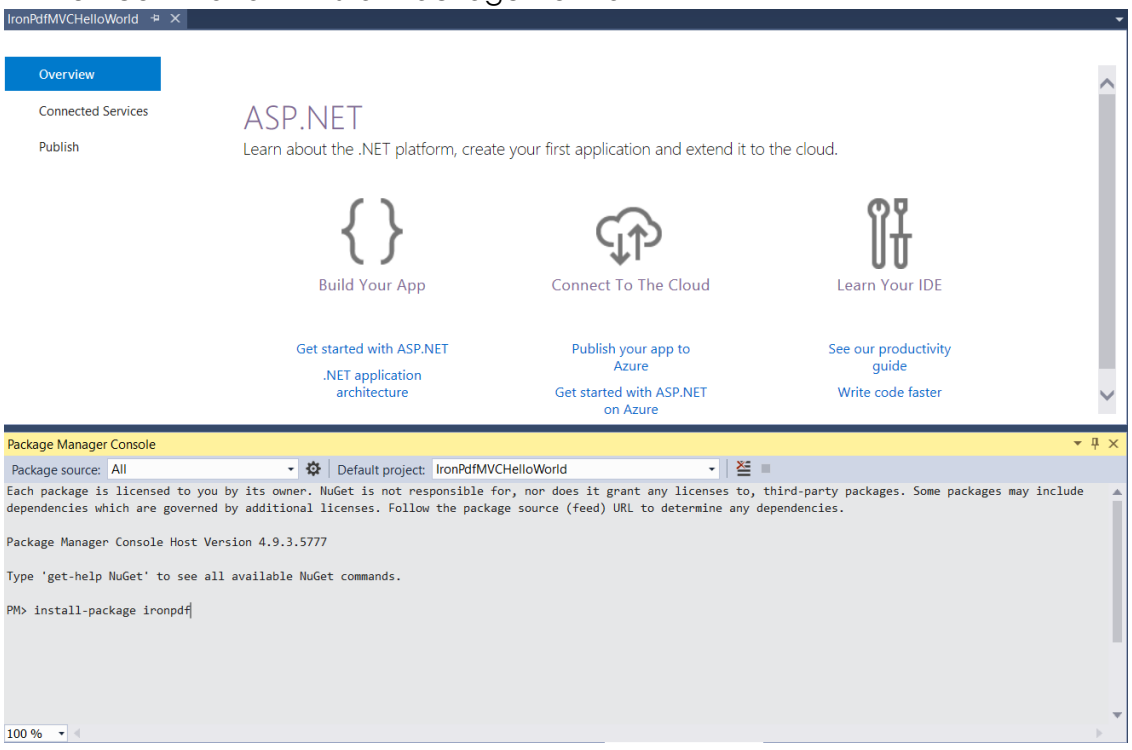


Using NuGet Package Console manager

1- From tools -> NuGet Package Manager -> Package Manager Console



2- Run command -> Install-Package IronPdf



We can now practice by implementing Hello World using .NET Core Console Application and another Hello world using .NET Core Web Application

Sample 1: HelloWorldConsole Console Application

Open visual studio => new => project

Sample 2: HelloWorldCore .Net Core Web Application

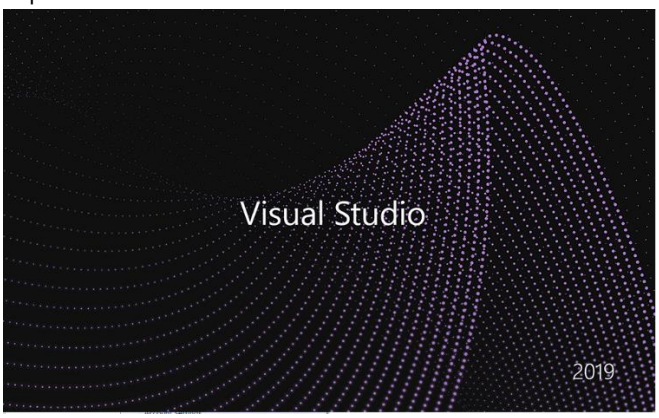
Chapter 2: Convert to Pdf

Convert online website to Pdf

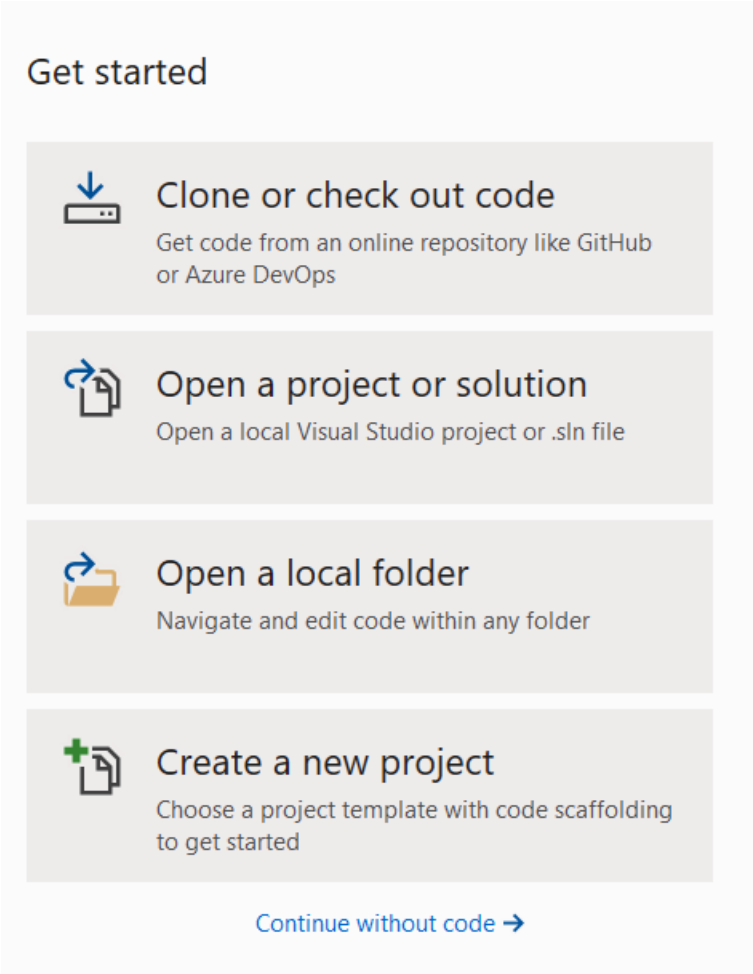
Sample: ConvertUrlToPdf console application

Follow coming steps to create new Asp.NET MVC Project

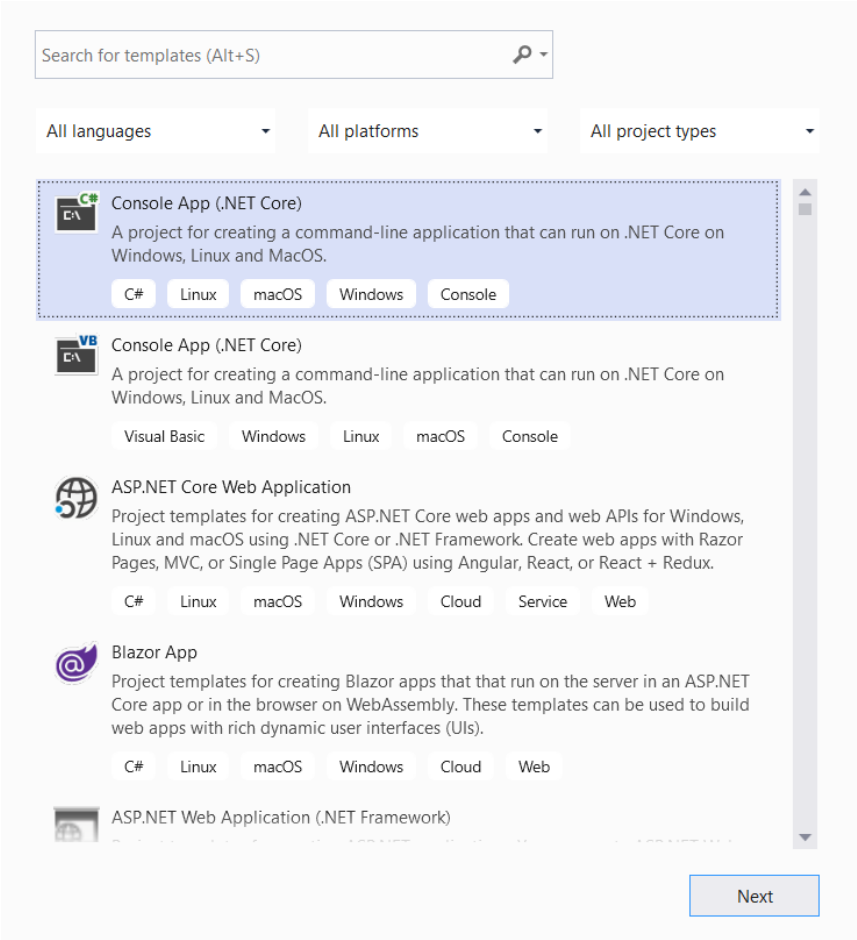
1- Open visual studio



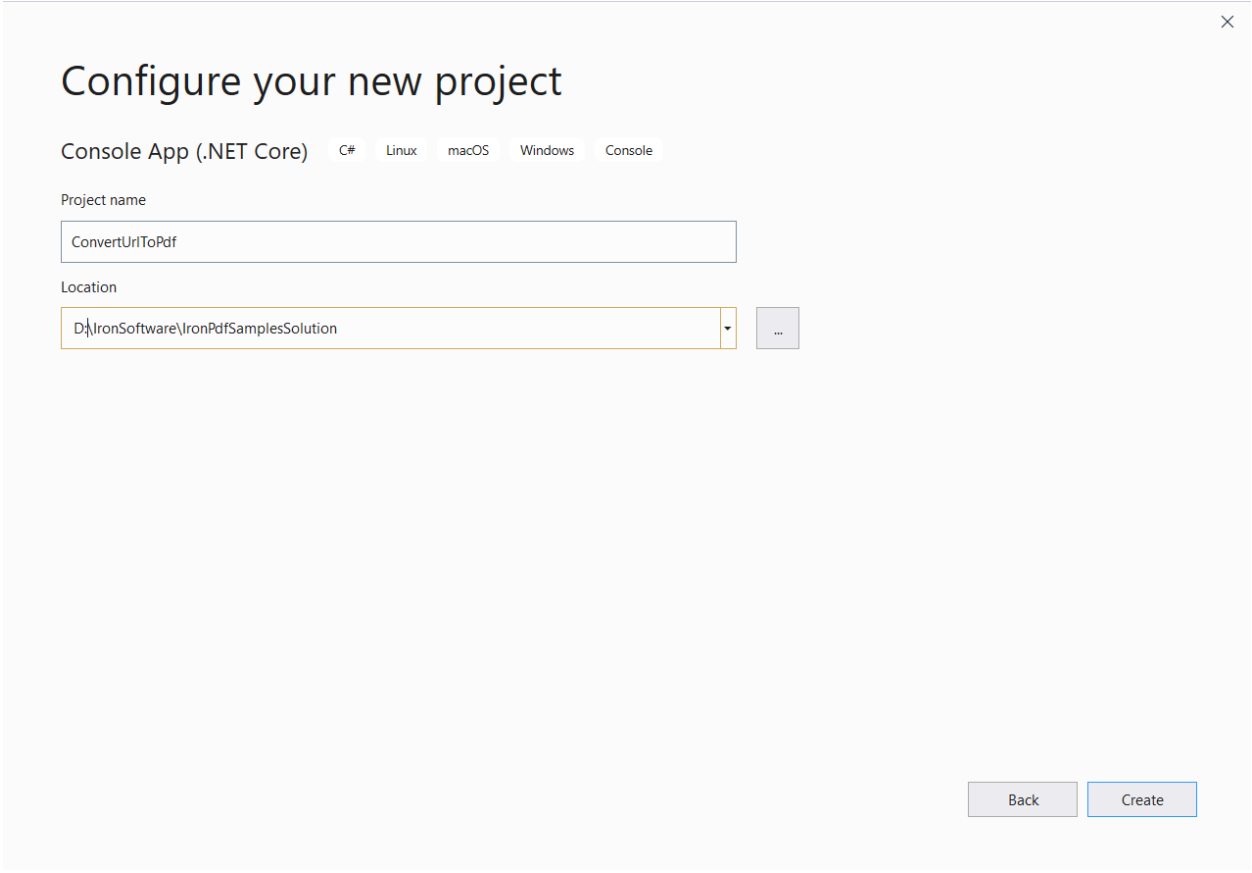
2- Choose Create new project



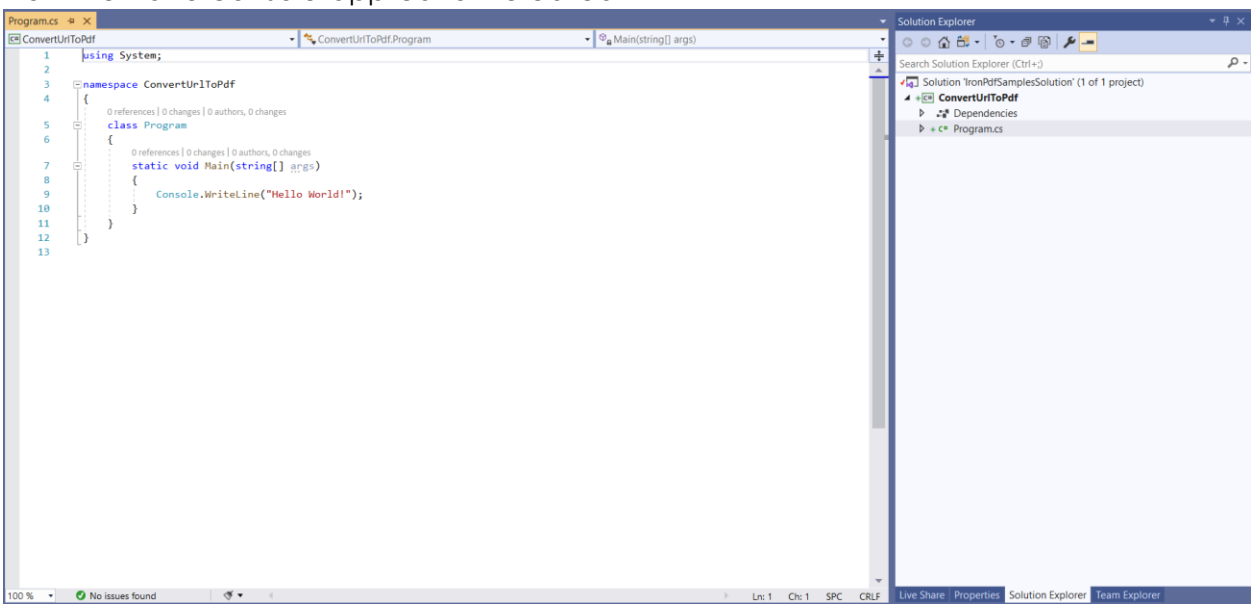
3- Choose Console App (.NET Core)



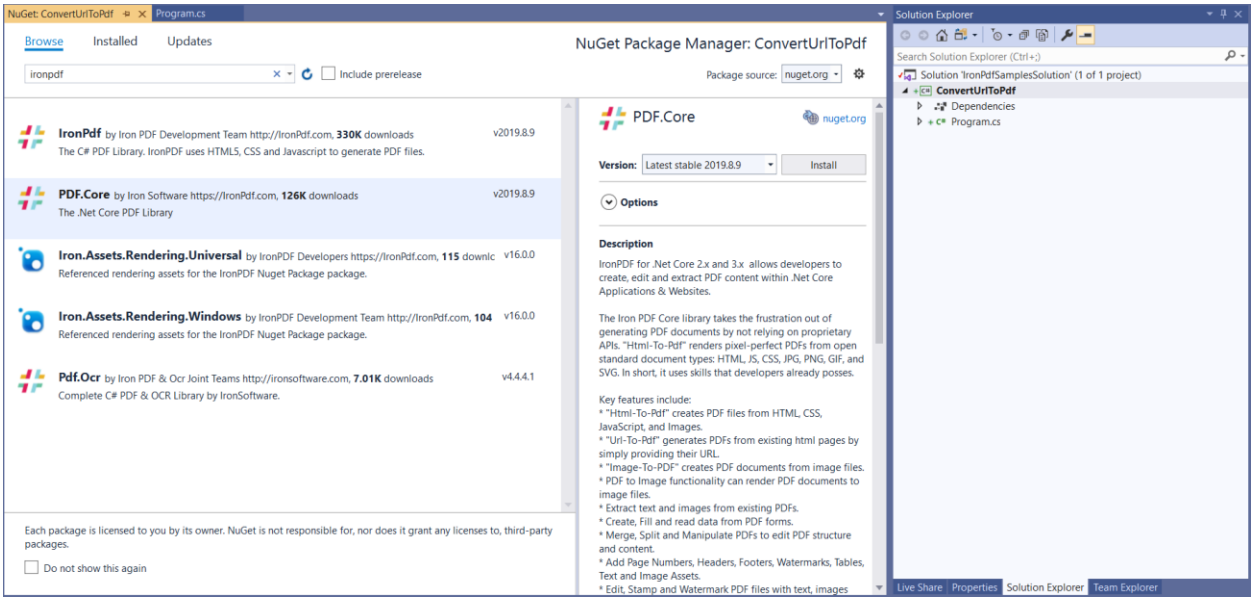
4- Give our sample name “ConvertUrlToPdf” and click create

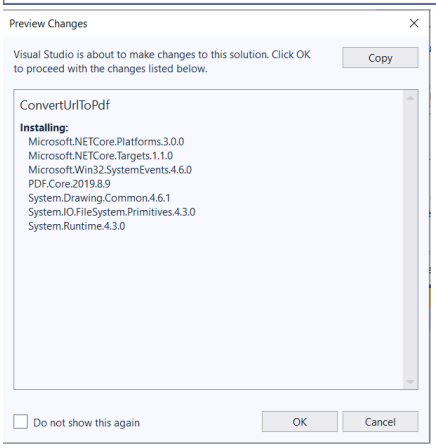


5- Now we have console application created



6- Add IronPdf => click install

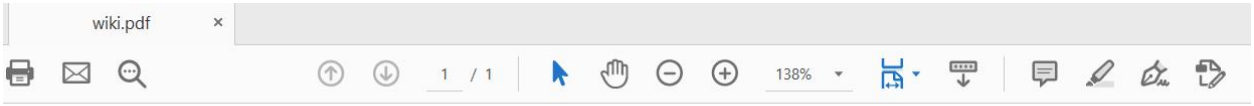


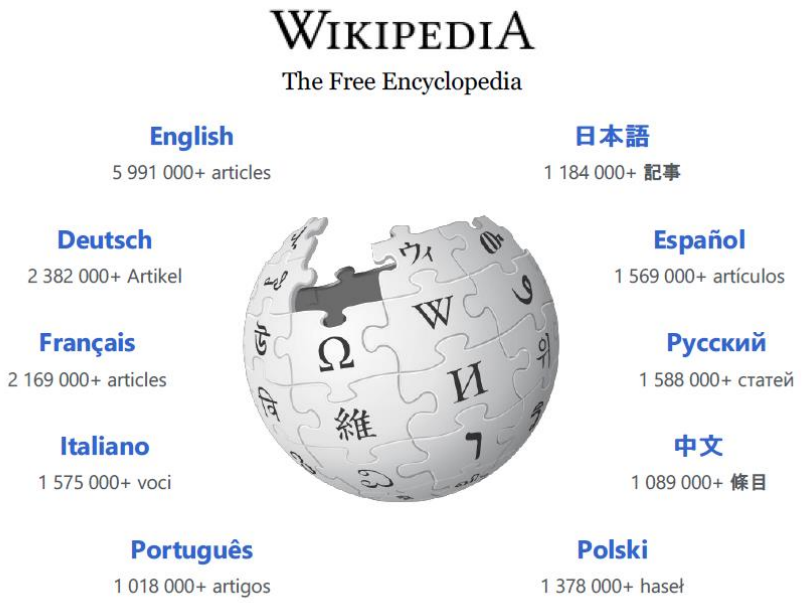


7- Add our first few lines that render Wikipedia website main page to pdf

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
    var render = new IronPdf.HtmlToPdf();
    var doc = render.RenderUrlAsPdf("https://www.wikipedia.org/");
    doc.SaveAs($@"{AppDomain.CurrentDomain.BaseDirectory}\wiki.pdf");
}
```

8- Run and check created file wiki.pdf





Convert HTML to Pdf

Sample: ConvertHTMLToPdf Console application

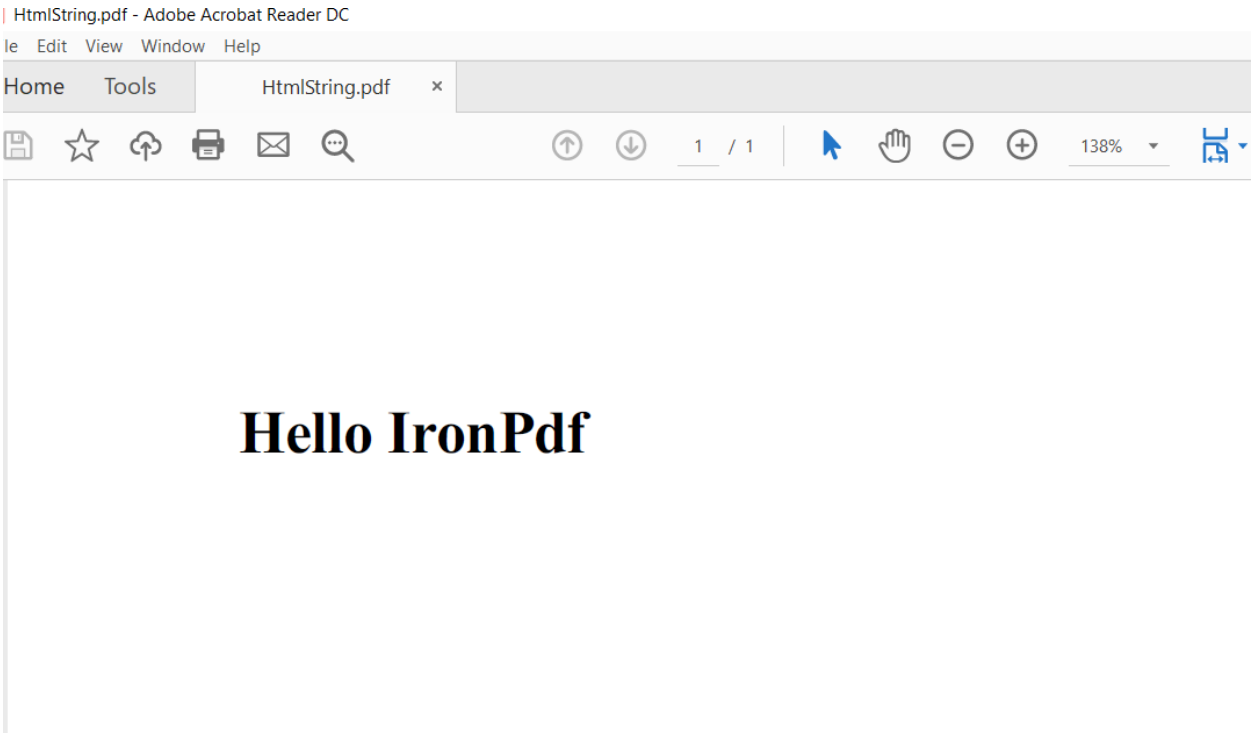
To render HTML to Pdf we have 2 way

1- Write html into string then render it
2- Write html into file and pass it path to ironPdf to render it

Rendering html string sample code like

```
static void Main(string[] args)
      {
          var render = new IronPdf.HtmlToPdf();
          var doc = render.RenderHtmlAsPdf("<h1>Hello IronPdf</h1>");
          doc.SaveAs($@"{AppDomain.CurrentDomain.BaseDirectory}\HtmlString.pdf");
       }
```

Result pdf like



Convert MVC Partial view to Pdf

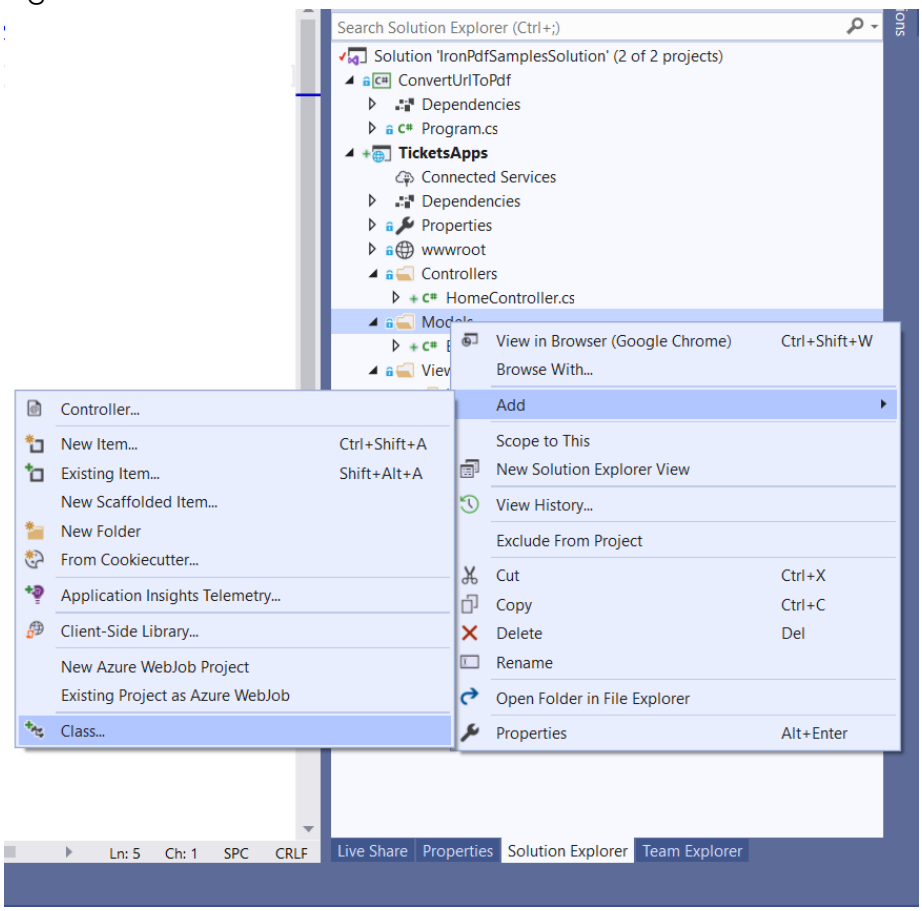
Let's implement real life example I'll choose online ticketing site you open the site and navigate to book ticket then fill required information's then you get your copy as downloadable pdf file so let's go.

We will go throw this step: -

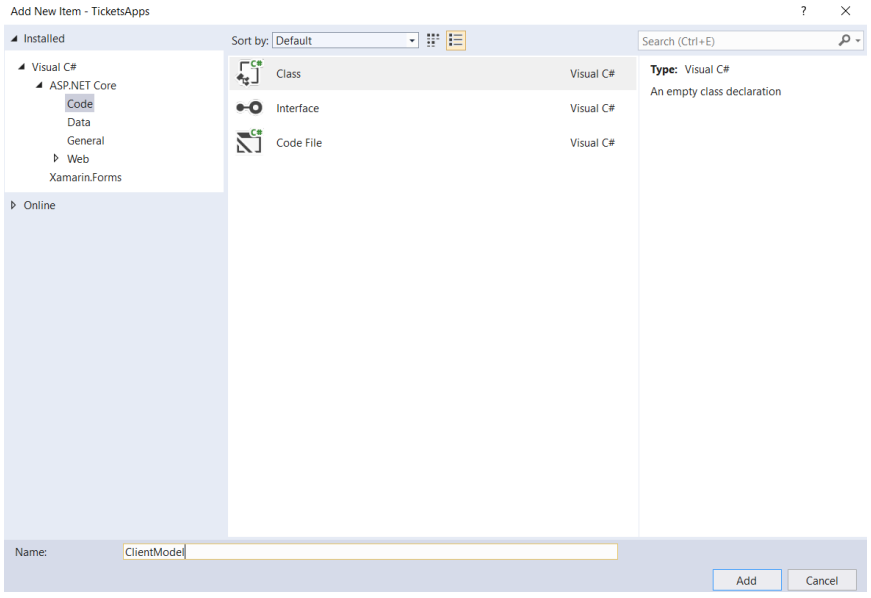
1- Create client object model
2- Create client services (add, view)
3- Add pages (Register, view)
4- Download pdf ticket

So now I'll start by creating client object model

1.1. Right click on models' folders choose to add choose class



1.2. Name the model "ClientModel" then click add



1.3. add to ClientModel attributes Name , phone and email and make them all required by adding required attribute over them as follow

```
public class ClientModel
{
    [Required]
    public string Name { get; set; }
    [Required]
    public string Phone { get; set; }
    [Required]
    public string Email { get; set; }
}
```

2. Step 2 add services
   2.1. Create folder and with name "services"
   2.2. Then add class with name "ClientServices"
   2.3. add static object of type "ClientModel" to use it as repository
   2.4. add two functions one for saving client to repository and 2nd to get saved clients

```
public class ClientServices
{
    private static ClientModel _clientModel;
    public static void AddClient(ClientModel clientModel)
    {
        _clientModel = clientModel;
    }
    public static ClientModel GetClient()
    {
       return _clientModel;
    }
}
```

3. 3rd steps book your ticket page
   3.1. From solution explorer right click over controller folder choose add then choose controller
   3.2. update html as follow

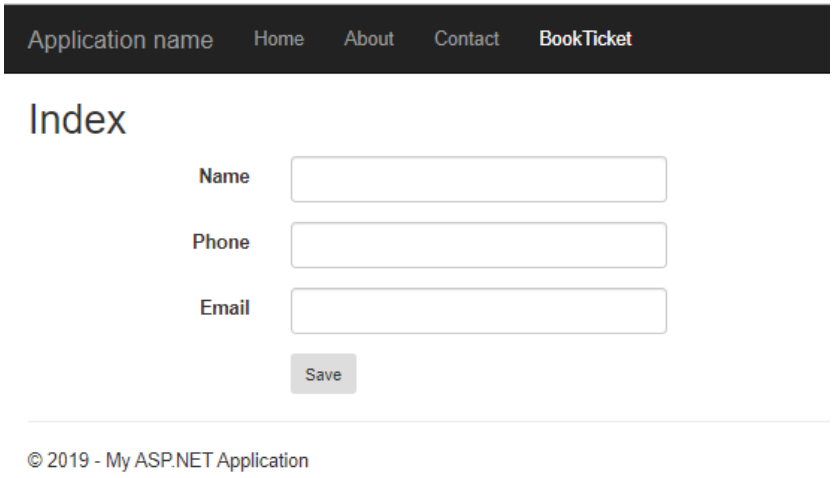
```
@model IronPdfMVCHelloWorld.Models.ClientModel
@{
    ViewBag.Title = "Book Ticket";
}

<h2>Index</h2>

@using (Html.BeginForm())
{
    <div class="form-horizontal">
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
```

```
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Name, "", new { @class =
"text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.Phone, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Phone, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Phone, "", new { @class
= "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.Email, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Email, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Email, "", new { @class
= "text-danger" })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-10 pull-right">
                <button type="submit" value="Save" class="btn btn-sm">
                    <i class="fa fa-plus"></i>
                    <span>
                        Save
                    </span>
                </button>
            </div>
        </div>
    </div>
}
```

3.3. add link to BookTicket Page to enable our website visitors to navigate to our new booking page by updating layout exist in path (view-> shared-> layout.chtml)

```
    <li>@Html.ActionLink("BookTicket", "Index", "BookTicket")</li>
```

3.4. result should be looks like this

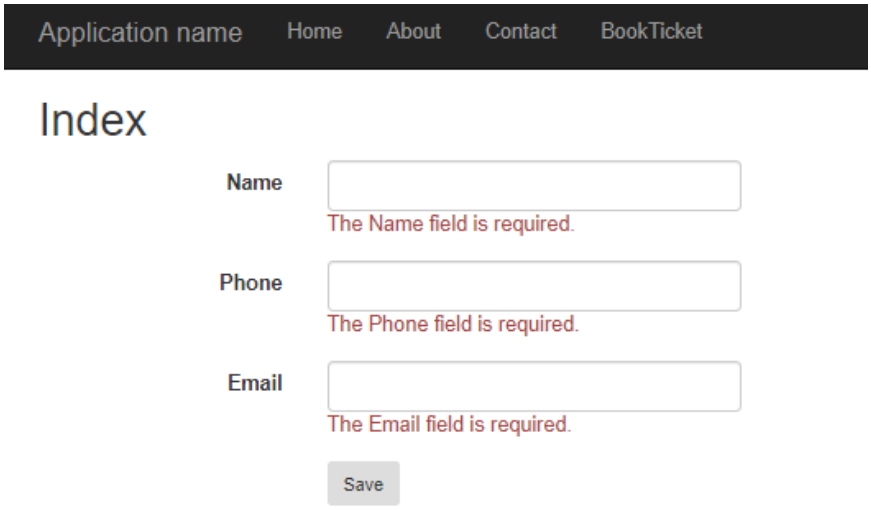


3.5. Now let's add the action that will validate and save the book information

3.6. Add another index action with attribute [HttpPost] to inform MVC engine that this action is for submitting data, I validate sent model if it valid code will redirect visitor to TicketView Page if not valid visitor will receive error validation messages on screen .

```
[HttpPost]
public ActionResult Index(ClientModel model)
{
    if (ModelState.IsValid)
    {
        ClientServices.AddClient(model);
        Return RedirectToAction("TicketView");
    }
    return View(model);
  }
```

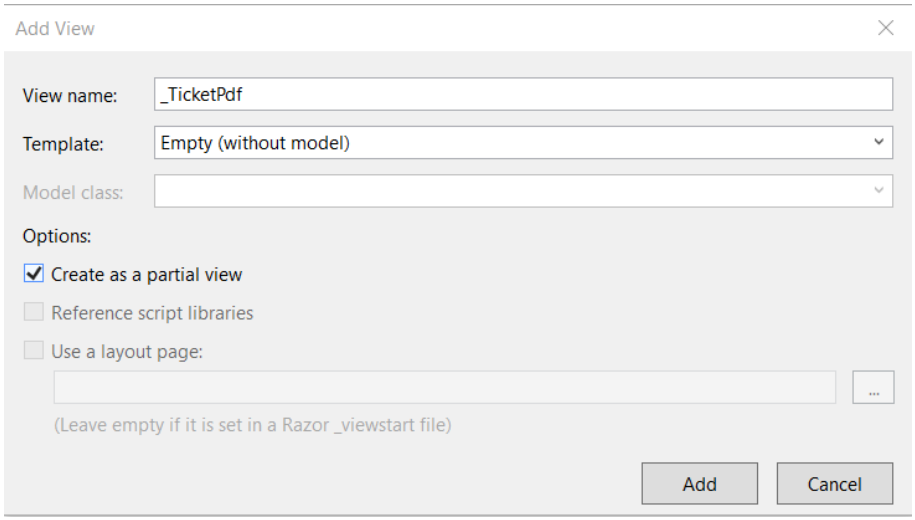
3.7. Sample of error messages



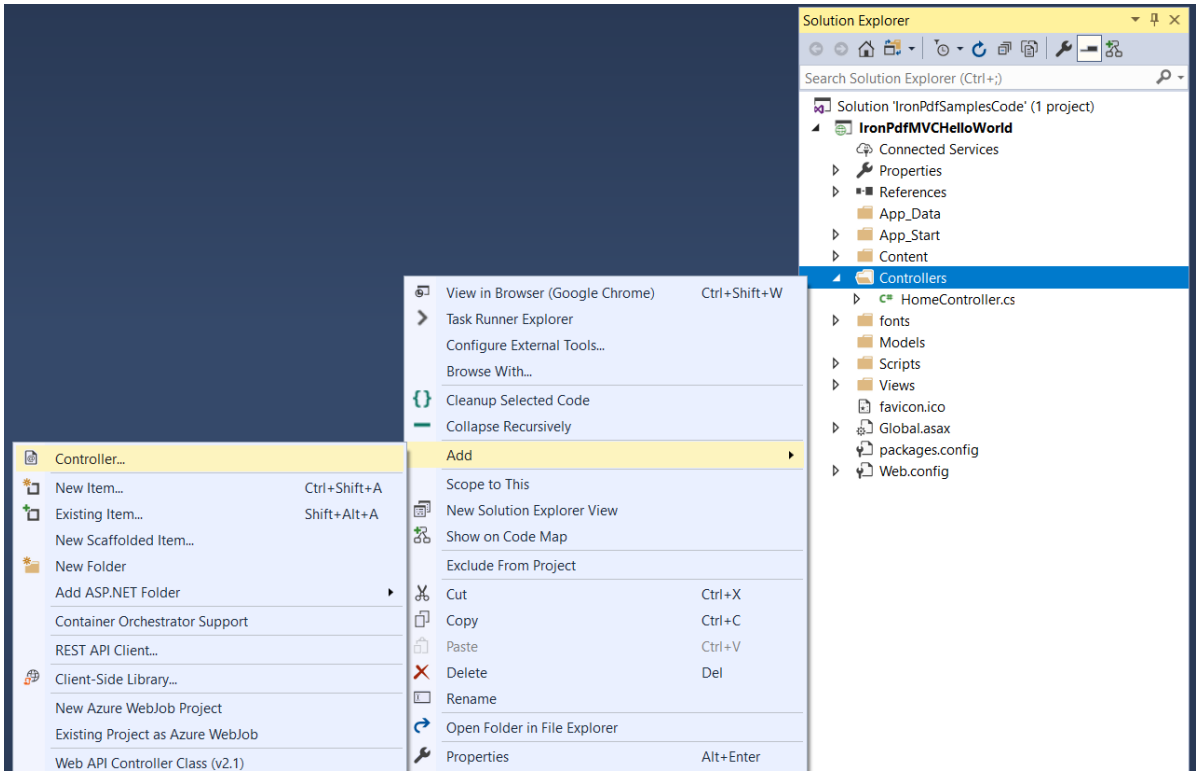
3.8. add TicketView to display our ticket

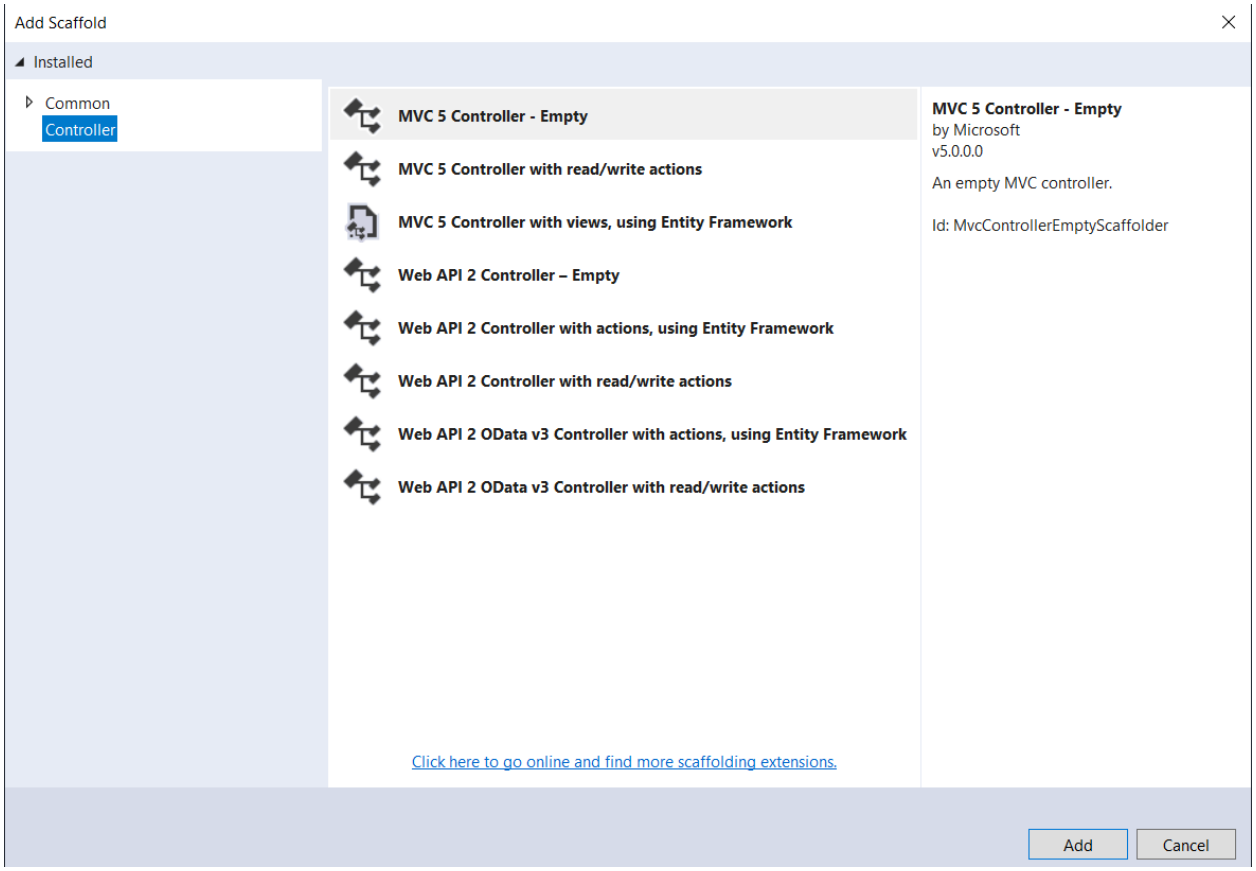
```
public ActionResult TicketView()
  {
      var ticket = ClientServices.GetClient();
      return View(ticket);
    }
```

3.9. Add its view

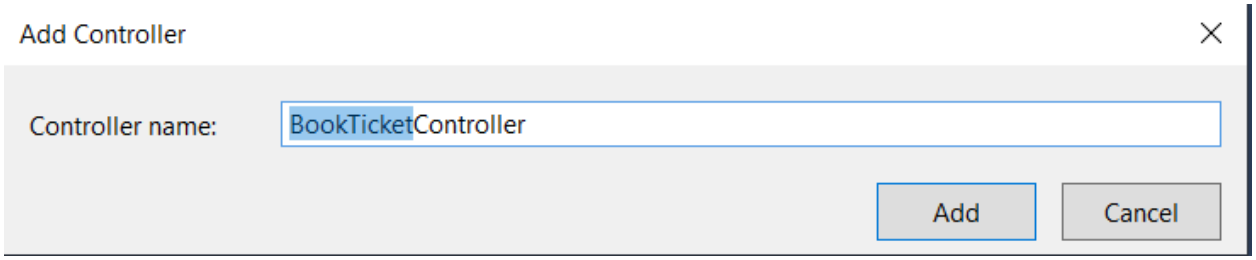


3.10.

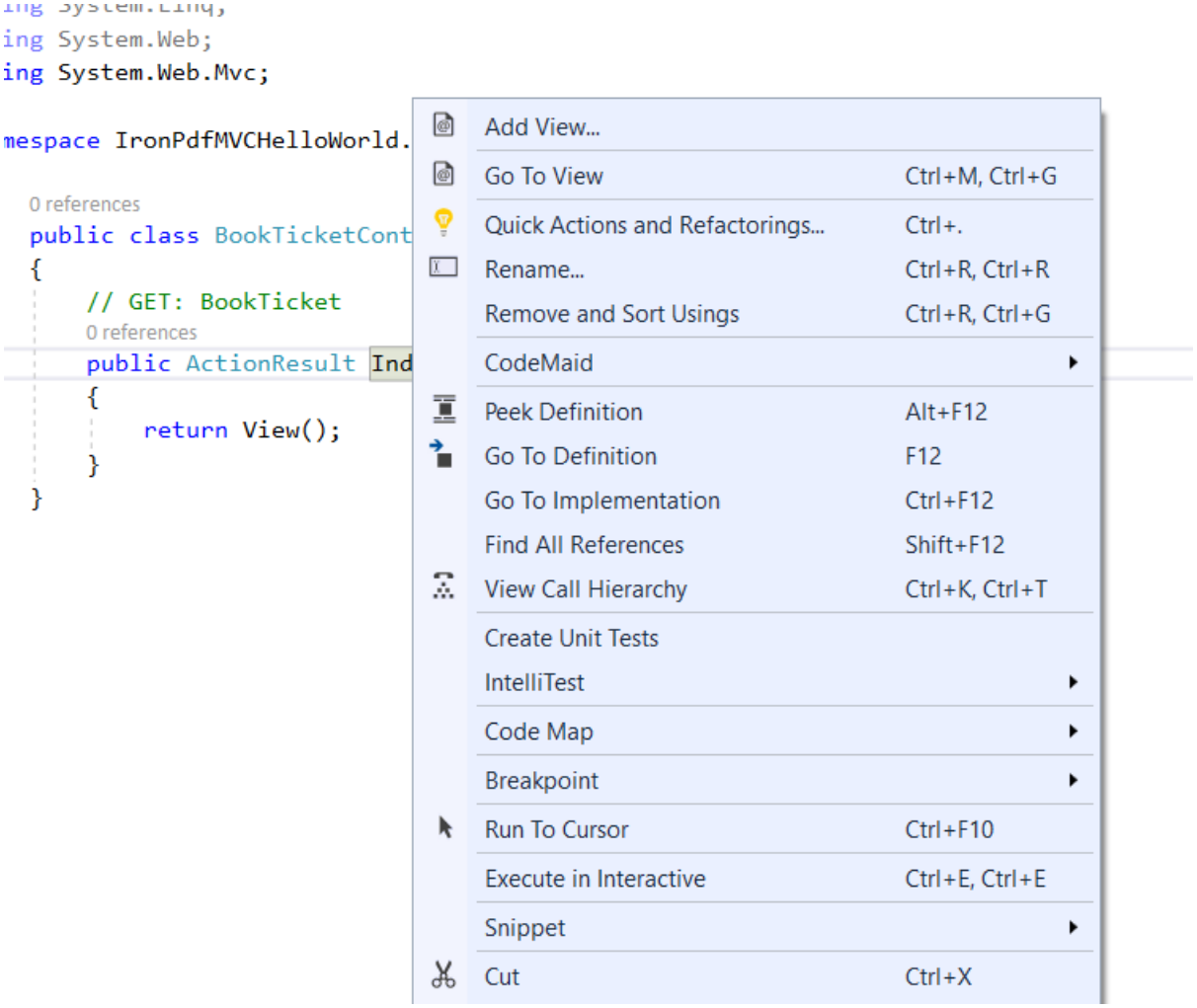
4. 

5. From opened window select MVC 5 Controller – Empty
6. 

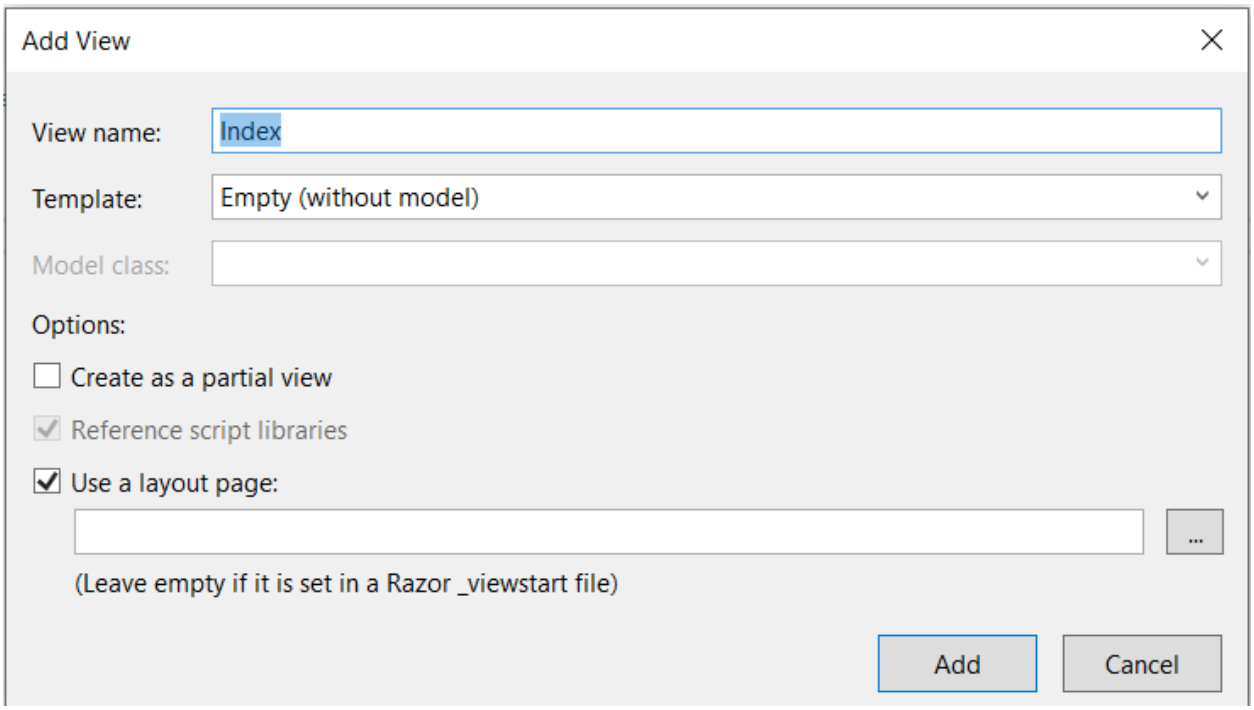
7. Name it BookTicketController



8. Right click on index function (or we called it action) and choose add view to add html



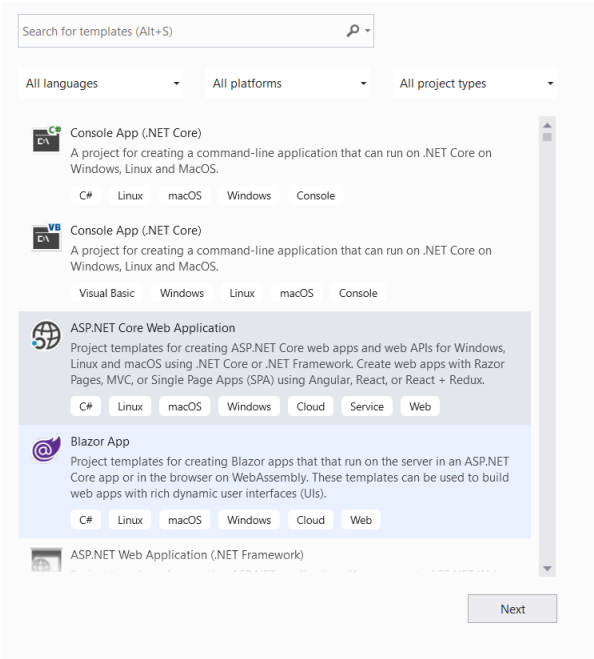
9.
10. Set view name “index” then click add

11. 

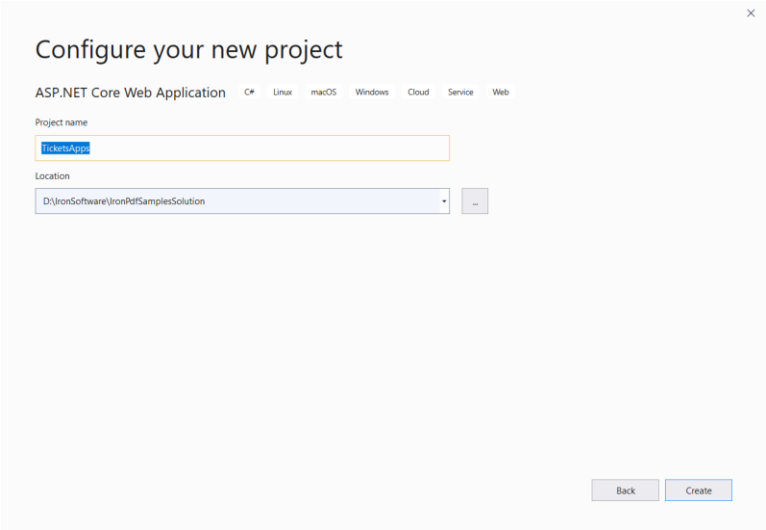
    11.1.

Sample: TicketsApps .NET Core MVC Application

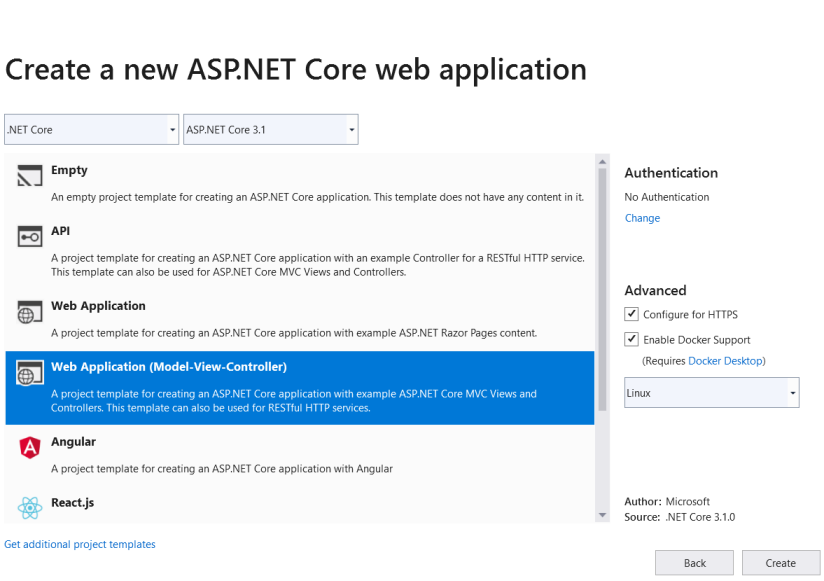
1- Choose ASP.NET core web applications



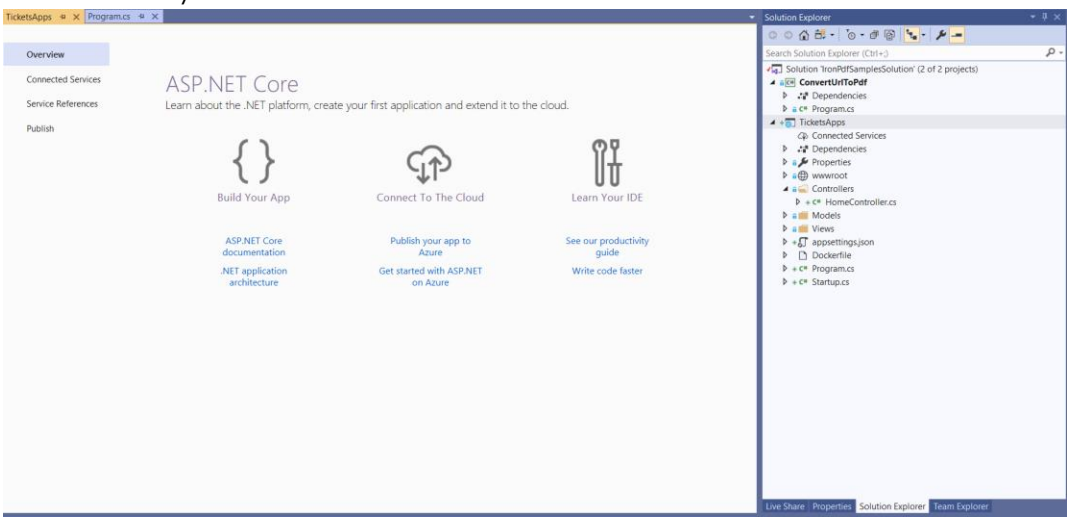
2- Name the project “TicketsApps”



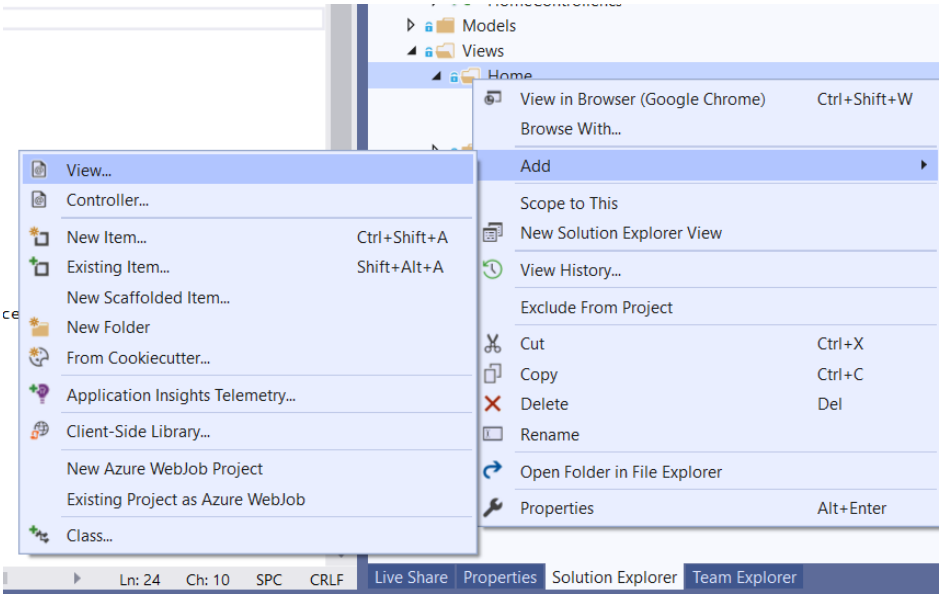
3- Choose “.NET Core”, “ASP.NET core 3.1” , “Web Application (Model-View-Controller)” , check enable docker and choose Linux Image



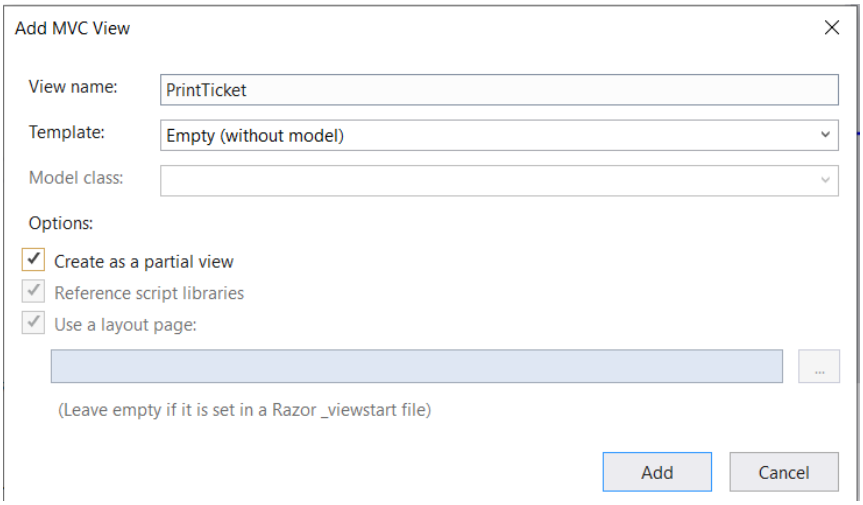
4- Now its ready



5- using mouse right click over folder views -> Home and select home



6- add partial view



Advanced options

Chapter 3: Deployment with dockers

Deploy to Linux docker

Deploy to windows docker

Chapter 4: Working with Pdf Document

Open Pdf
Merge Pdf
Add Header Or footer to Pdf
Pdf security
Pdf extraction and conversions

Summary

Appendix (A)

References

1- Dockize and asp.net core application
https://docs.docker.com/engine/examples/dotnetcore/
2- Introduction to .NET and Docker
https://docs.microsoft.com/en-us/dotnet/core/docker/introduction
3- Containerize a .NET Core app
https://docs.microsoft.com/en-us/dotnet/core/docker/build-container

Author

Ahmed Aboelmagd is a Full-stack experienced and certified Microsoft technology specialist with 12+ year's experience in IT and Software development, delivered 15+ successful project in many size scales from small to an enterprise for industries like (tourism, educational, manufacturing, etc.)