

IronPdf Guide with .NET Core

Version 2019.8.9

Contents

Introduction	2
Document Organization.....	2
Chapter 1: Install IronPdf	2
Install using NuGet	2
Using NuGet Package Manager	3
Using NuGet Package Console manager.....	5
Sample 1: HelloWorldConsole Console Application	6
Sample 2: HelloWorldCore .Net Core Web Application	6
Chapter 2: Convert to Pdf.....	6
Convert online website to Pdf	6
Sample: ConvertUrlToPdf console application	6
Convert HTML to Pdf.....	11
Sample: ConvertHTMLToPdf Console application	11
Convert MVC Partial view to Pdf.....	12
Sample: TicketsApps .NET Core MVC Application	20
Advanced options.....	23
Chapter 3: Deployment with dockers	23
Chapter 4: Working with Pdf Document	23
Open Pdf.....	23
Merge Pdf	23
Add Header Or footer to Pdf	23
Pdf security.....	23
Pdf extraction and conversions	23
Summary	23
Appendix (A)	23
References	29
Author.....	30

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.

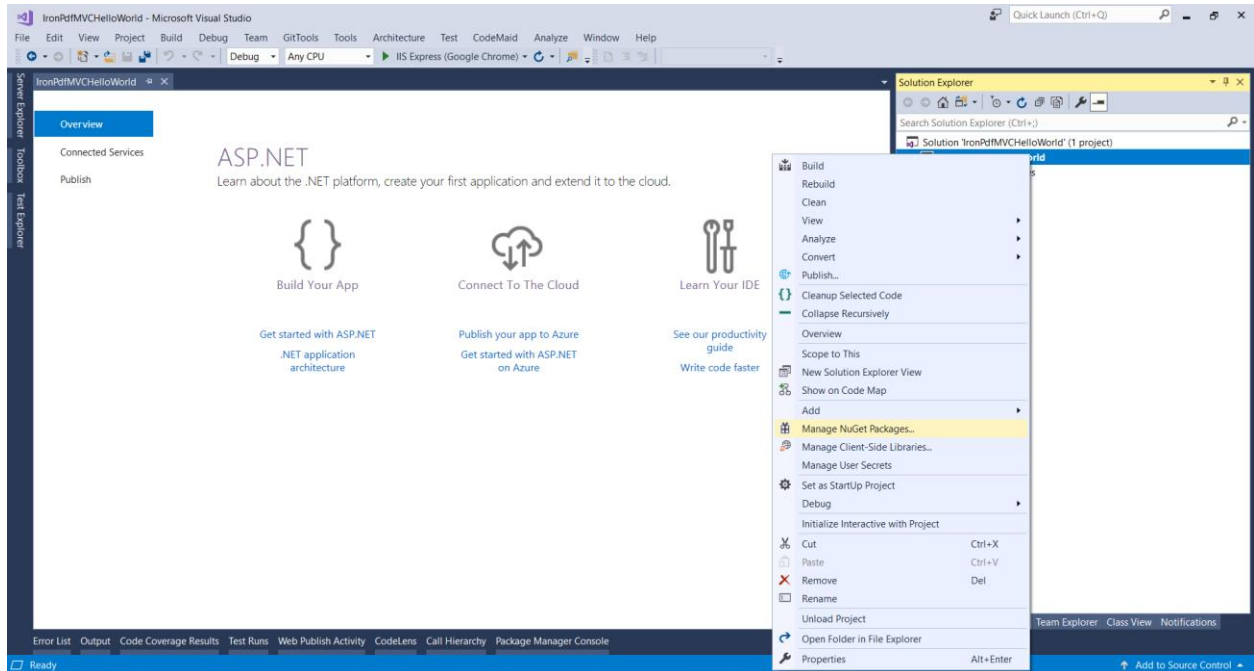
To add IronPdf library to the project we have two ways, from Visual studio editor install using NuGet or command line using package console manager as following: -

Install using NuGet

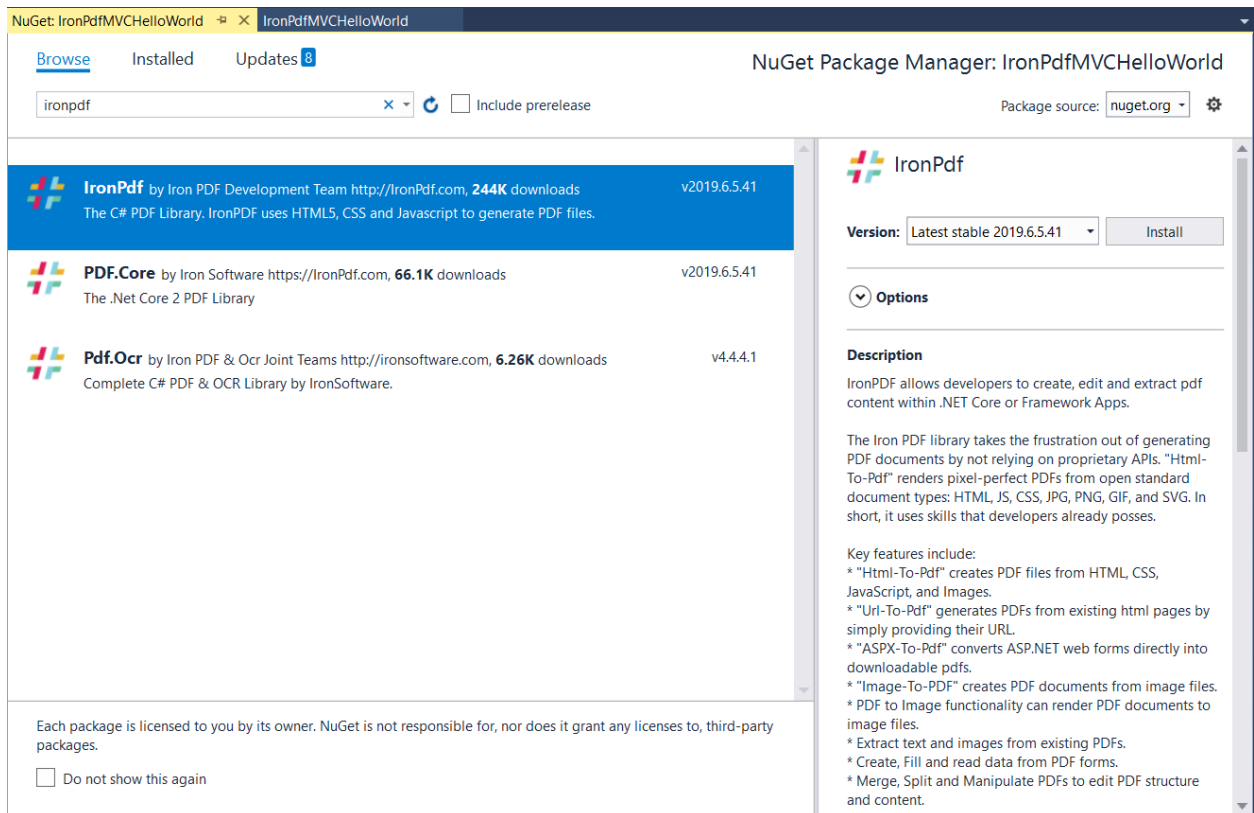
To add IronPdf library to our project using NuGet we can do it using visualized interface (NuGet Package Manager) or by command using Package Manager Console as following: -

Using NuGet Package Manager

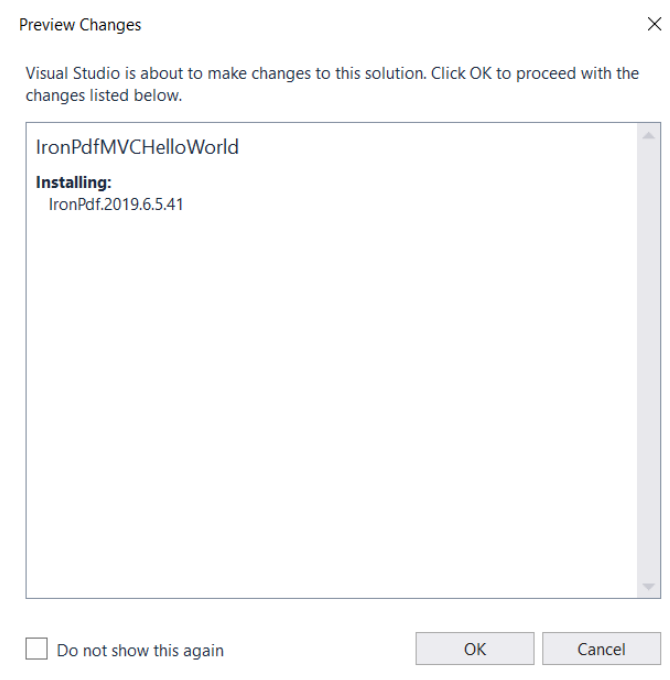
1- Using mouse -> right click on project name -> Select manage NuGet Package



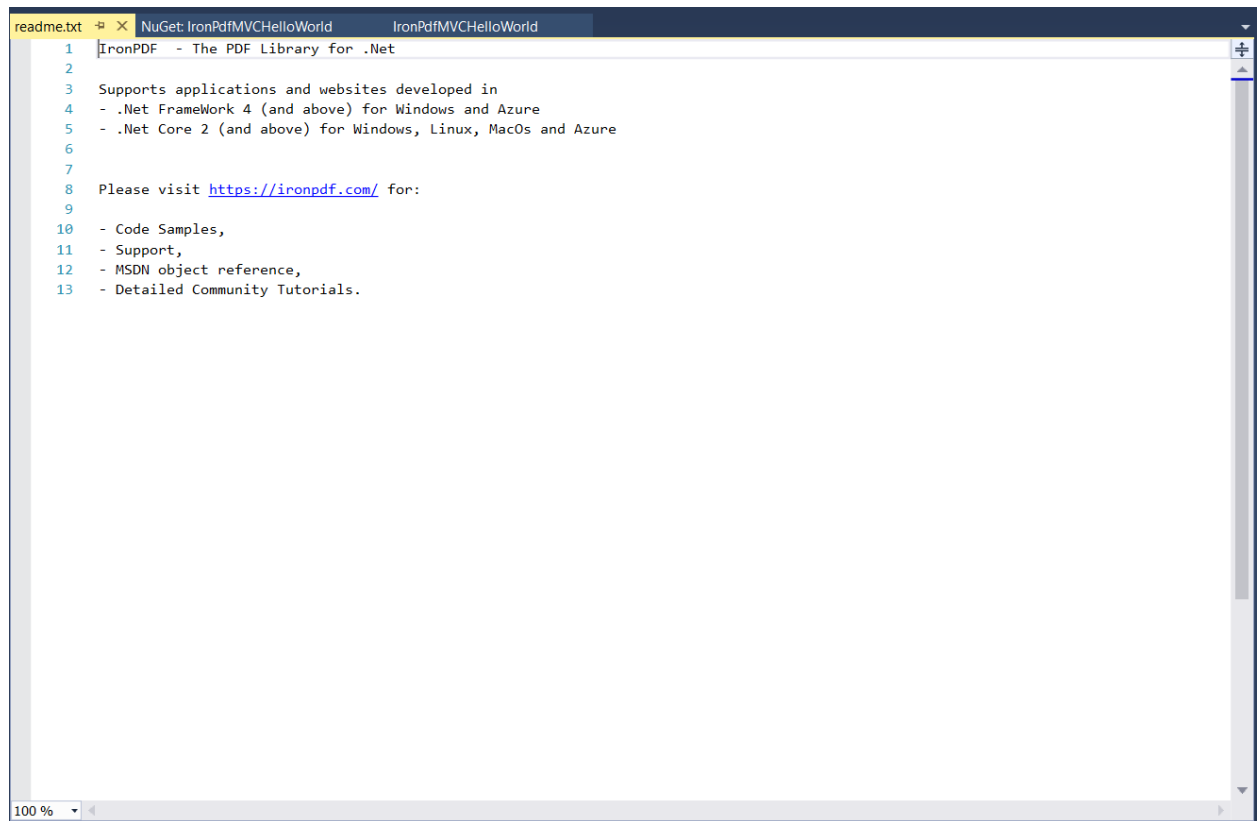
2- From brows tab -> search for IronPdf -> Install



3- Click Ok

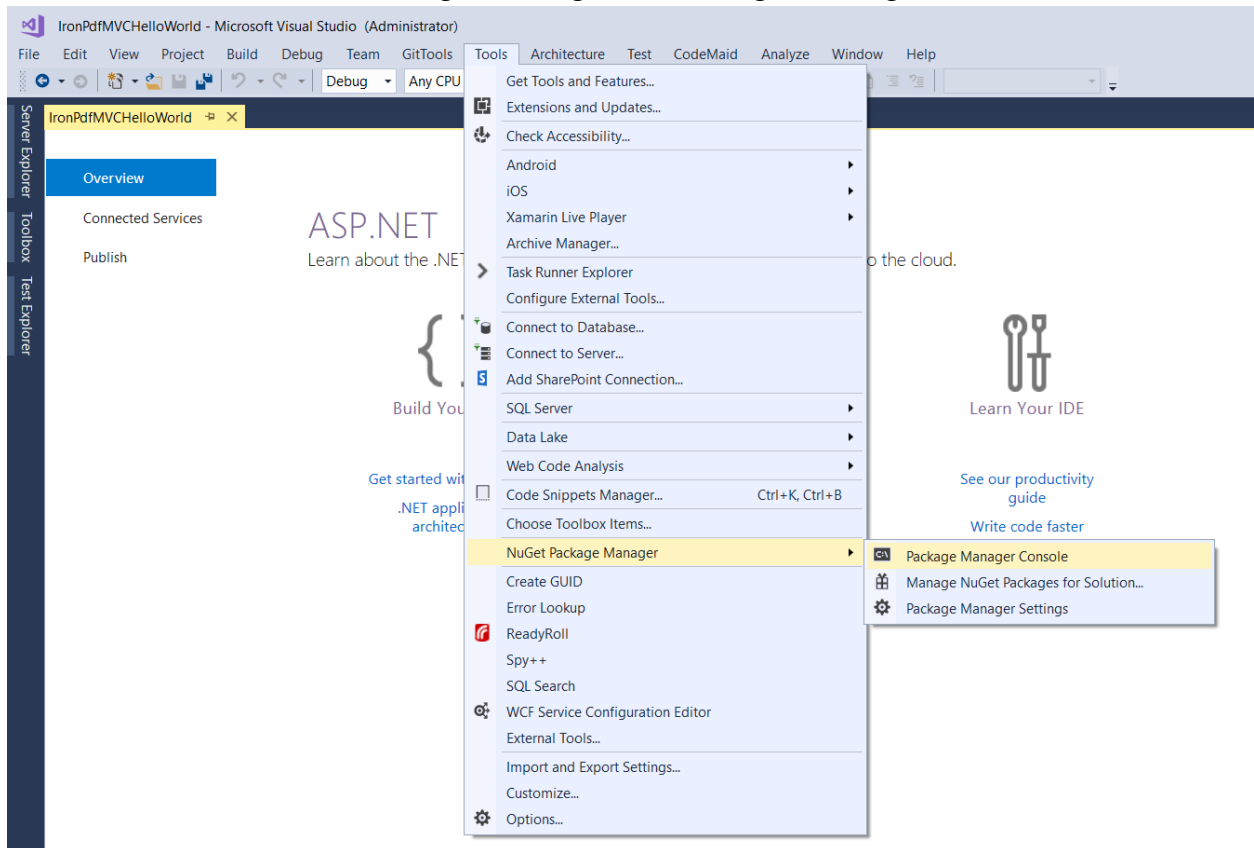


4- And we are Done

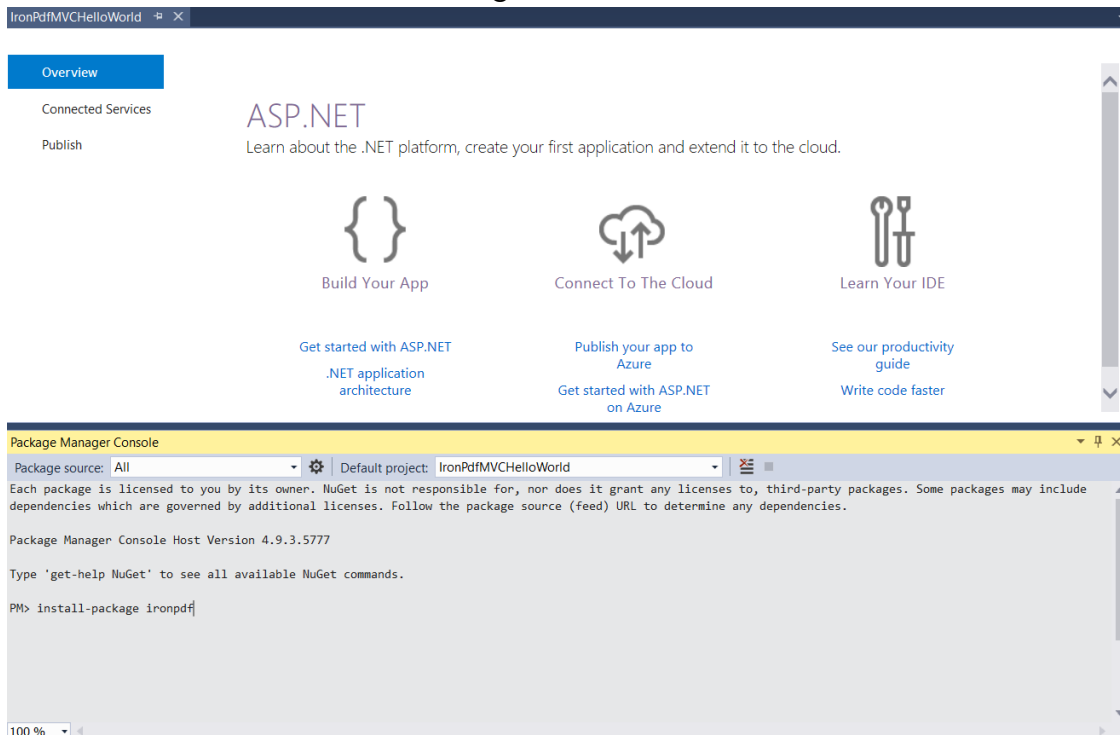


Using NuGet Package Console manager

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

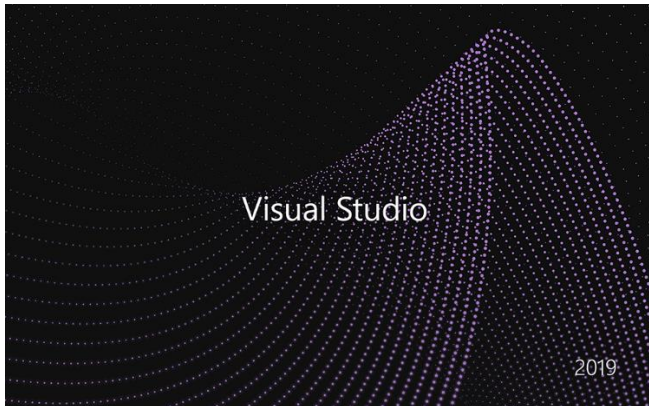
Chapter 2: Convert to Pdf

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

Get started



Clone or check out code

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

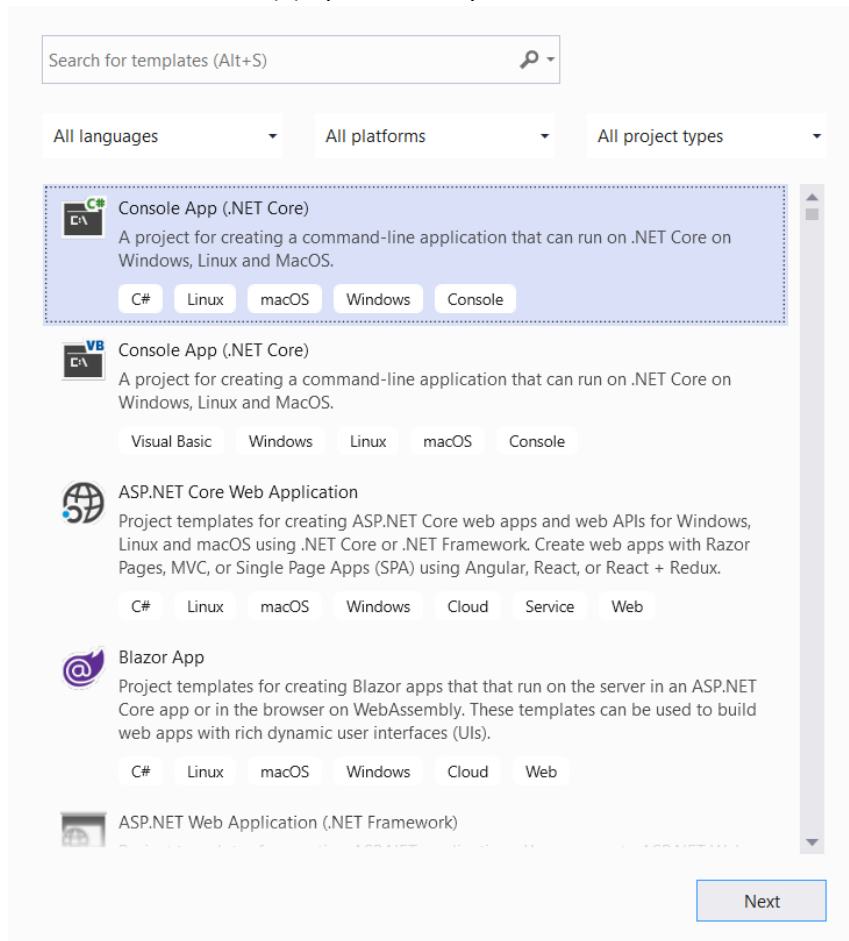


Create a new project

Choose a project template with code scaffolding to get started

[Continue without code →](#)

3- Choose Console App (.NET Core)



- 4- Give our sample name "ConvertUrlToPdf" and click create

×

Configure your new project

Console App (.NET Core) C# Linux macOS Windows Console

Project name

ConvertUrlToPdf

Location

D:\IronSoftware\IronPdfSamplesSolution

...

Back Create

- 5- Now we have console application created

The screenshot shows the Visual Studio IDE with the 'ConvertUrlToPdf' project open. The main editor displays the 'Program.cs' file with the following code:

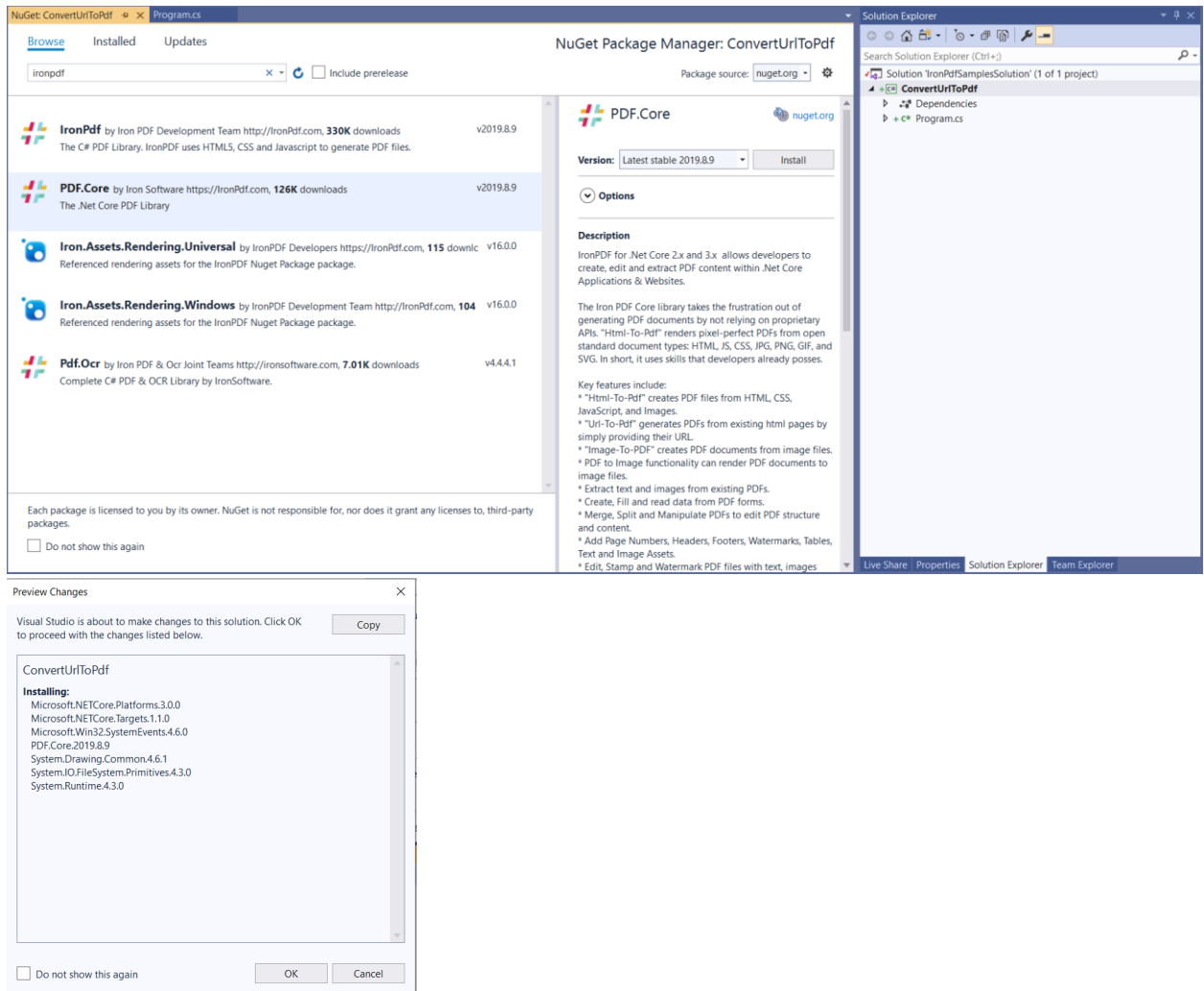
```
1 using System;
2
3 namespace ConvertUrlToPdf
4 {
5     // 0 references | 0 changes | 0 authors, 0 changes
6     class Program
7     {
8         // 0 references | 0 changes | 0 authors, 0 changes
9         static void Main(string[] args)
10         {
11             Console.WriteLine("Hello World!");
12         }
13     }
14 }
```

The Solution Explorer on the right shows the project structure:

- Solution 'IronPdfSamplesSolution' (1 of 1 project)
 - ConvertUrlToPdf
 - Dependencies
 - Program.cs

The status bar at the bottom indicates '100 %', 'No issues found', and 'Ln: 1 Cn: 1 SPC CRLF'.

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

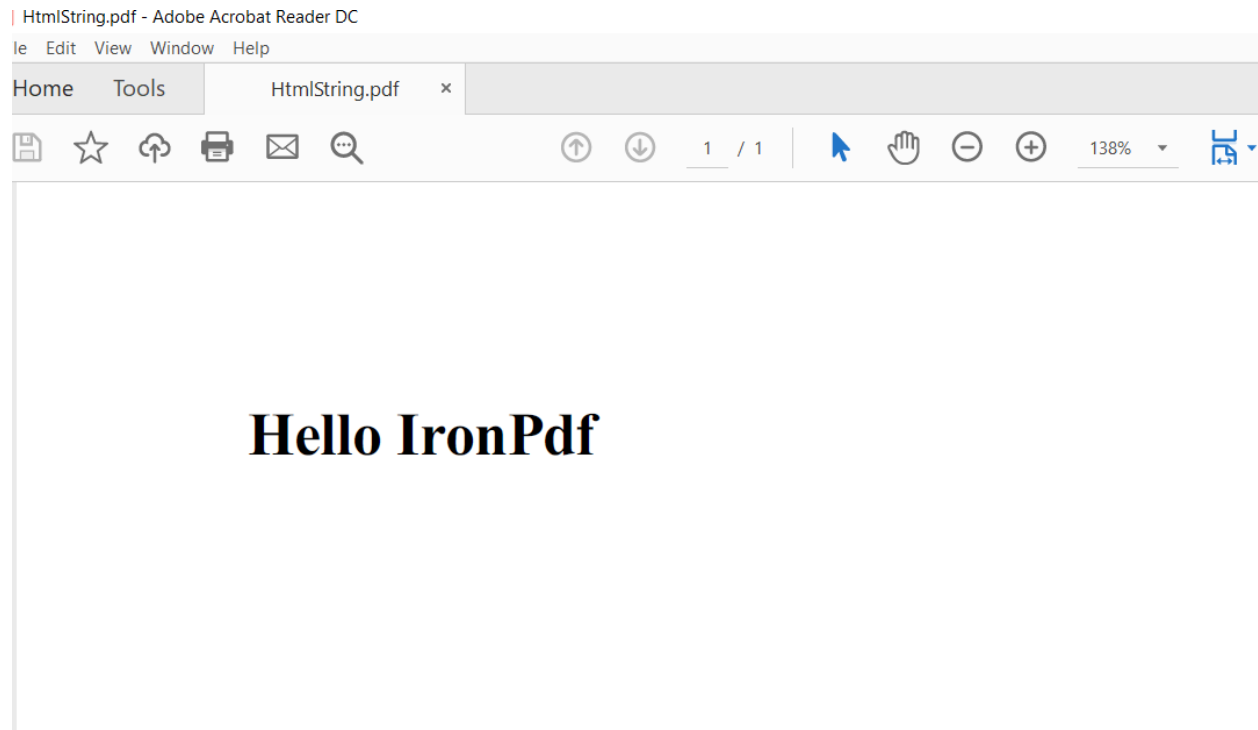
To render HTML to Pdf we have 2 way

- 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

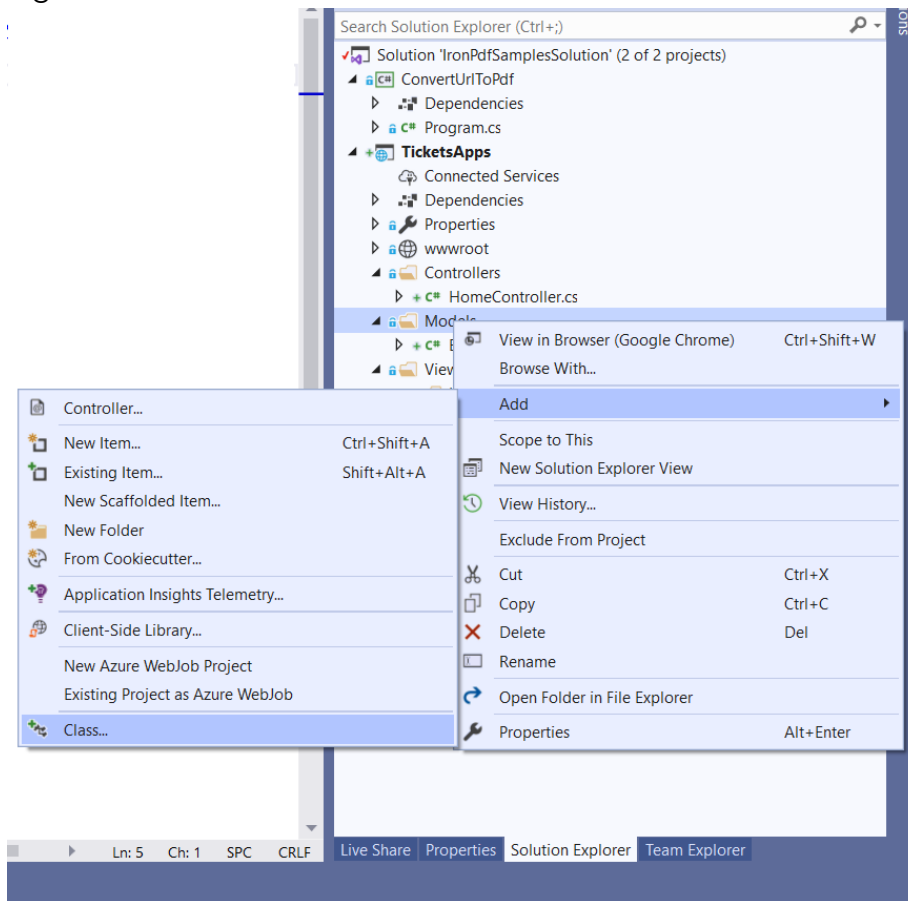
Let's implement real life example I'll choose online ticketing site you open the site and navigate to book ticket then fill required information's then you get your copy as downloadable pdf file so let's go.

We will go through this step: -

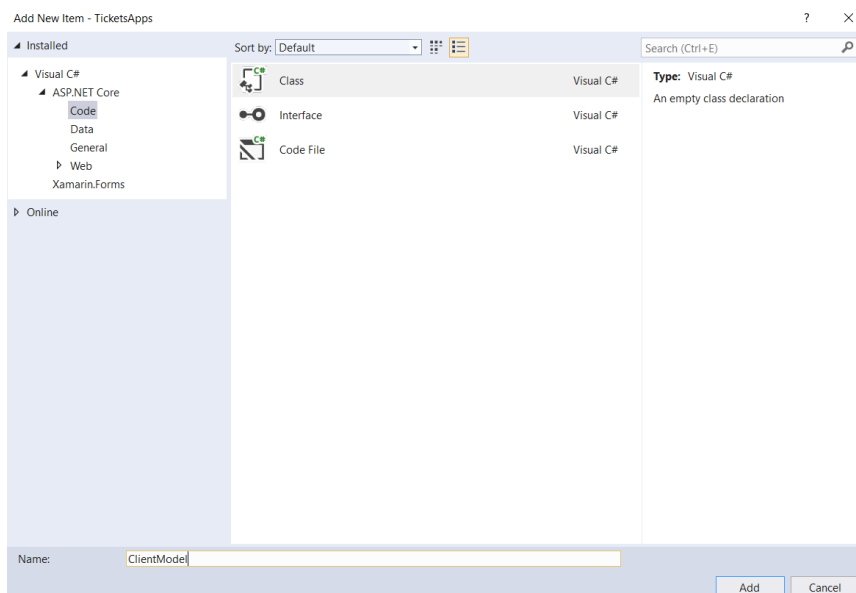
- 1- Create client object model
- 2- Create client services (add, view)
- 3- Add pages (Register, view)
- 4- Download pdf ticket

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.html)

```

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

```

3.4. result should be looks like this

Index

Name

Phone

Email

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

Index

Name
The Name field is required.

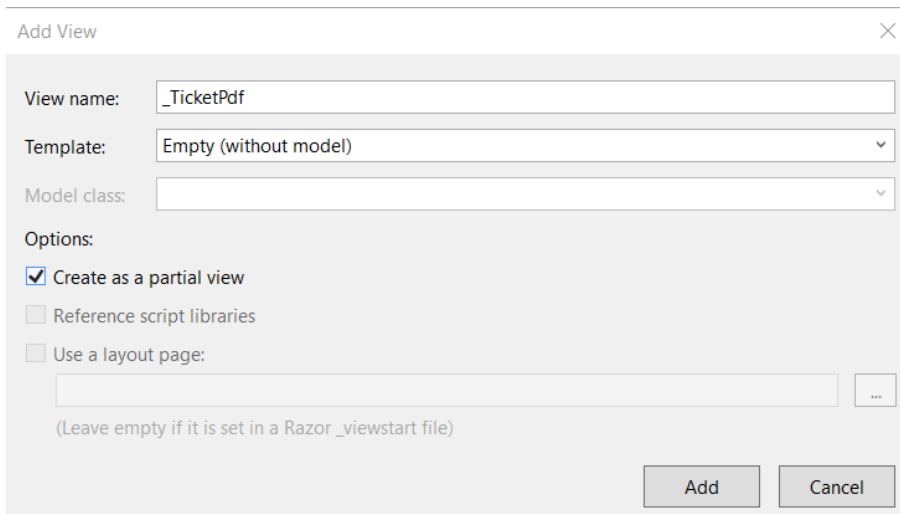
Phone
The Phone field is required.

Email
The Email field is required.

3.8. add TicketView to display our ticket

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

3.9. Add its view



Add View

View name:

Template:

Model class:

Options:

☒ Create as a partial view

☐ Reference script libraries

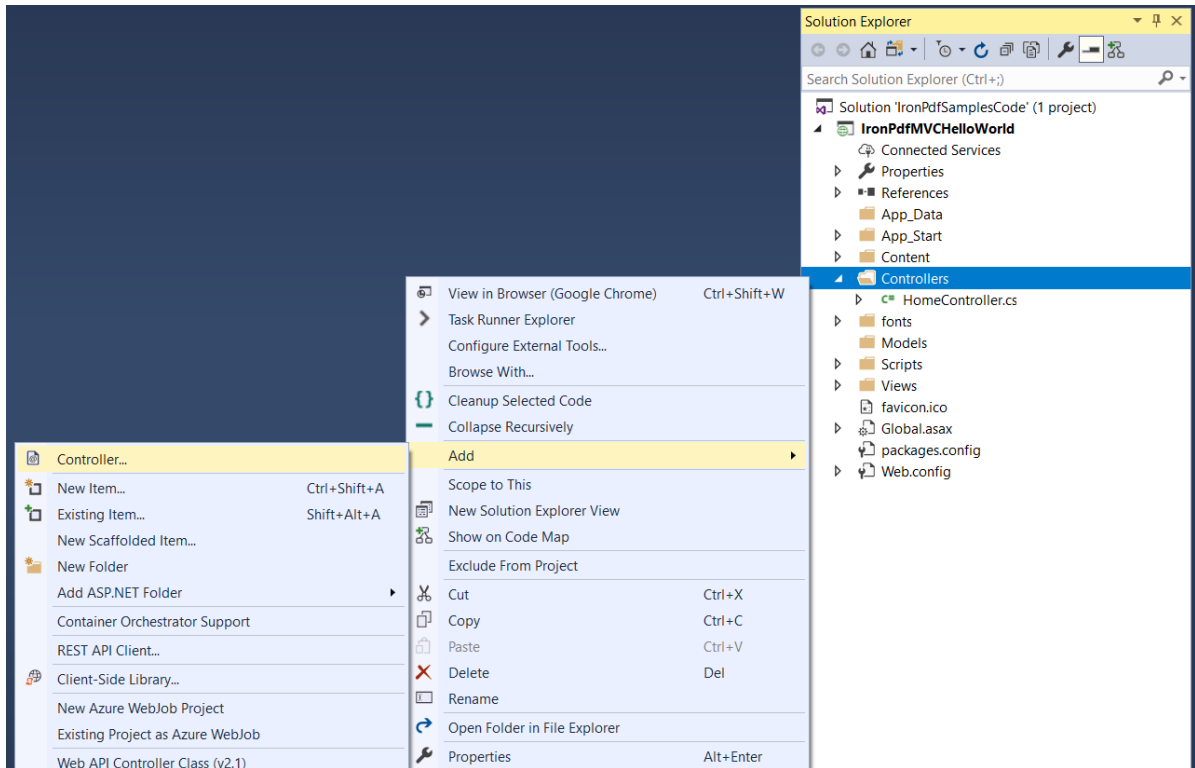
☐ Use a layout page:

...

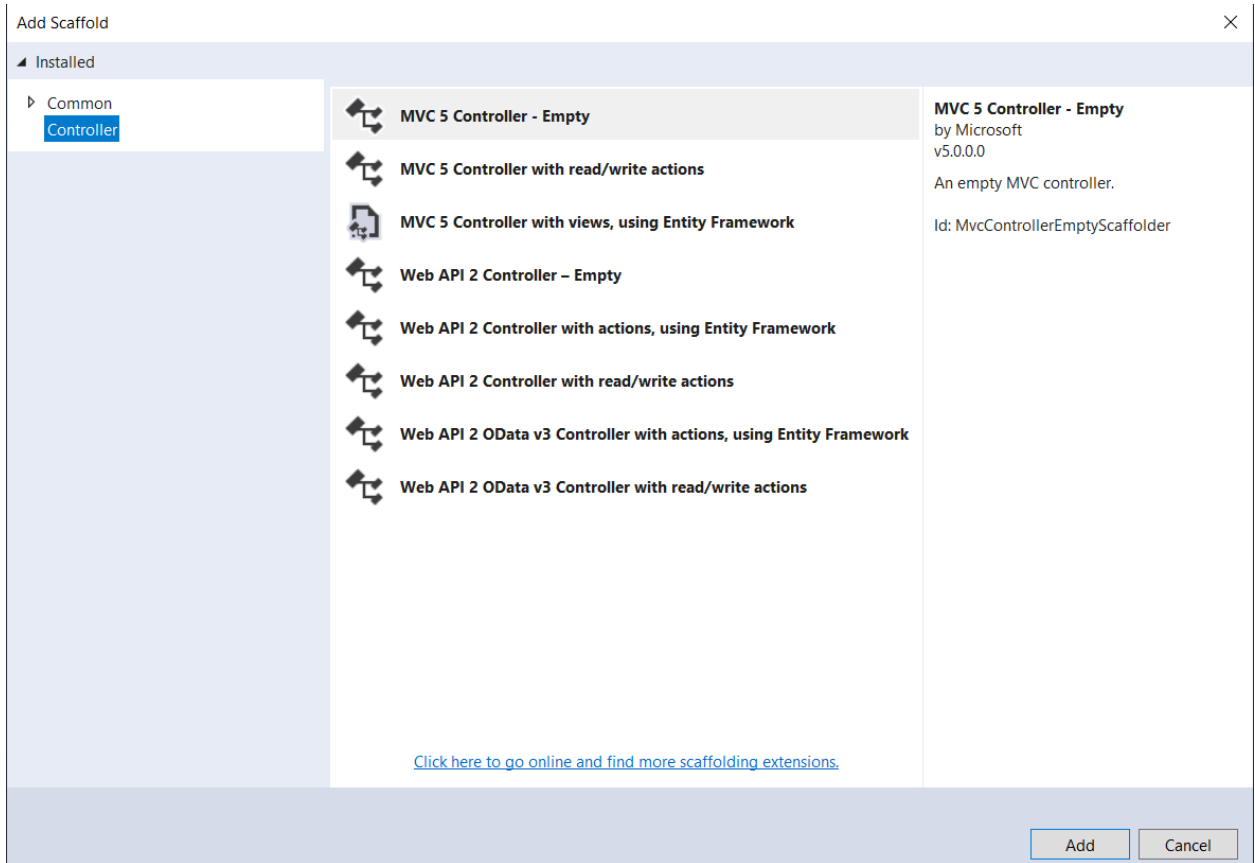
(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

3.10.

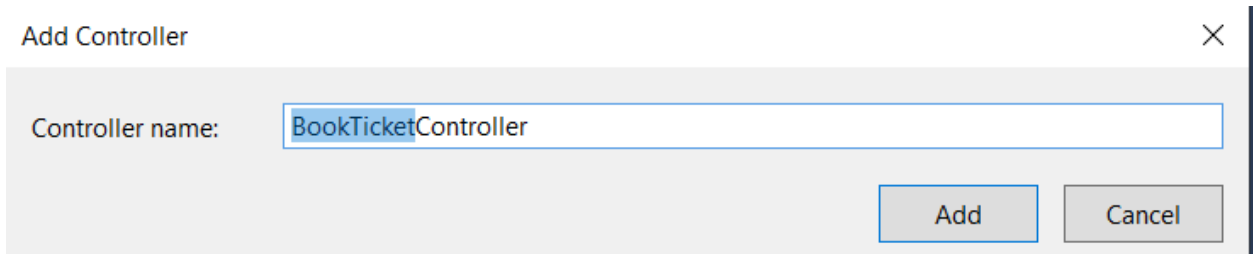


- 4.
5. From opened window select MVC 5 Controller – Empty

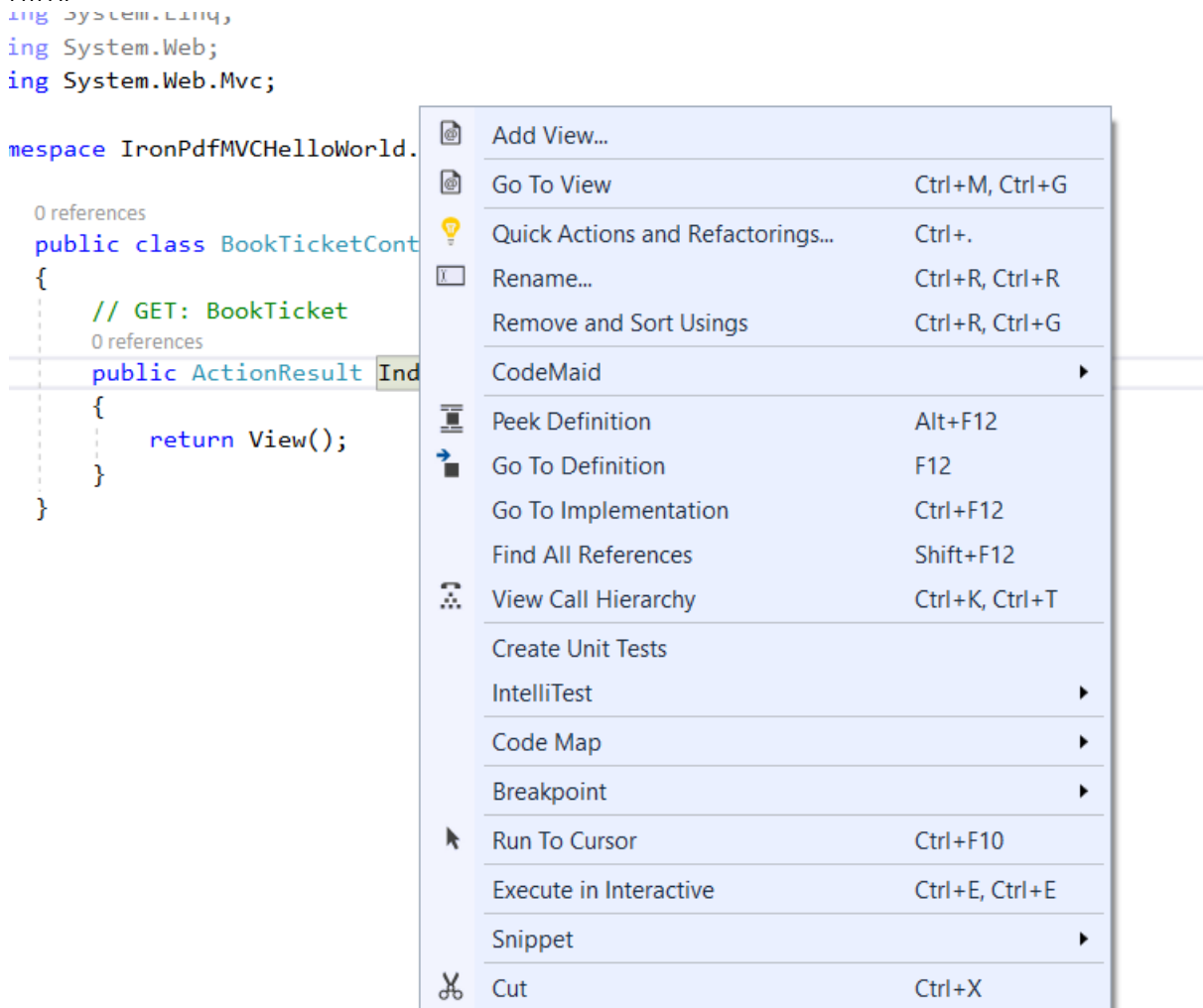


- 6.

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

Add View

View name:

Index

Template:

Empty (without model)

Model class:

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

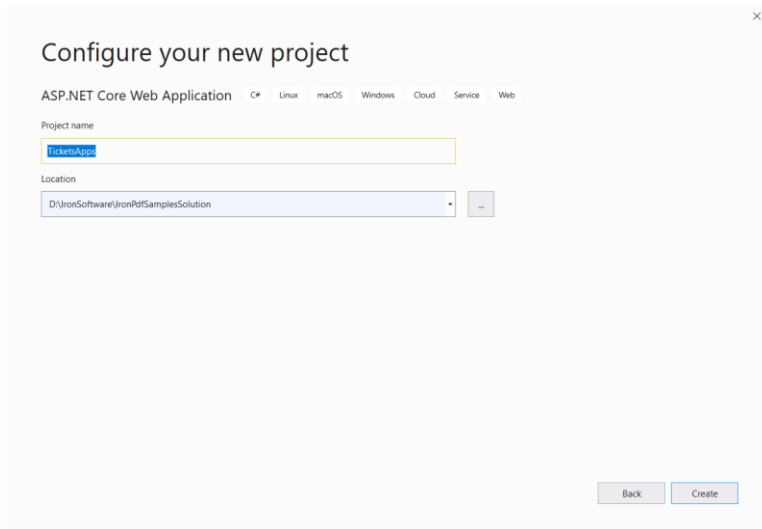
11.1.

Sample: TicketsApps .NET Core MVC Application

1- Choose ASP.NET core web applications

The screenshot shows the 'New Project' dialog in Visual Studio. At the top is a search bar containing 'Search for templates (Alt+S)' and a magnifying glass icon. Below the search bar are three filter tabs: 'All languages', 'All platforms', and 'All project types'. The main area displays two project templates. The first is 'Console App (.NET Core)', represented by a terminal icon, with a description: 'A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.' Below the description are five buttons: 'C#', 'Linux', 'macOS', 'Windows', and 'Console'. The second template is 'Console App (.NET Core)' again, but with a Visual Basic icon, and a description: 'A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.' Below this are five buttons: 'Visual Basic', 'Windows', 'Linux', 'macOS', and 'Console'. Below these two templates is a section for 'ASP.NET Core Web Application', represented by a globe icon. Its description is: 'Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.' Below this description are six buttons: 'C#', 'Linux', 'macOS', 'Windows', 'Cloud', 'Service', and 'Web'. Below that is a section for 'Blazor App', represented by a '@' icon. Its description is: 'Project templates for creating Blazor apps that that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).' Below this description are five buttons: 'C#', 'Linux', 'macOS', 'Windows', 'Cloud', and 'Web'. At the bottom of the dialog is an 'ASP.NET Web Application (.NET Framework)' template, partially visible. At the very bottom right is a 'Next' button.

2- Name the project "TicketsApps"



Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

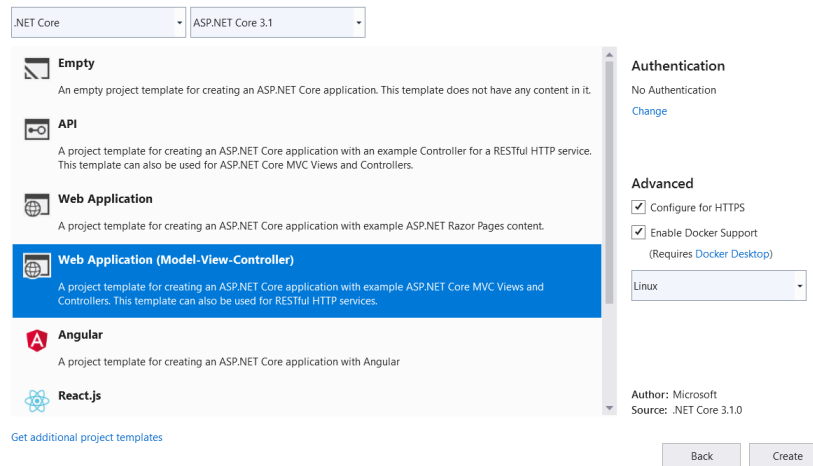
Project name
TicketsApps

Location
D:\IronSoftware\IronPdfSamplesSolution

Back Create

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

Create a new ASP.NET Core web application



.NET Core ASP.NET Core 3.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

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

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (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.

Angular
A project template for creating an ASP.NET Core application with Angular

React.js

[Get additional project templates](#)

Authentication
No Authentication
[Change](#)

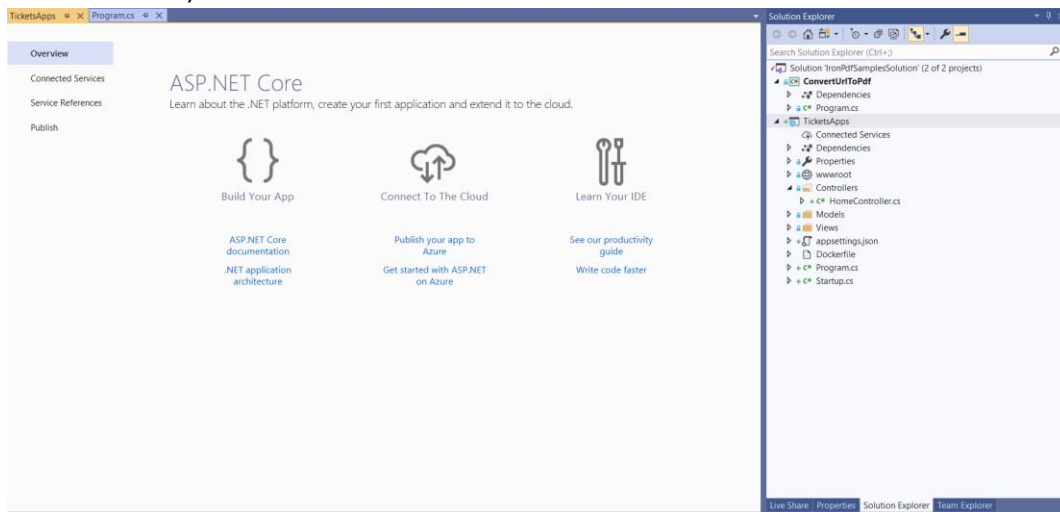
Advanced
☒ Configure for HTTPS
☒ Enable Docker Support
(Requires [Docker Desktop](#))

Linux

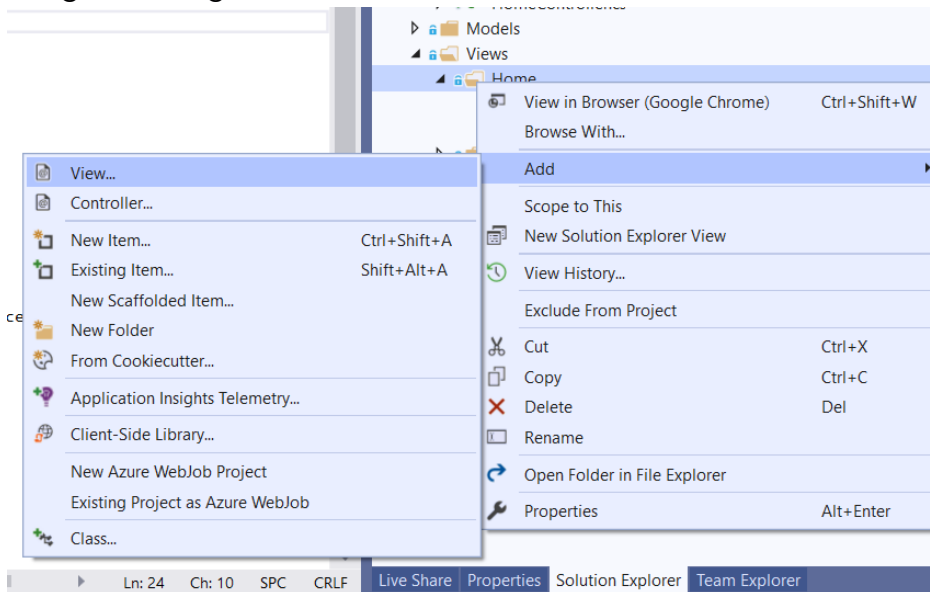
Author: Microsoft
Source: .NET Core 3.1.0

Back Create

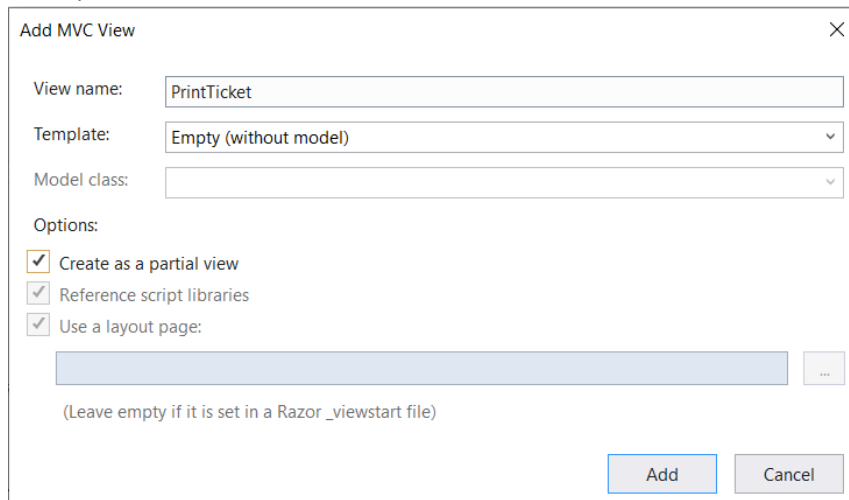
4- Now its ready



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



6- add partial view



The image shows a 'Add MVC View' dialog box. It has a title bar with a close button. Inside, there are three input fields: 'View name:' with the text 'PrintTicket', 'Template:' with a dropdown menu showing 'Empty (without model)', and 'Model class:' with an empty dropdown. Below these is an 'Options:' section with three checked checkboxes: 'Create as a partial view', 'Reference script libraries', and 'Use a layout page:'. The 'Use a layout page:' checkbox has a text box next to it. At the bottom right are 'Add' and 'Cancel' buttons. A note at the bottom says '(Leave empty if it is set in a Razor _viewstart file)'.

Add MVC View

View name:

Template:

Model class:

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

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

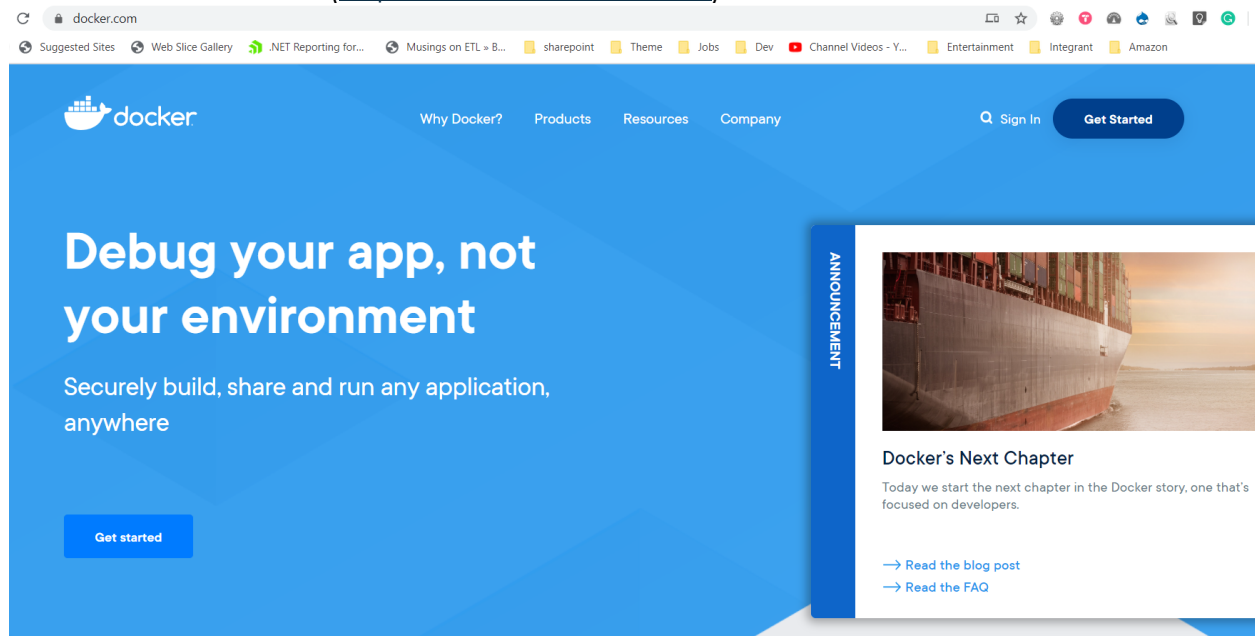
Summary

Appendix (A)

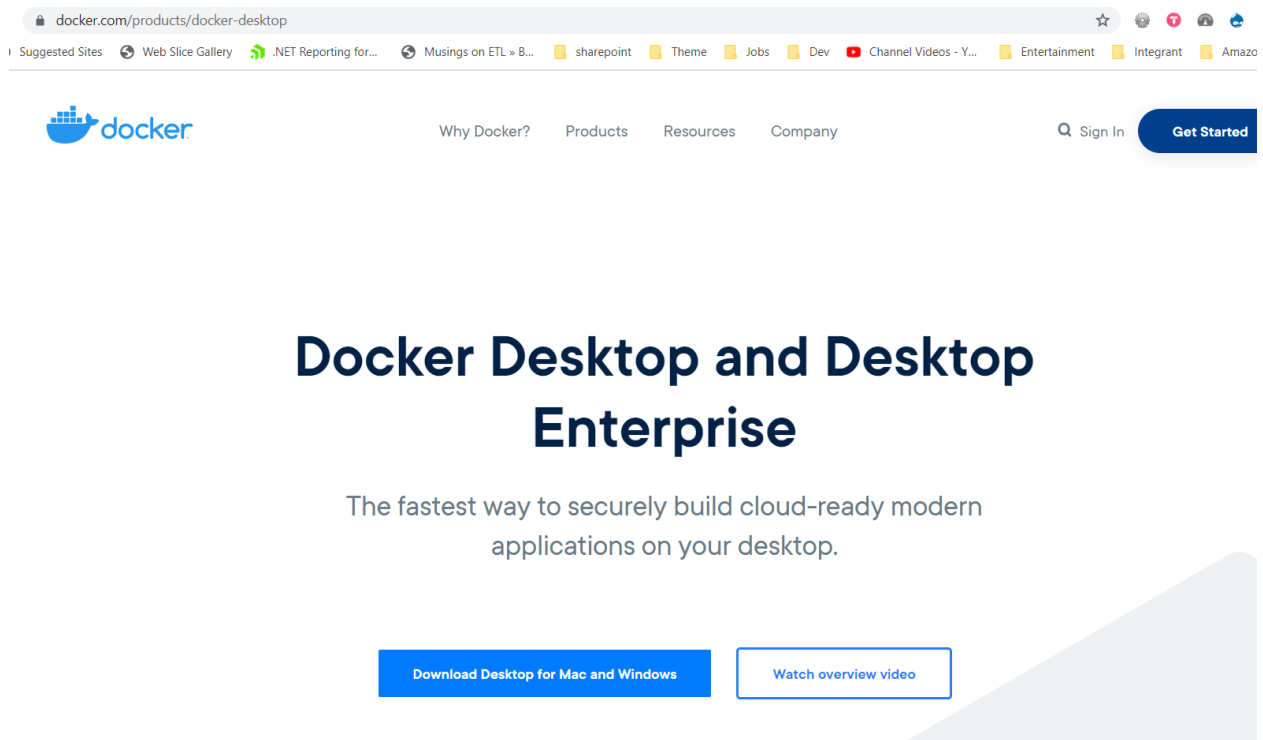
What is Docker

Install Docker

Go to Docker website (<https://www.docker.com/>)



Click Get started



Click download for mac and windows




Create a Docker ID.

Already have an account? [Sign In](#)



☐ Send me occasional product updates and announcements.

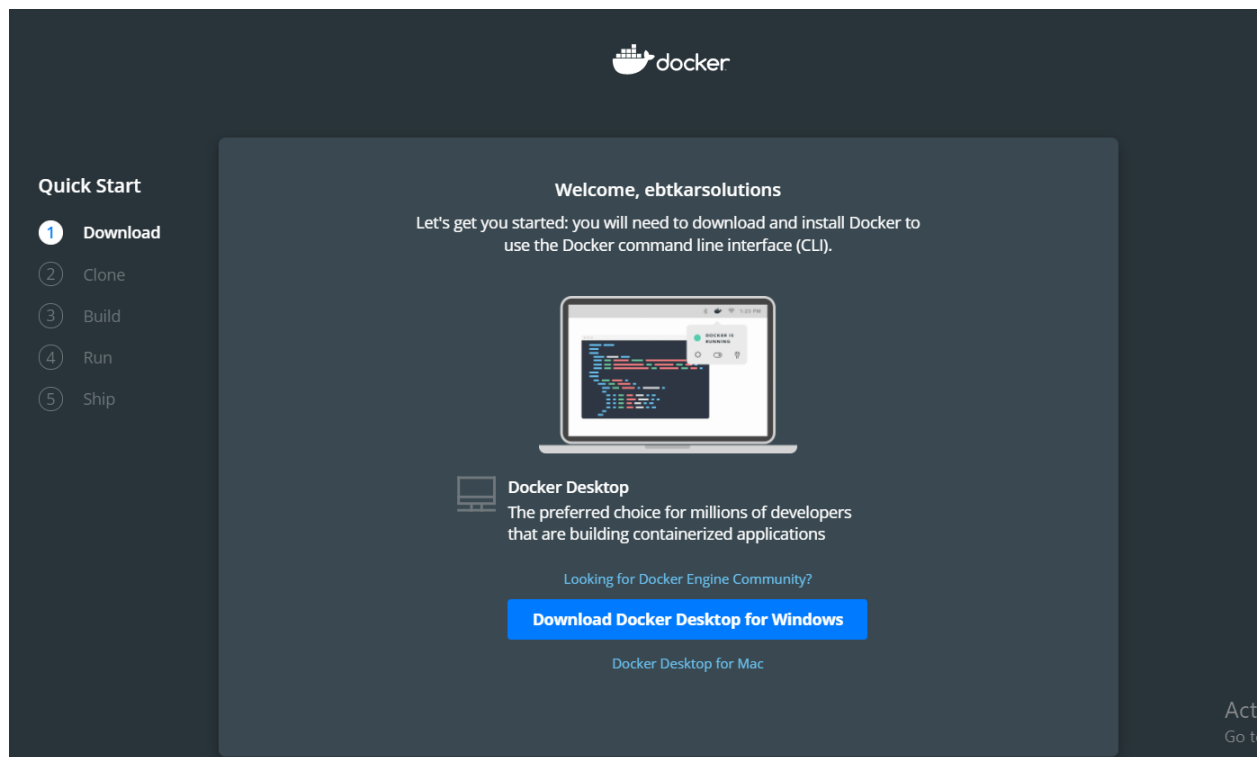
☐ I'm not a robot


reCAPTCHA
[Privacy - Terms](#)

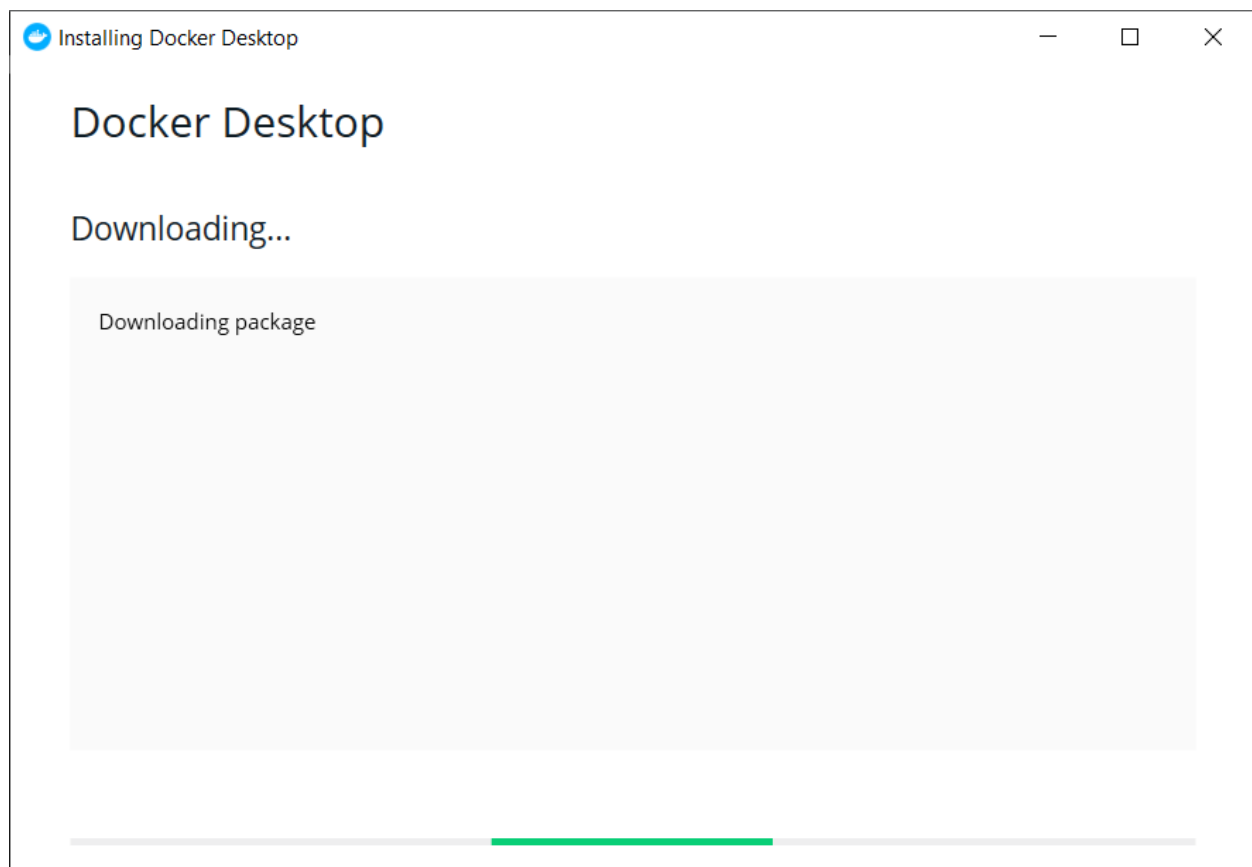
Sign Up

By creating an account, you agree to the [Terms of Service](#),
[Privacy Policy](#), and [Data Processing Terms](#).

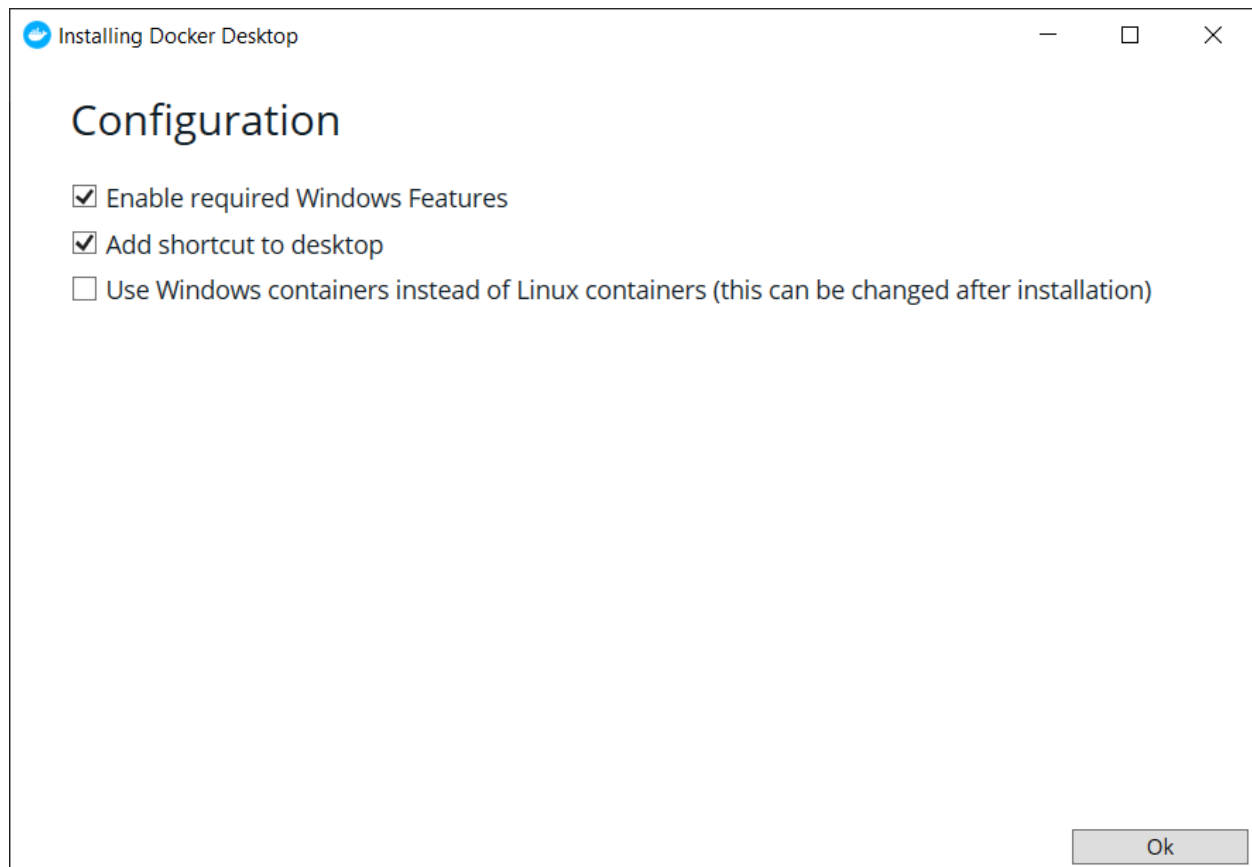
Signup for free then login



Download Docker for windows



Start installing Docker.

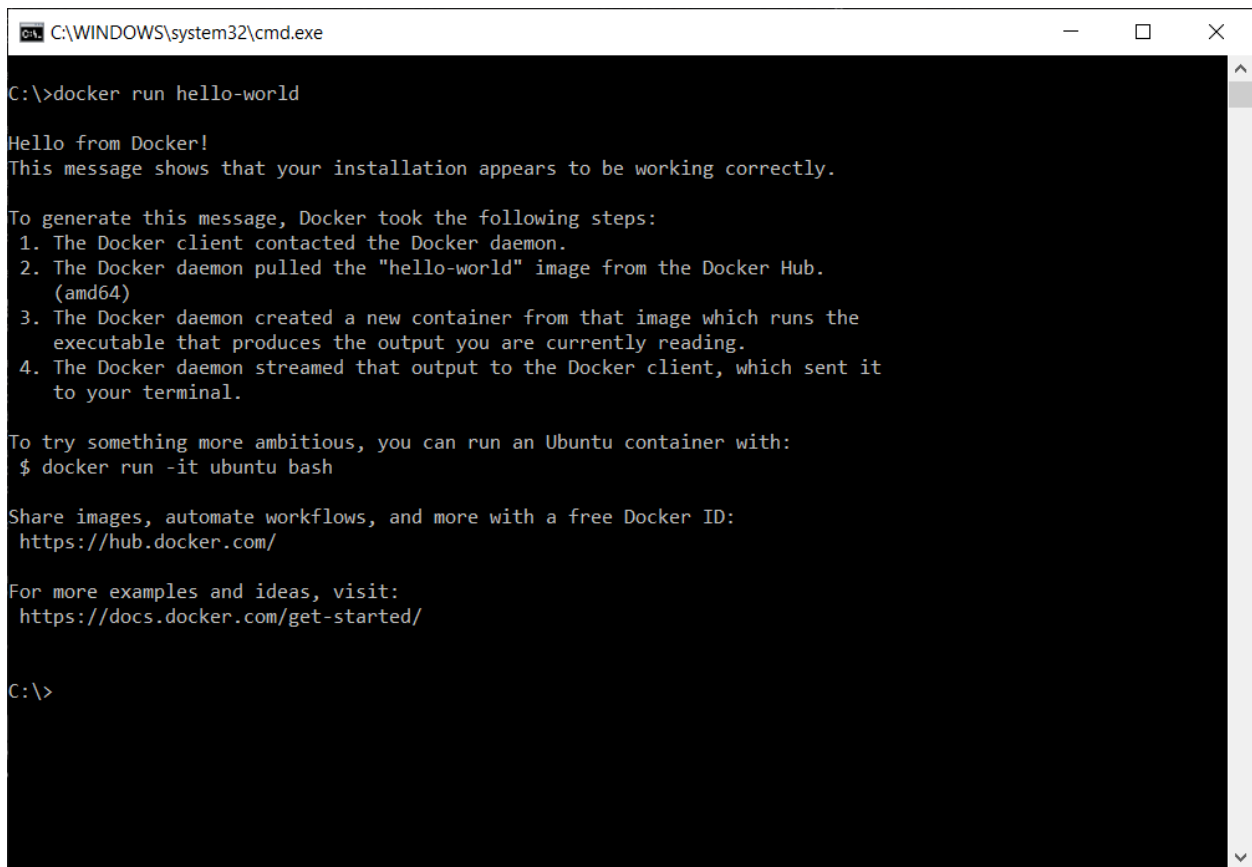


It will require restart and after restart machine login to Docker



Now you can run Docker hello world by opening windows command line or powershell script and write

Docker run hello-world

A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the command 'C:\>docker run hello-world' and its output. The output includes a 'Hello from Docker!' message, a confirmation that the installation is working, a list of four steps Docker took to generate the message, and instructions on how to run an Ubuntu container and where to find more information.

```
C:\>docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

C:\>
```

Now this is list of most important command line that help you

- docker images => To list all available images on this machine
- docker ps => to list all running containers
- docker ps -a => to list all containers

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>
- 4- Docker
<https://www.docker.com/>

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.)