|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| IronPdf Guide  with .NET Core  Version 2019.8.9 |
| |  |  |  | | --- | --- | --- | | IronSoftware |  | Ahmed Aboelmagd | |

Contents

[Introduction 2](#_Toc30939050)

[Document Organization 2](#_Toc30939051)

[Chapter 1: Install IronPdf 2](#_Toc30939052)

[Install using NuGet 2](#_Toc30939053)

[Using NuGet Package Manager 3](#_Toc30939054)

[Using NuGet Package Console manager 5](#_Toc30939055)

[Sample 1: HelloWorldConsole Console Application 6](#_Toc30939056)

[Sample 2: HelloWorldCore .Net Core Web Application 6](#_Toc30939057)

[Chapter 2: Convert to Pdf 6](#_Toc30939058)

[Convert online website to Pdf 6](#_Toc30939059)

[Sample: ConvertUrlToPdf console application 6](#_Toc30939060)

[Convert HTML to Pdf 11](#_Toc30939061)

[Sample: ConvertHTMLToPdf Console application 11](#_Toc30939062)

[Convert MVC view to Pdf 12](#_Toc30939063)

[Sample: TicketsApps .NET Core MVC Application 12](#_Toc30939064)

[Advanced options 28](#_Toc30939065)

[Chapter 3: Work with Dockers 29](#_Toc30939066)

[Chapter 4: Working with Pdf Document 30](#_Toc30939067)

[Open Pdf 30](#_Toc30939068)

[Merge Pdf 30](#_Toc30939069)

[Add Header Or footer to Pdf 30](#_Toc30939070)

[Pdf security 30](#_Toc30939071)

[Pdf extraction and conversions 30](#_Toc30939072)

[Summary 30](#_Toc30939073)

[Appendix (A) 30](#_Toc30939074)

[References 36](#_Toc30939075)

[Author 37](#_Toc30939076)

# 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

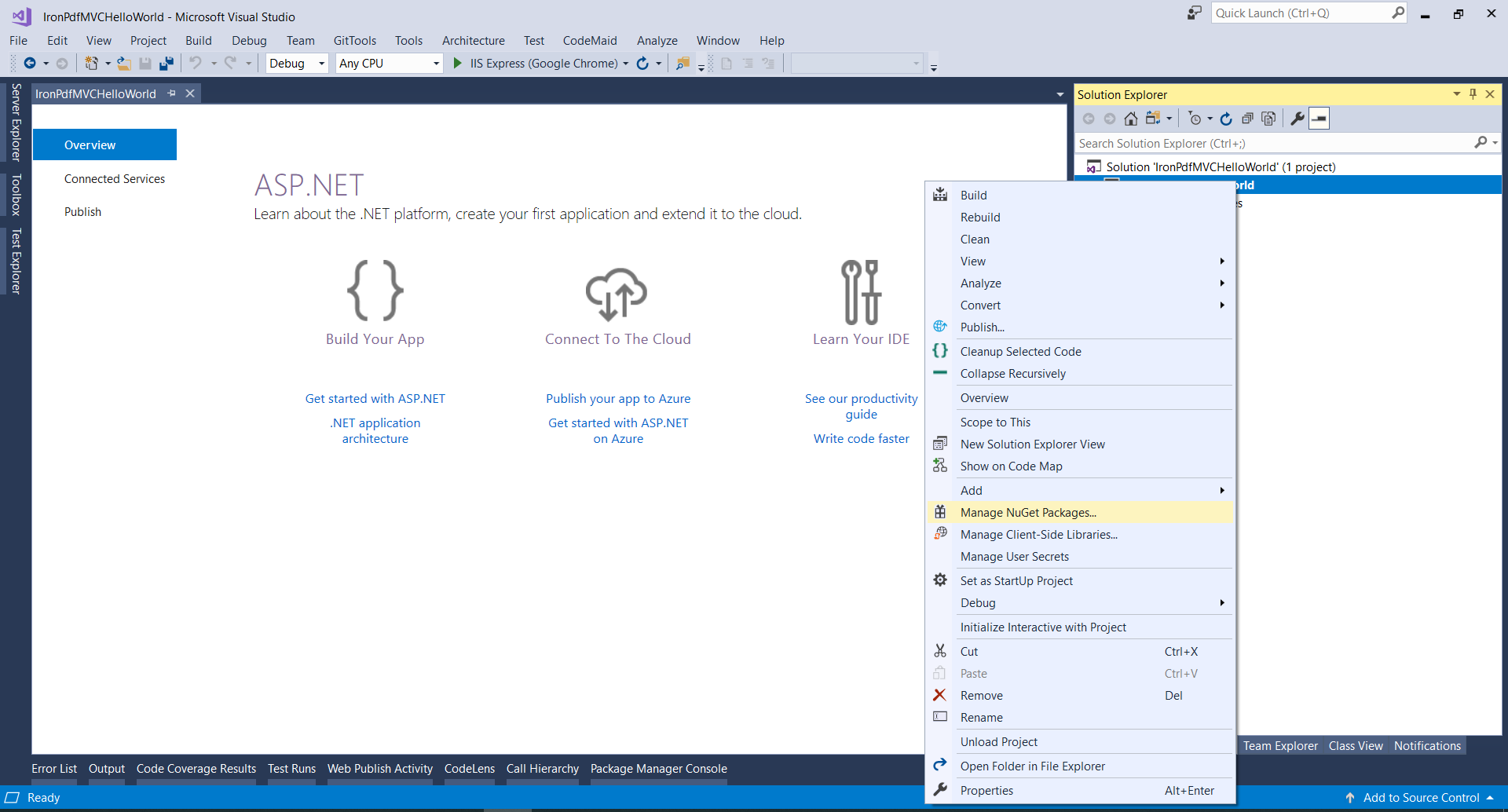
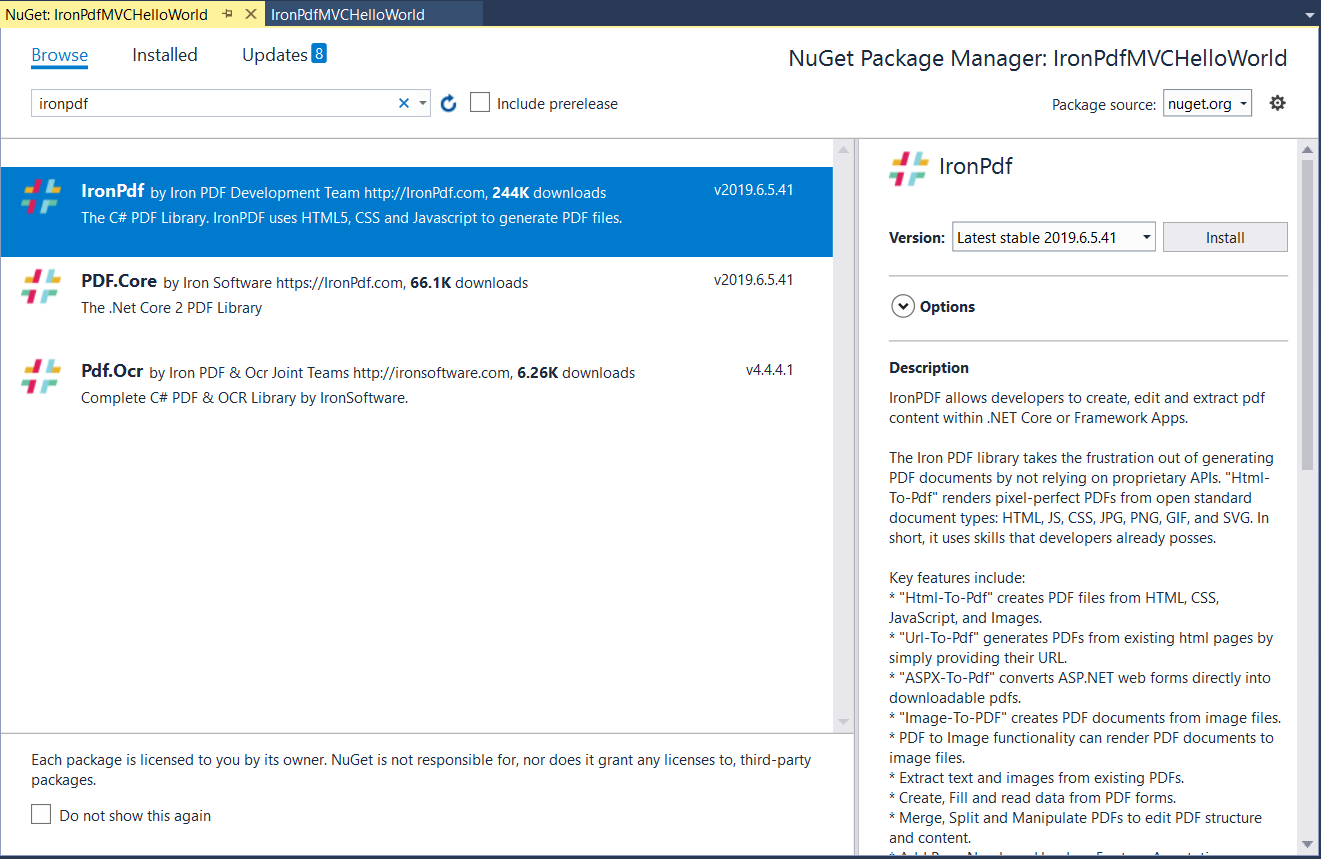
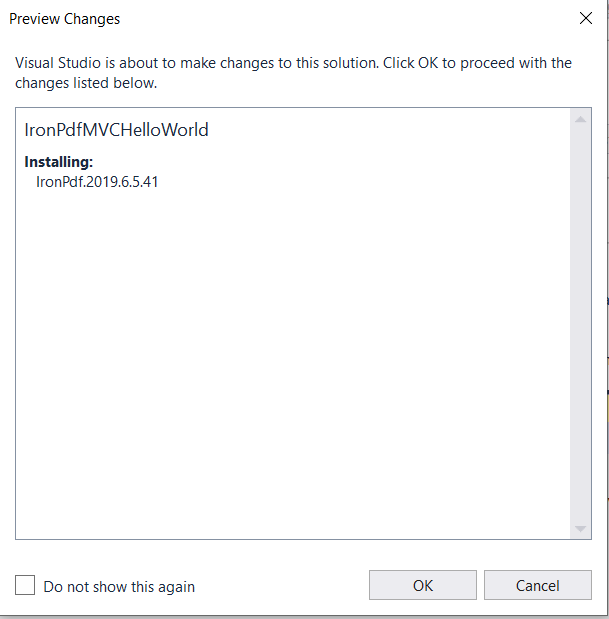
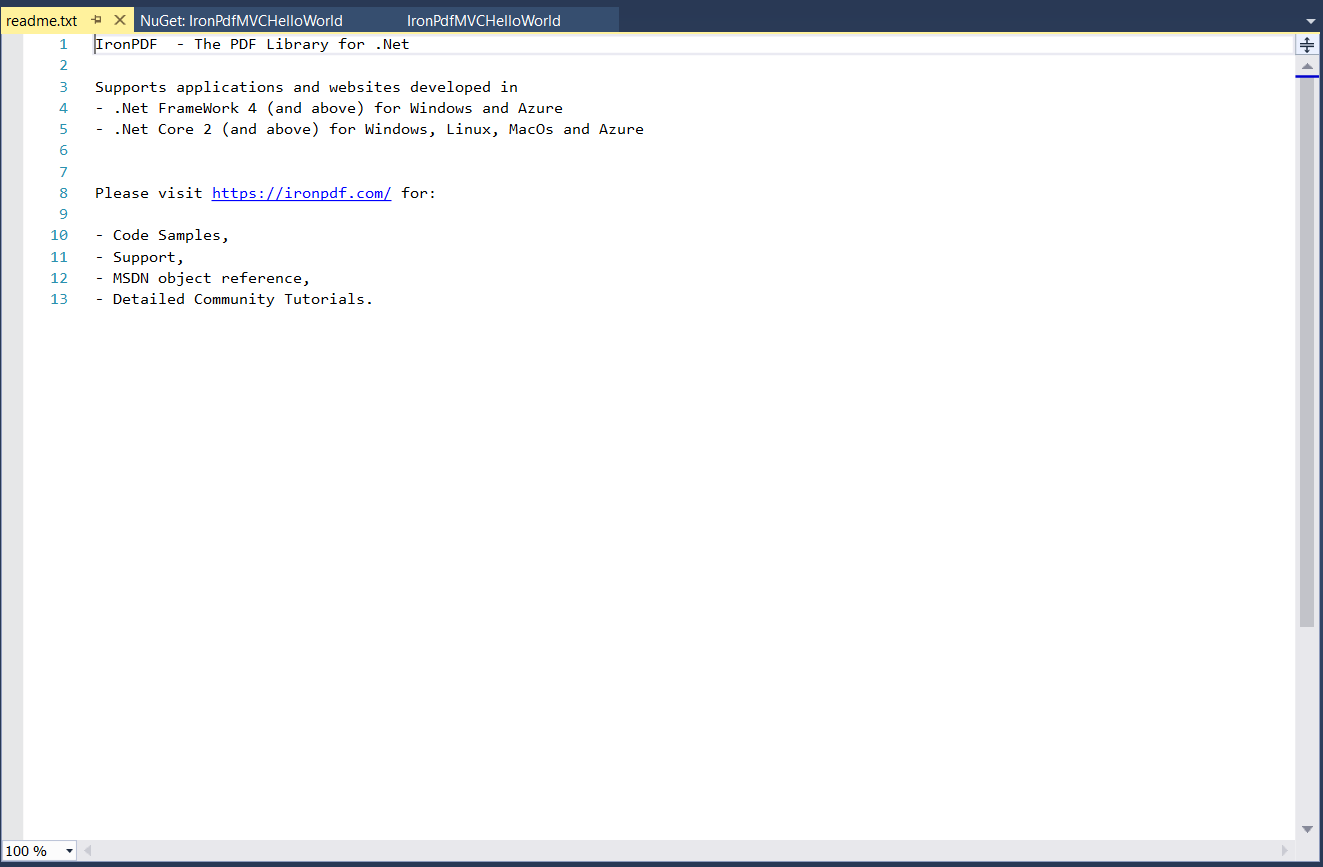
IronPdf can be installed and used on all of .NET projects type like windows application, ASP.NET MVC and .Net Core Application.

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: -

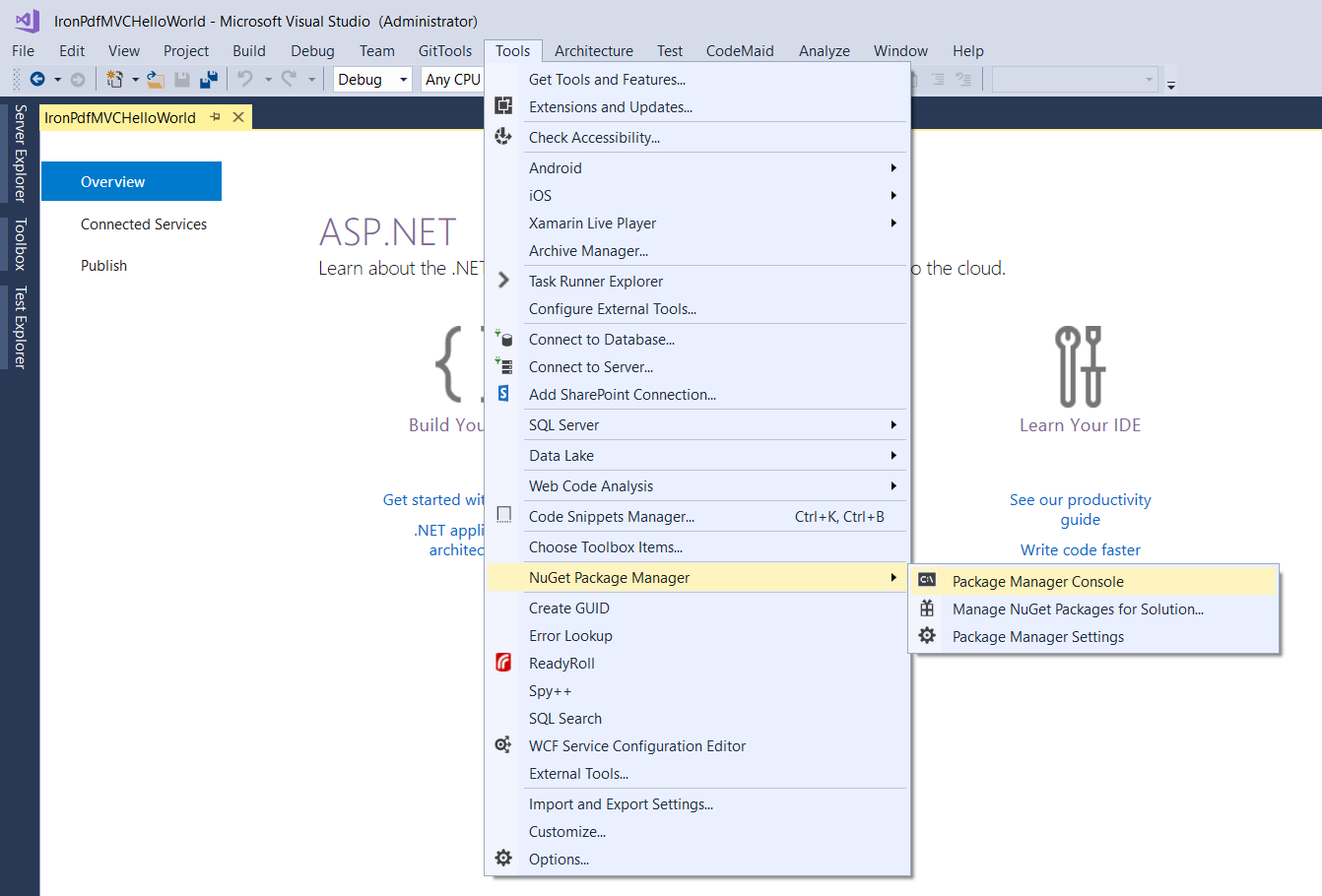
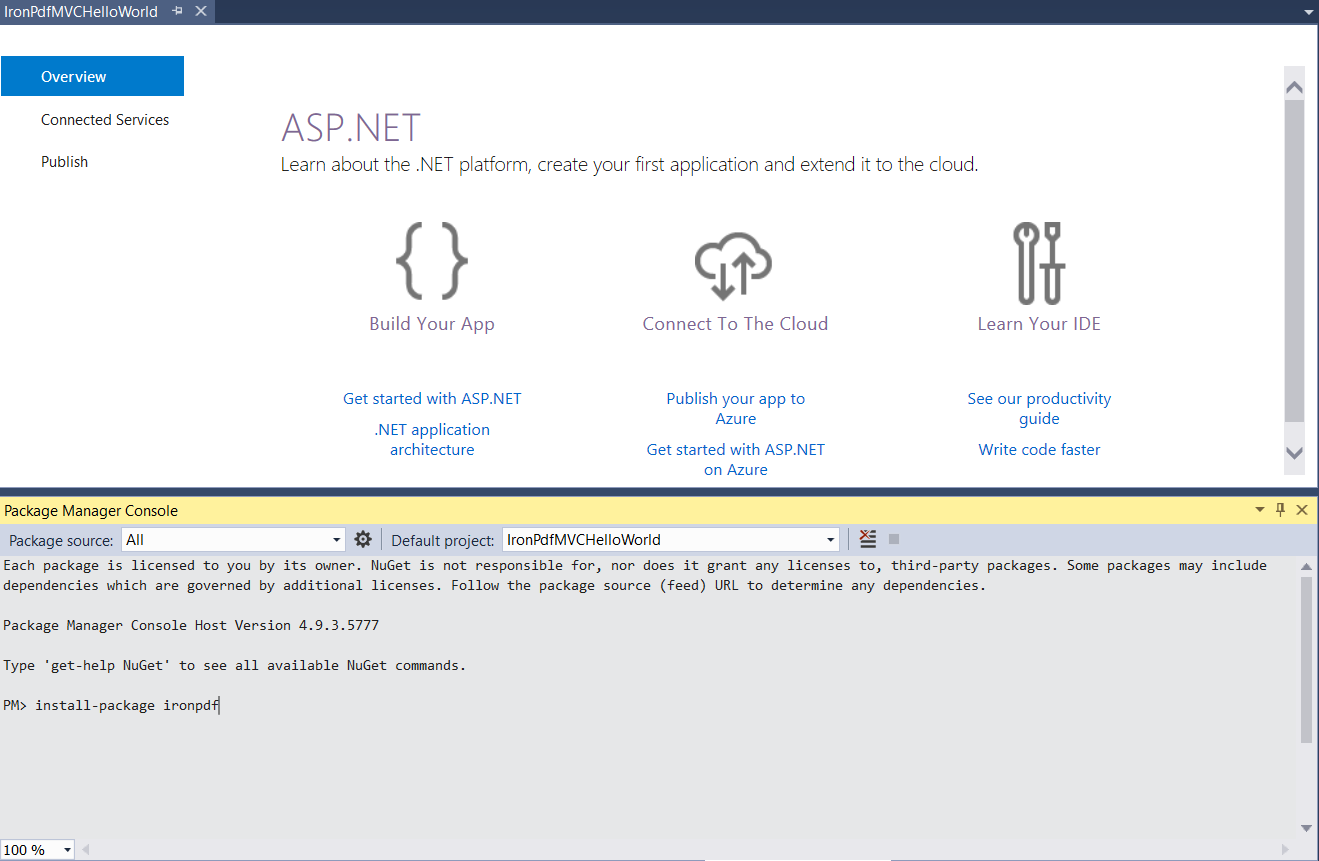
## Install using NuGet

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

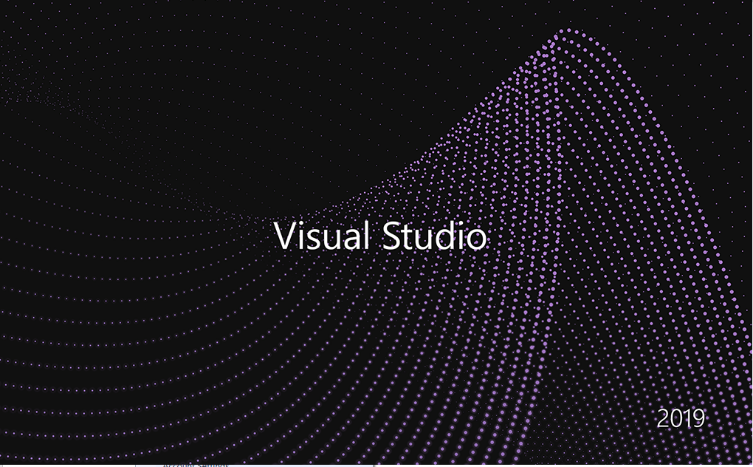
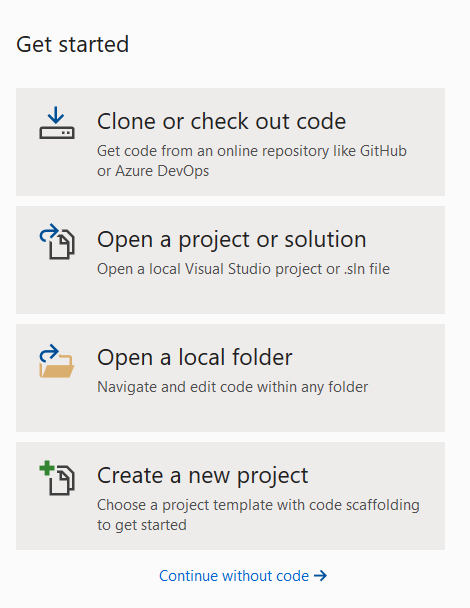
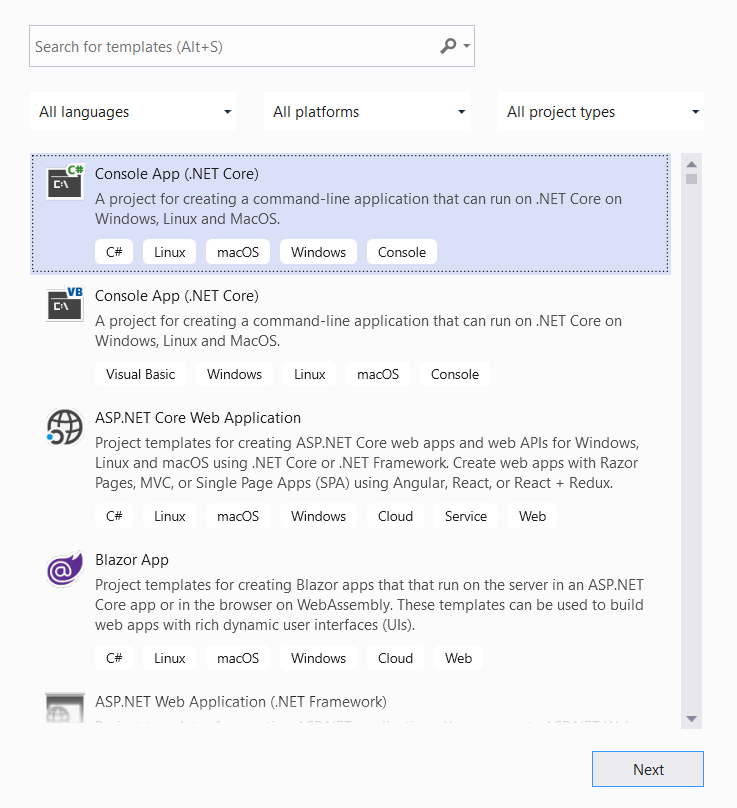
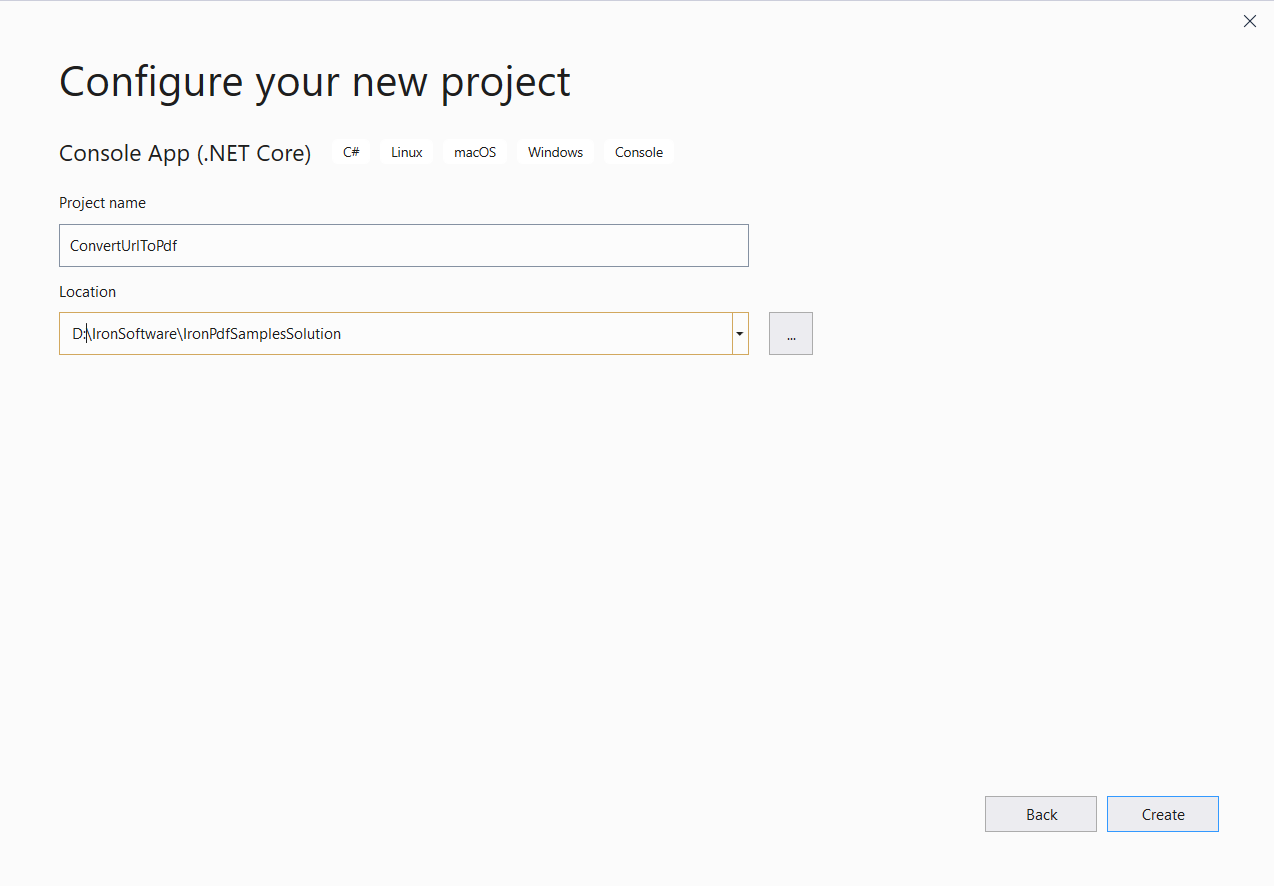
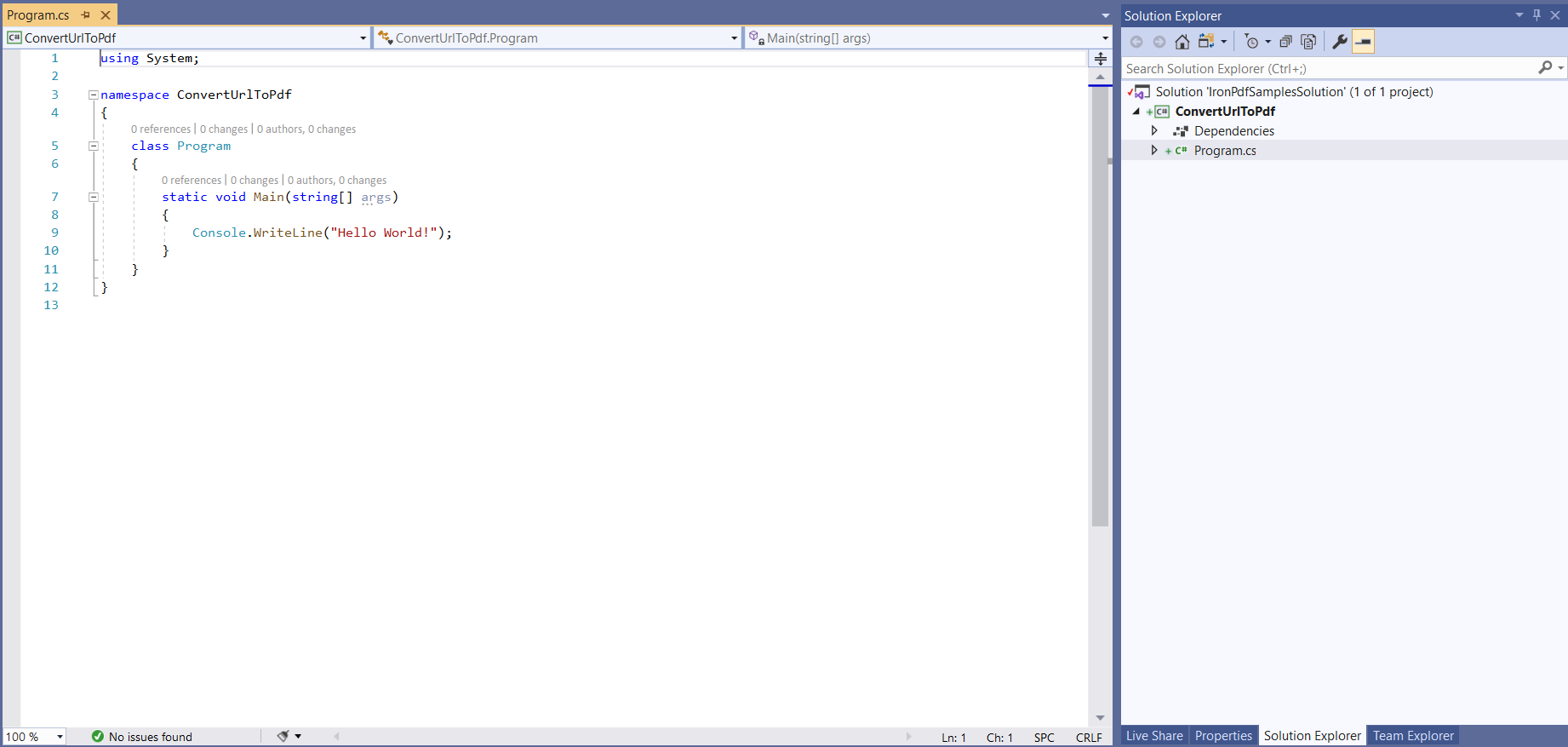
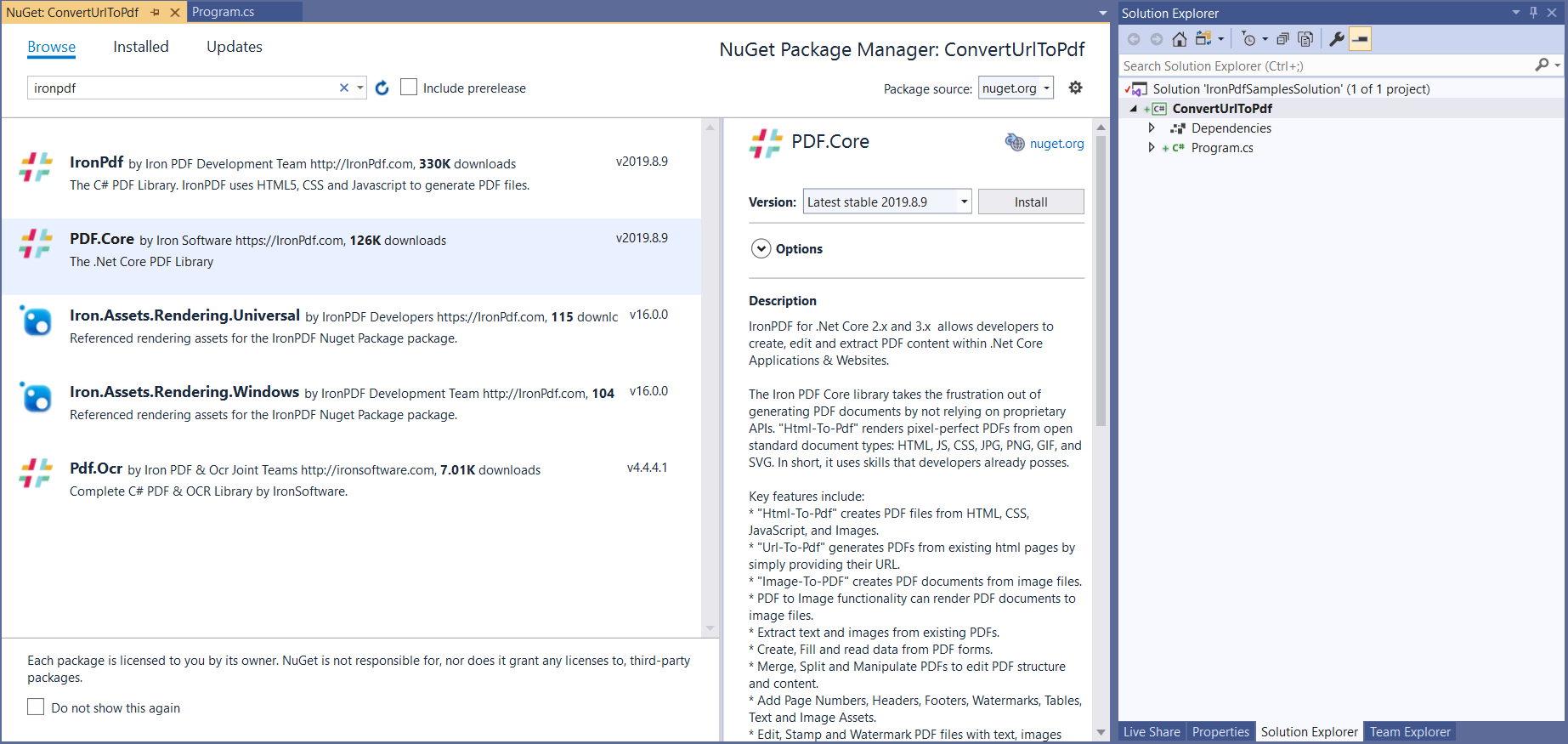
### Sample 2: HelloWorldCore .Net Core Web Application

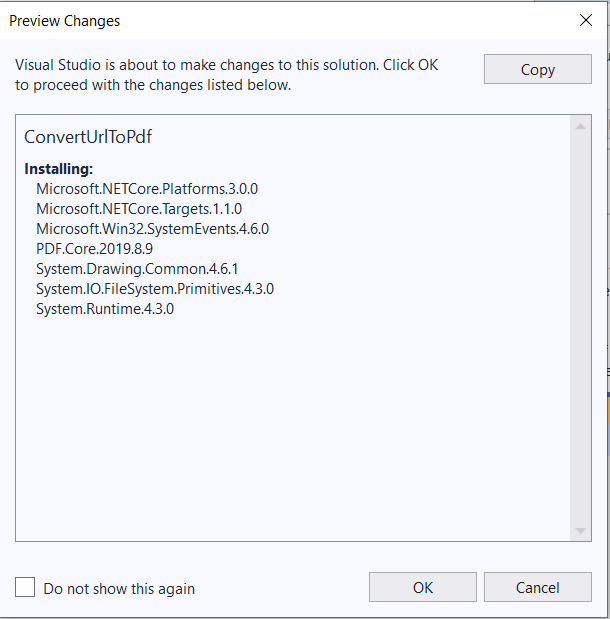
# Chapter 2: Convert to Pdf

## 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 



1. 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");  } |

1. Run and check created file wiki.pdf   
   

## Convert HTML to Pdf

### Sample: ConvertHTMLToPdf Console application

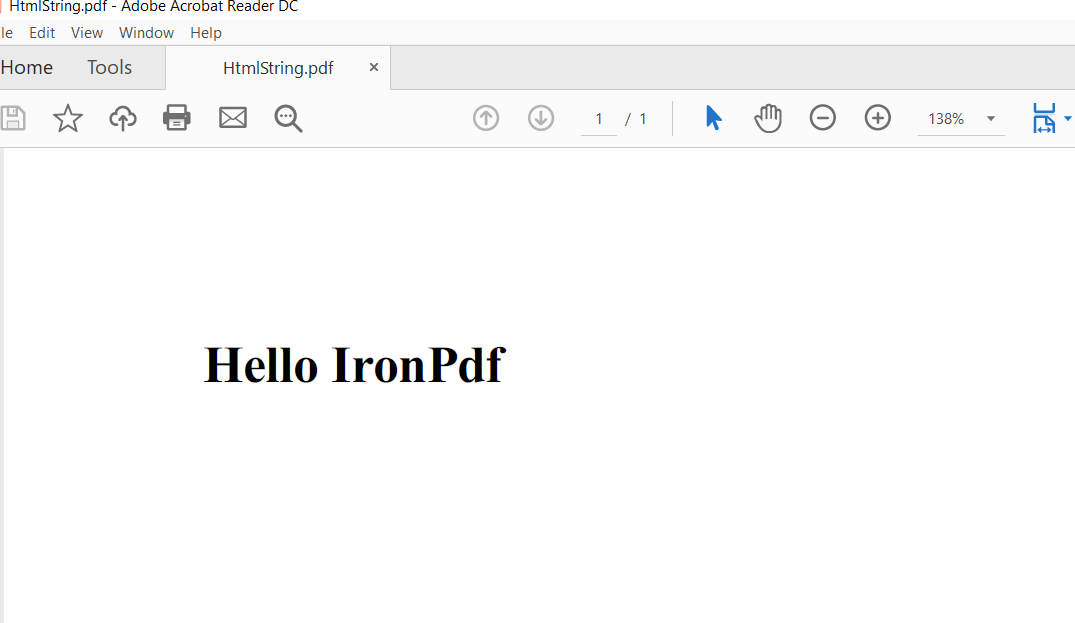
To render HTML to Pdf we have 2 way

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 view to Pdf

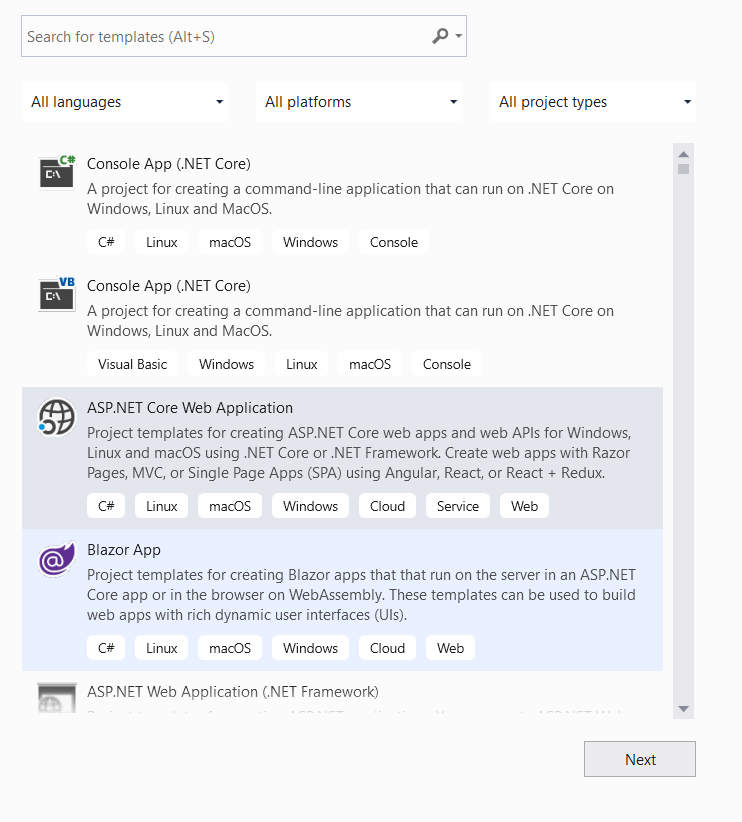
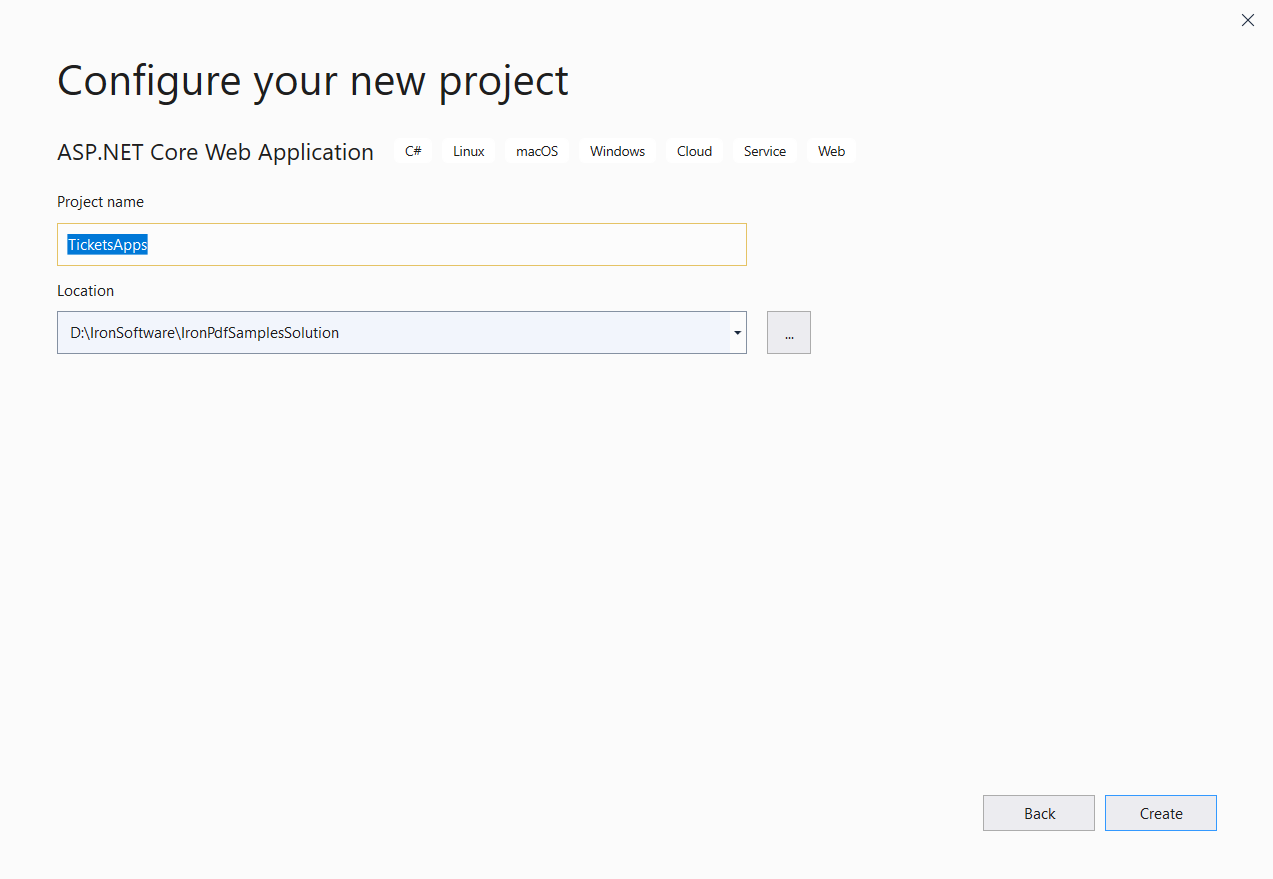
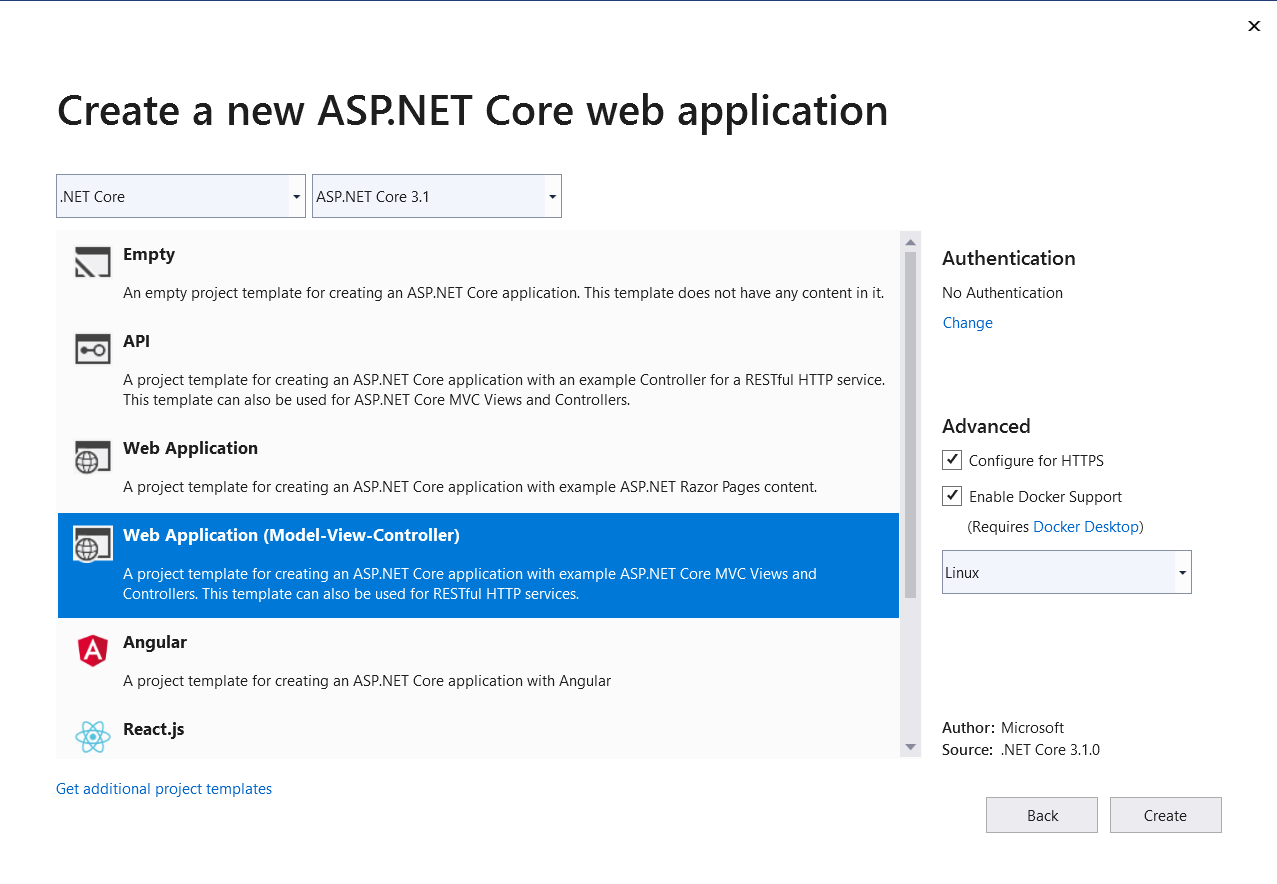
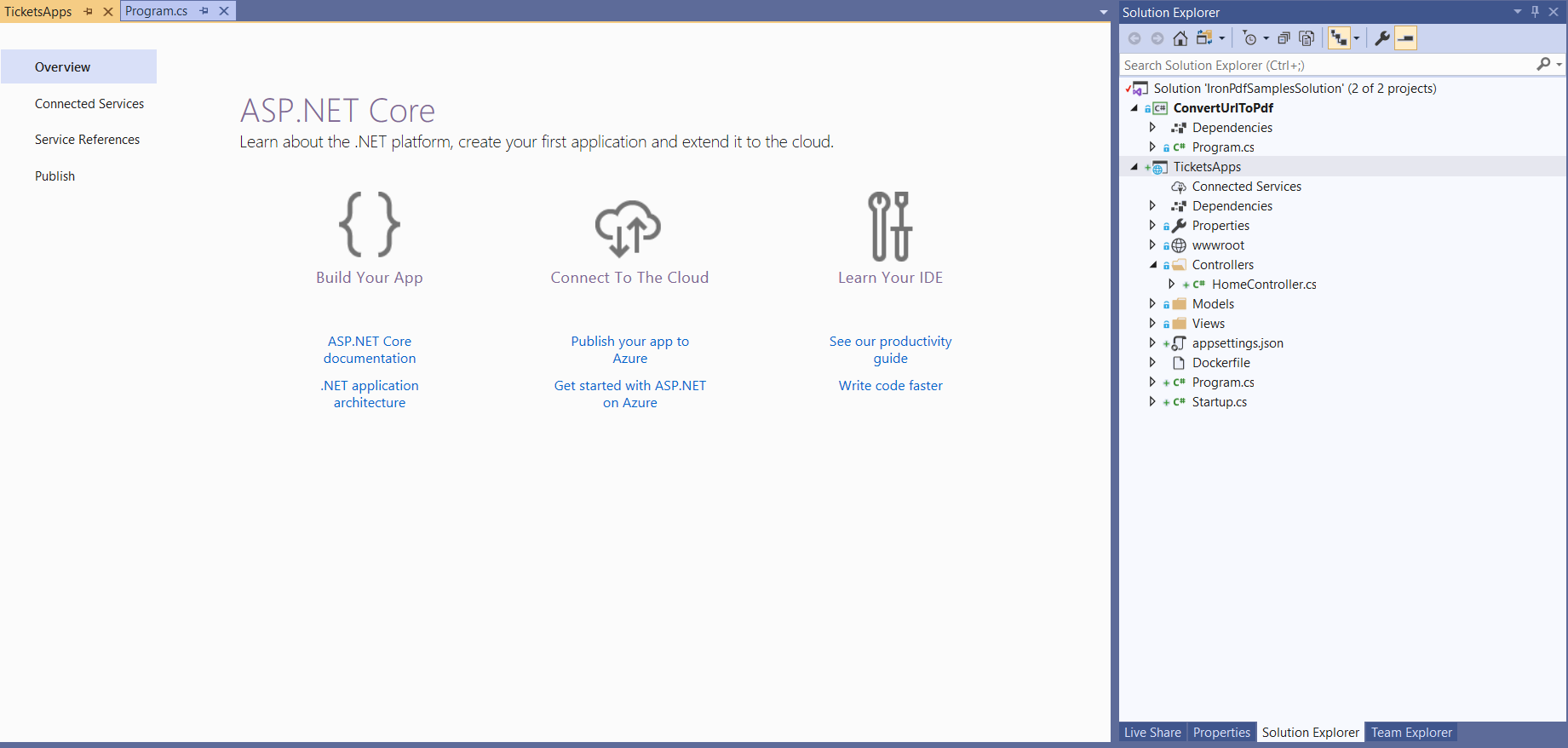
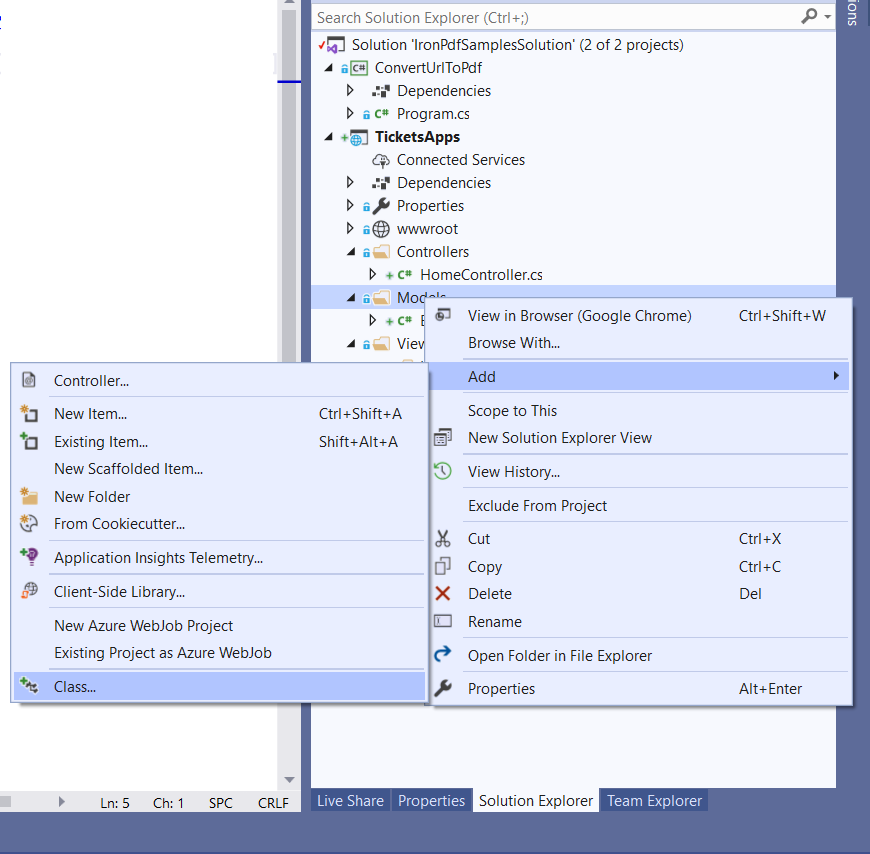
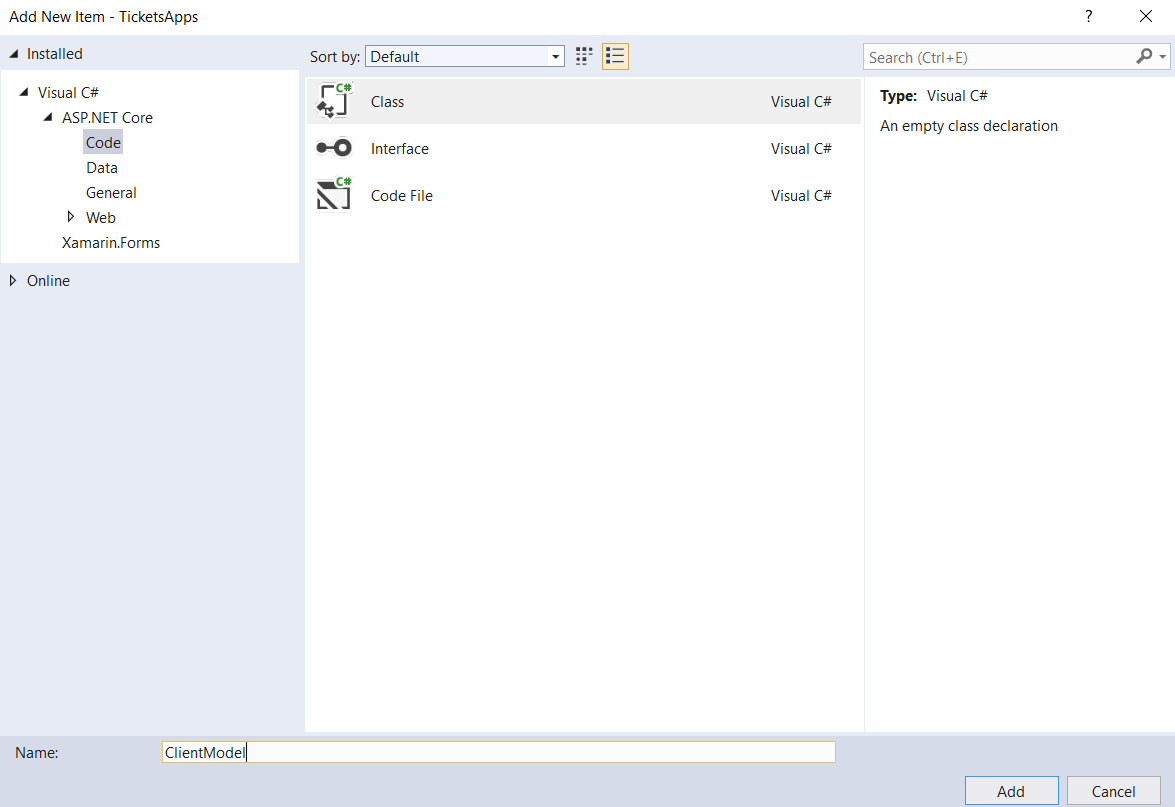
### Sample: TicketsApps .NET Core MVC Application

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

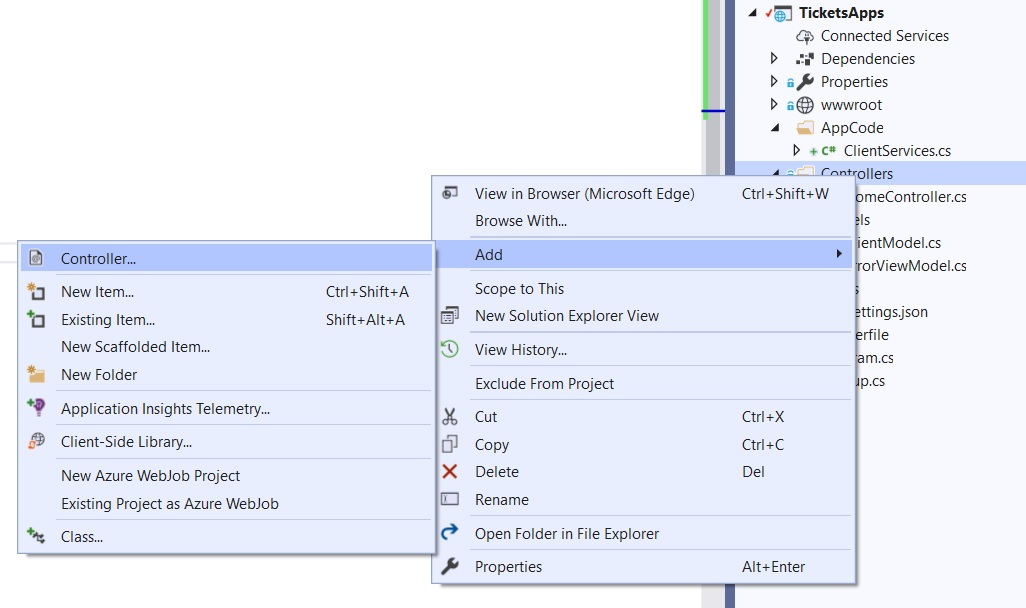
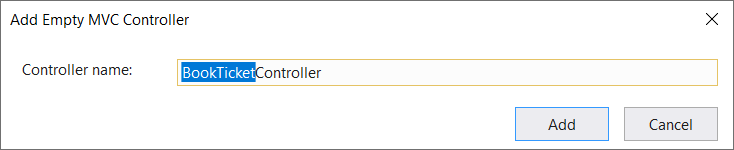
So now I’ll start by creating client object model

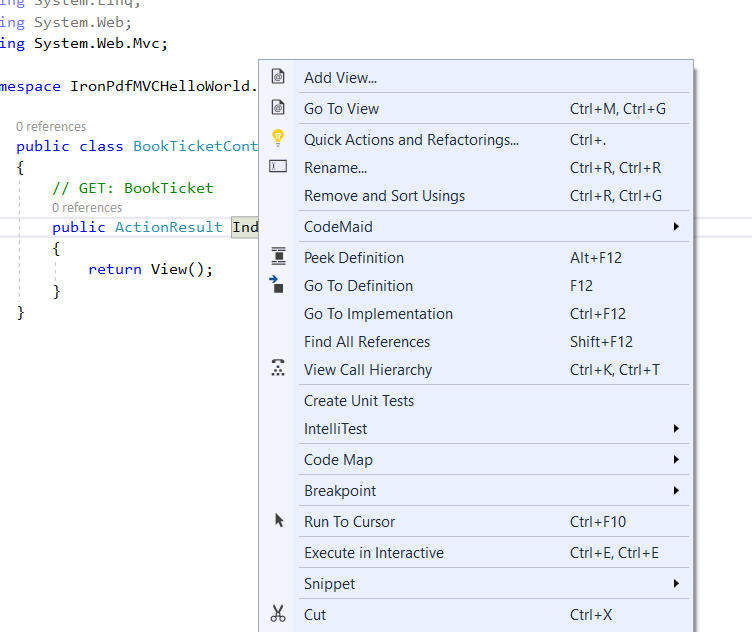
1. Choose ASP.NET core web applications   
   
2. Name the project “TicketsApps”  
   
3. Choose “.NET Core”, “ASP.NET core 3.1” , “Web Application (Model-View-Controller)” , check enable docker and choose Linux Image  
   
4. Now it’s ready   
   
5. Right click on models’ folders choose to add choose class  
   
6. Name the model “ClientModel” then click add  
   
7. 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; }  } |

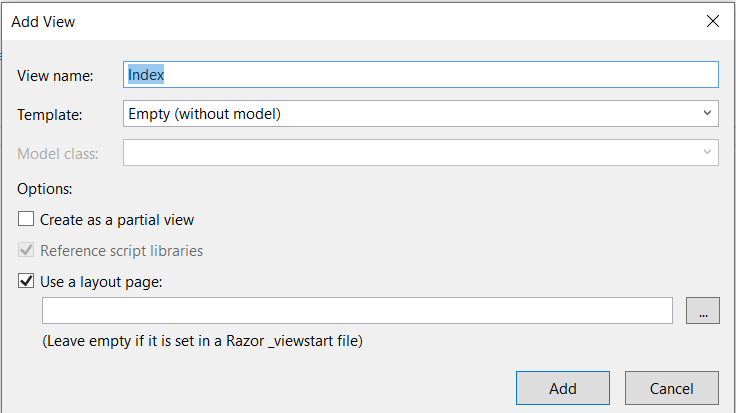
1. Step 2 add services
   1. Create folder and with name “services”
   2. Then add class with name “ClientServices”
   3. add static object of type “ClientModel” to use it as repository
   4. add two functions one for saving client to repository and 2nd to get saved clients

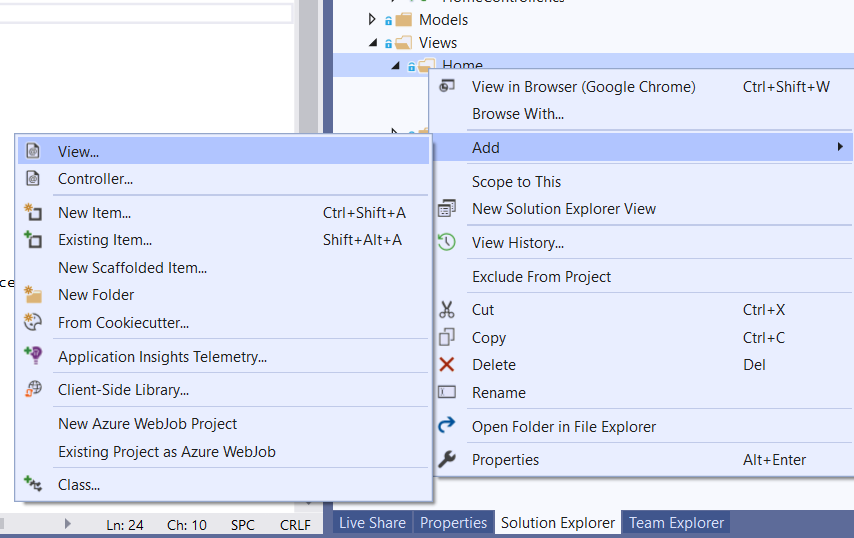
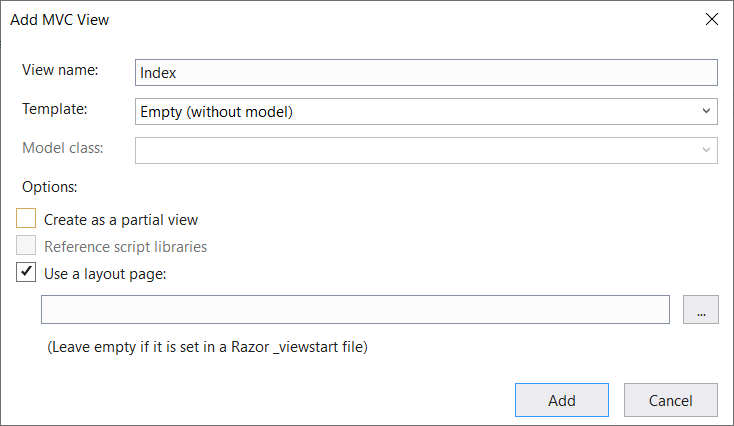
|  |
| --- |
| public class ClientServices  {  private static ClientModel \_clientModel;  public static void AddClient(ClientModel clientModel)  {  \_clientModel = clientModel;  }  public static ClientModel GetClient()  {  return \_clientModel;  }  } |

1. 3rd steps book your ticket page
2. From solution explorer right click over controller folder choose add then choose controller   
   
3. Name it BookTicketController  
   
4. Right click on index function (or we called it action) and choose add view to add html



1. Set view name “index” then click add



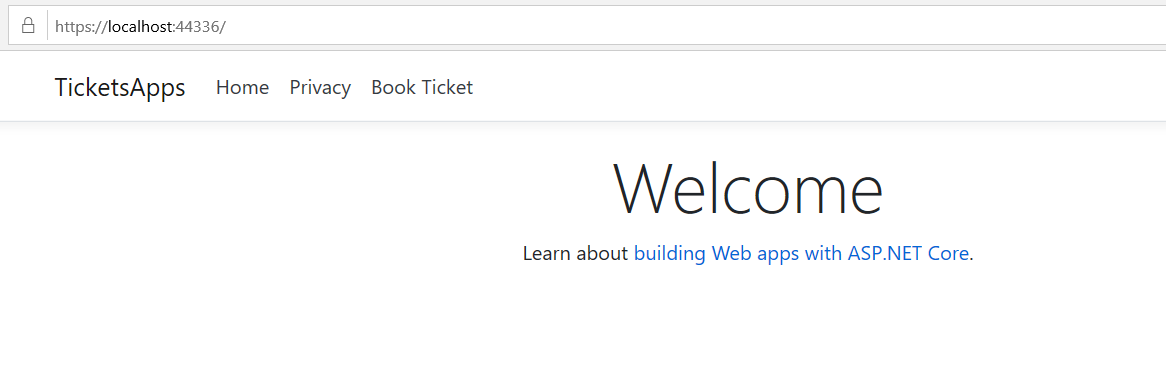
1. using mouse right click over folder views -> Home and select home   
   
2. add index view   
   
3. update html as follow

|  |
| --- |
| @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>  } |

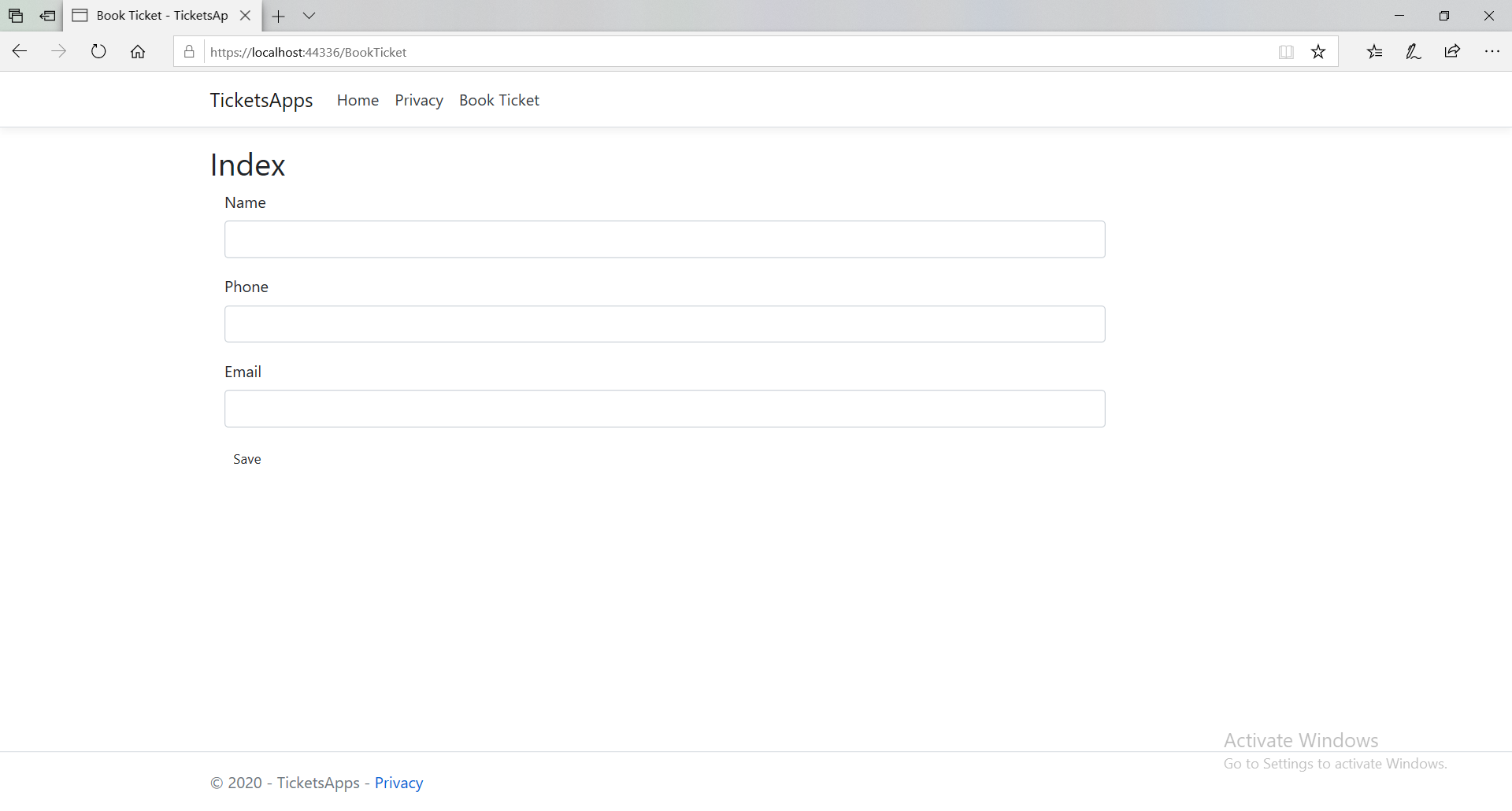
1. add link to BookTicket Page to enable our website visitors to navigate to our new booking page by updating layout exist in path (view-> shared-> layout.chtml)

|  |
| --- |
| <li><a class="nav-link text-dark" **asp-area**="" **asp-controller**="BookTicket" **asp-action**="Index">Book Ticket</a></li> |

1. result should be looks like this



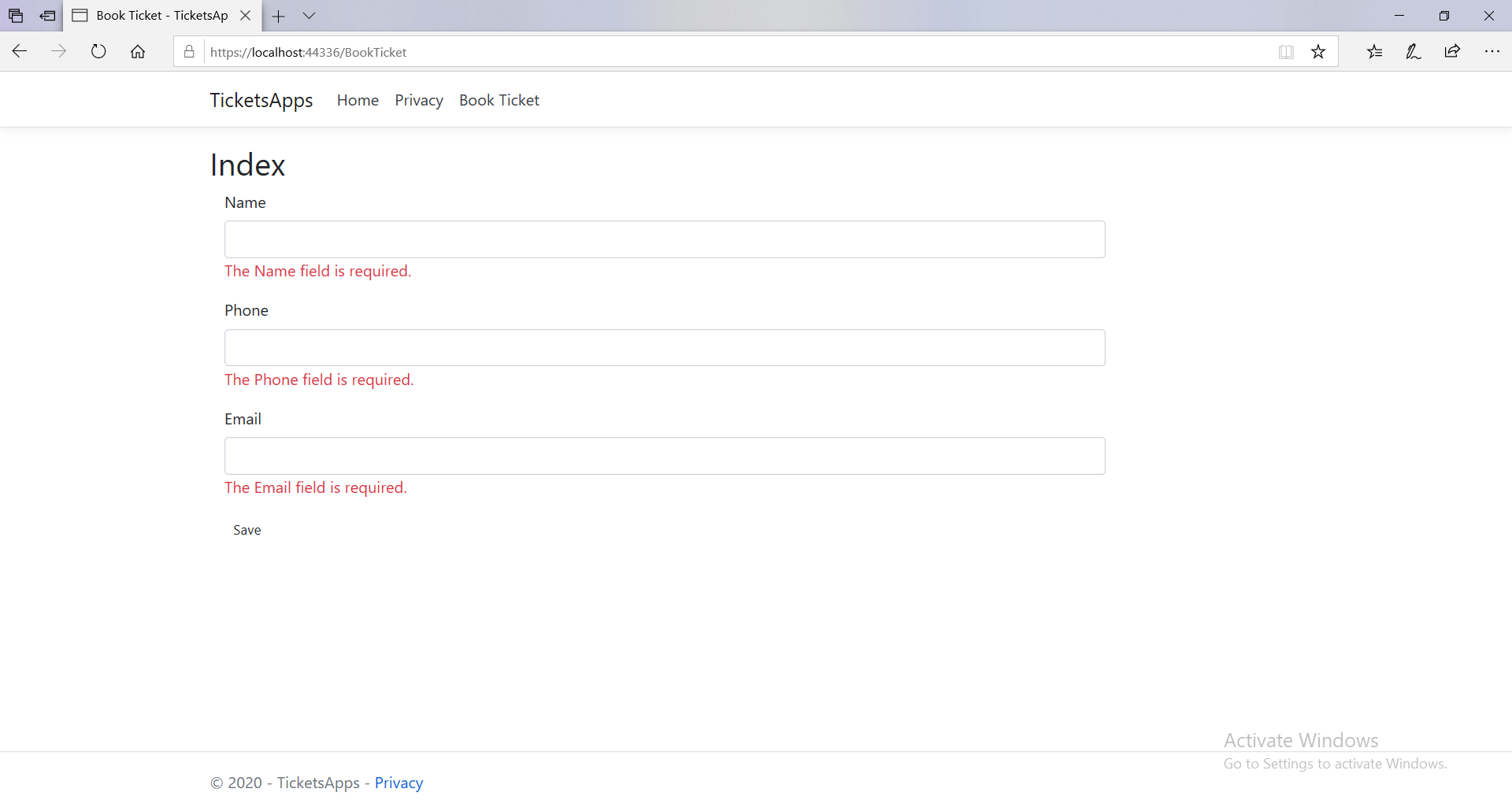
1. Navigate to book ticket page by clicking on its link you should find it look like this



1. Now let’s add the action that will validate and save the book information
2. Add another index action with the attribute [HttpPost] to inform MVC engine that this action is for submitting data, I validate the sent model if it valid code will redirect the visitor to TicketView Page if not a 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);  } |

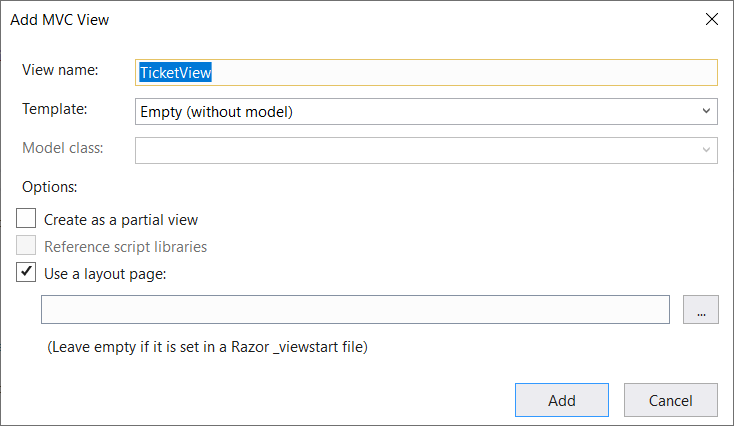
* 1. Sample of error messages

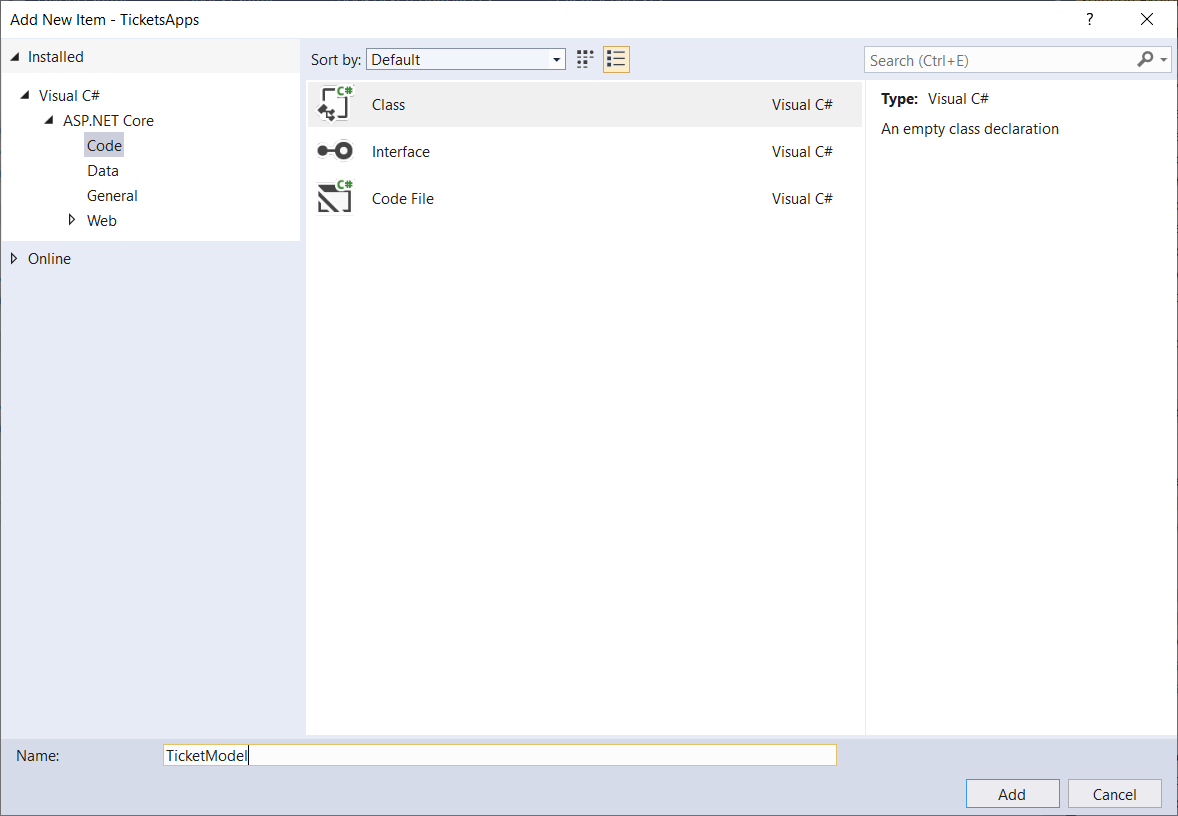


1. add TicketView to display our ticket

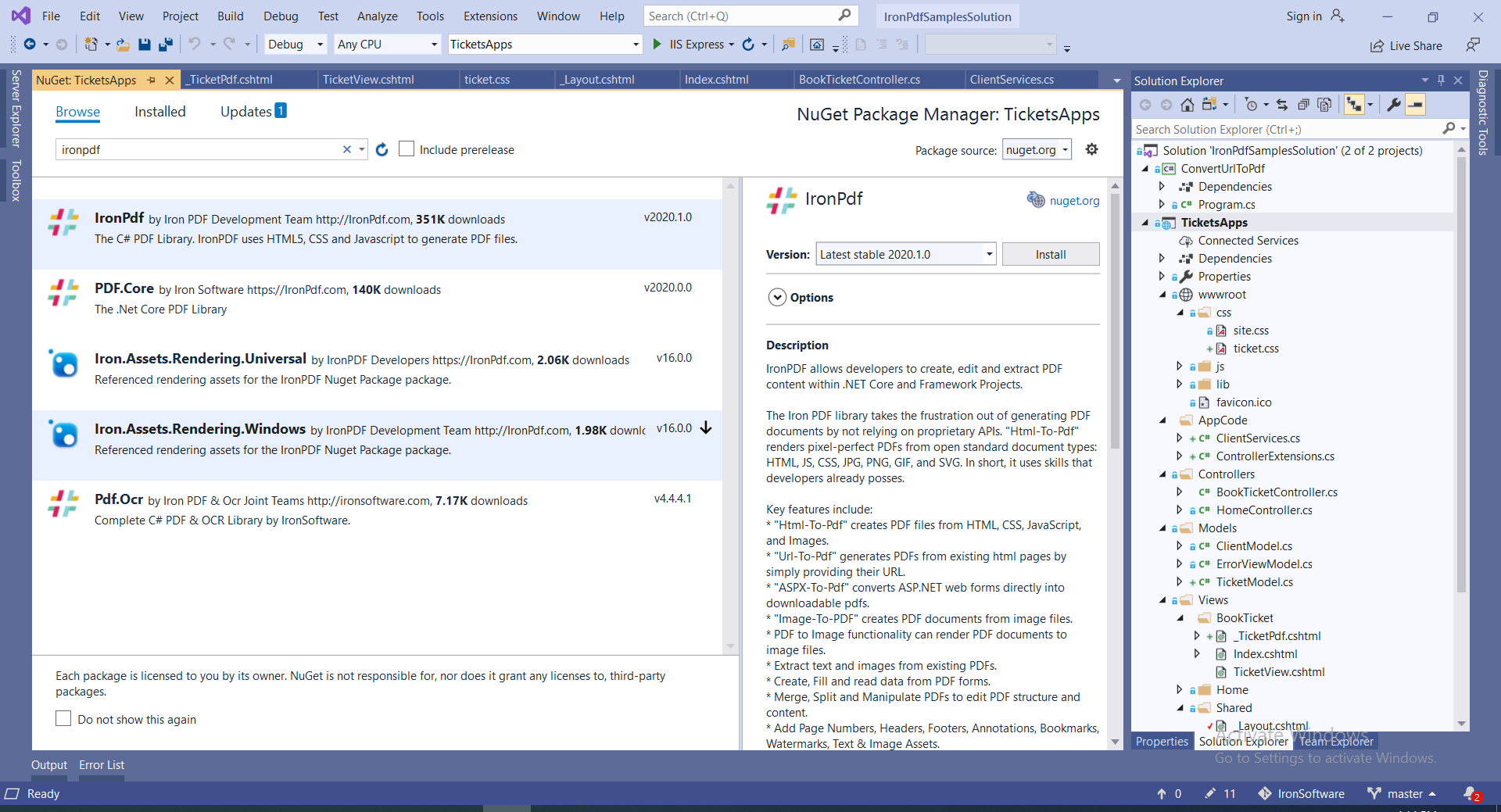
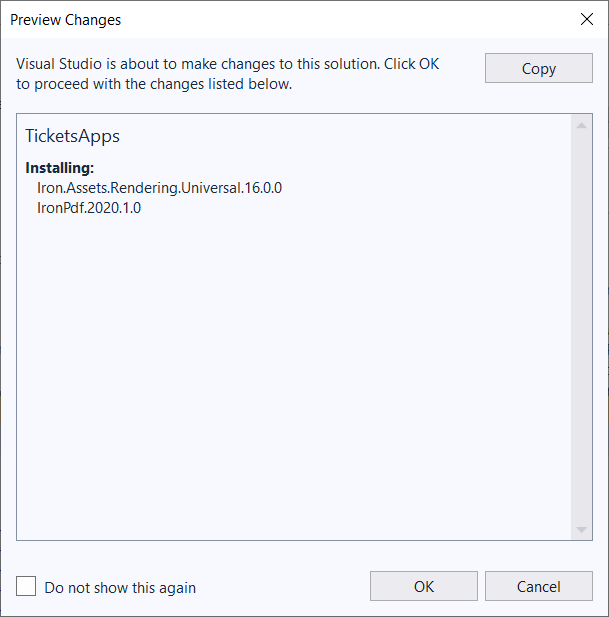
|  |
| --- |
| public ActionResult TicketView()  {  var ticket = ClientServices.GetClient();  return View(ticket);  } |

1. Add its view

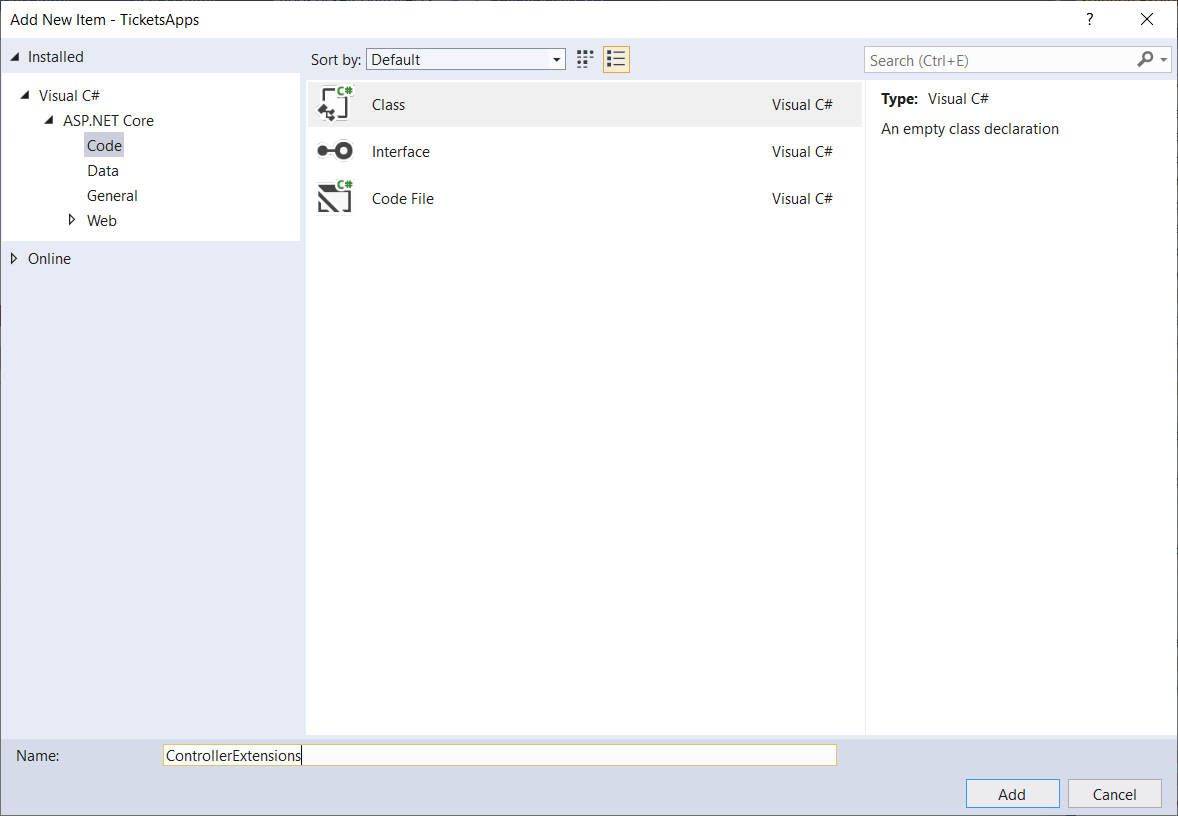


1. This view will host Ticket partial view that is responsible to display the ticket and will be used later to print Ticket
2. Add ticket model  
   
3. Ticket model code as follow

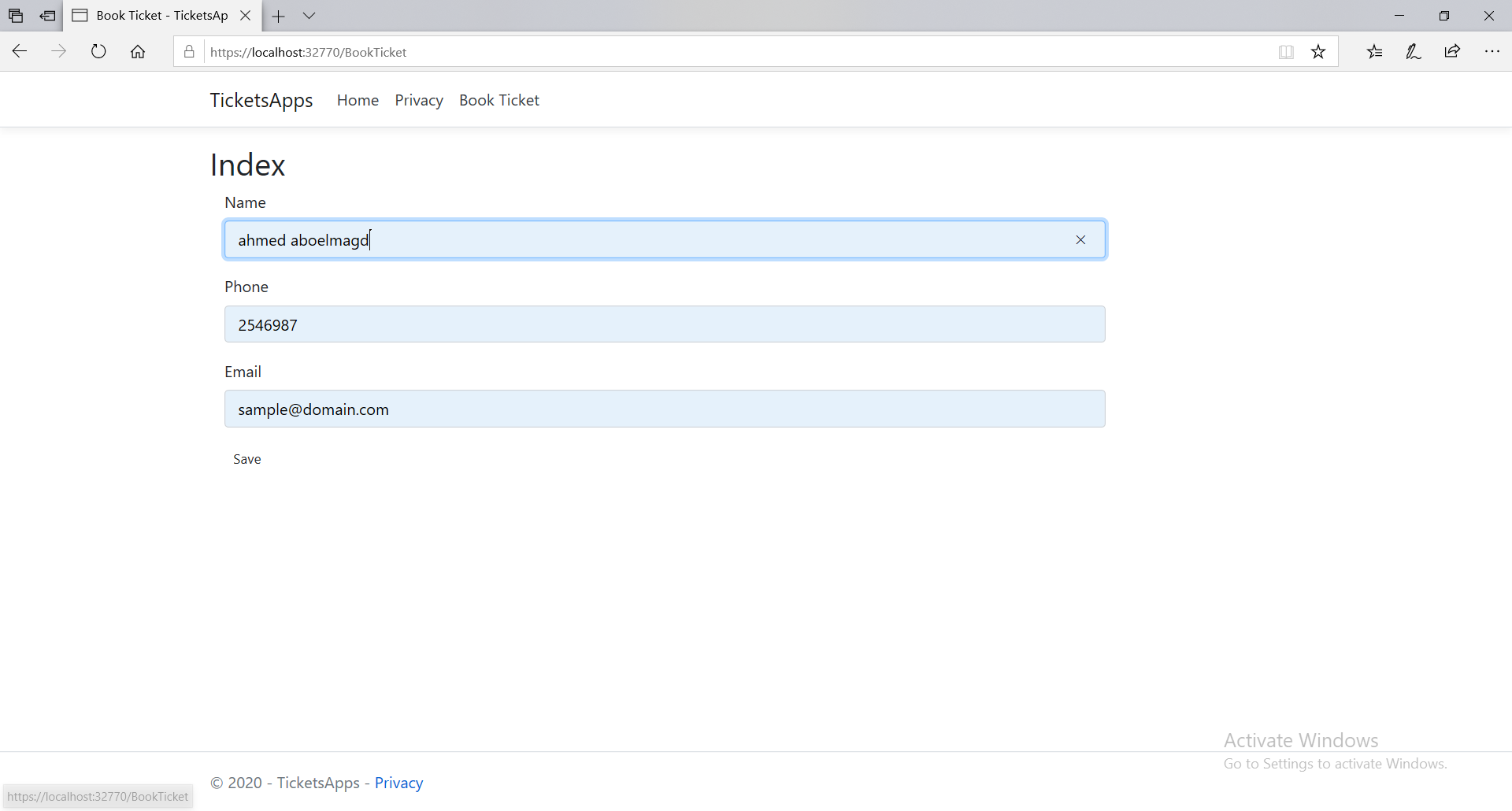
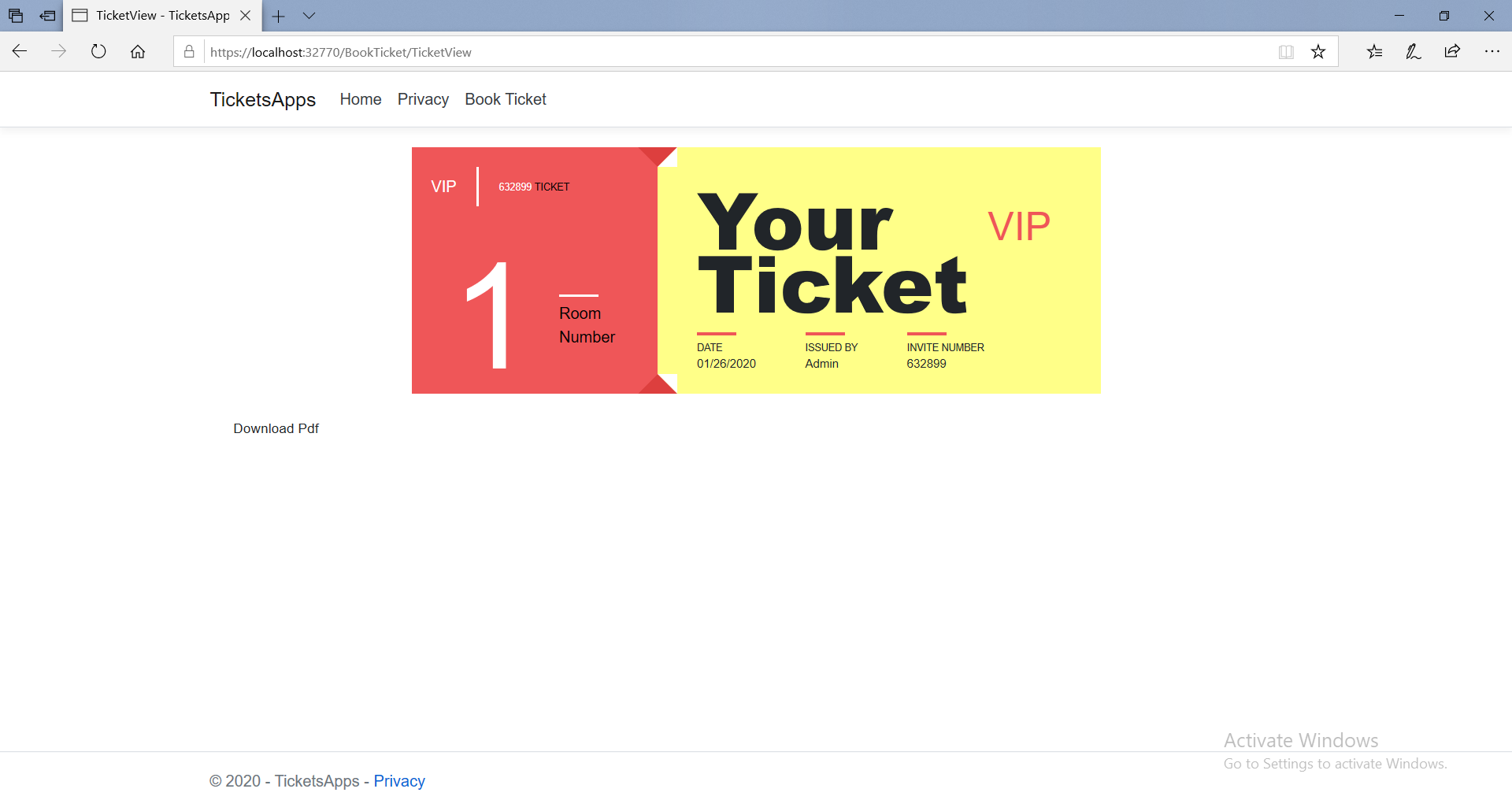
|  |
| --- |
| public class TicketModel : ClientModel  {  public int TicketNumber { get; set; }  public DateTime TicketDate { get; set; }  } |

1. Add ironPdf to project  
   
2. Click ok  
   
3. Add TicketView post method that will handle button download

|  |
| --- |
| [HttpPost]  public ActionResult TicketView(TicketModel model)  {  IronPdf.Installation.TempFolderPath = $@"{\_host.ContentRootPath}/irontemp/";  IronPdf.Installation.LinuxAndDockerDependenciesAutoConfig = true;  var html = this.RenderViewAsync("\_TicketPdf", model);  var ironPdfRender = new IronPdf.HtmlToPdf();  var pdfDoc = ironPdfRender.RenderHtmlAsPdf(html.Result);  return File(pdfDoc.Stream.ToArray(), "application/pdf");  } |

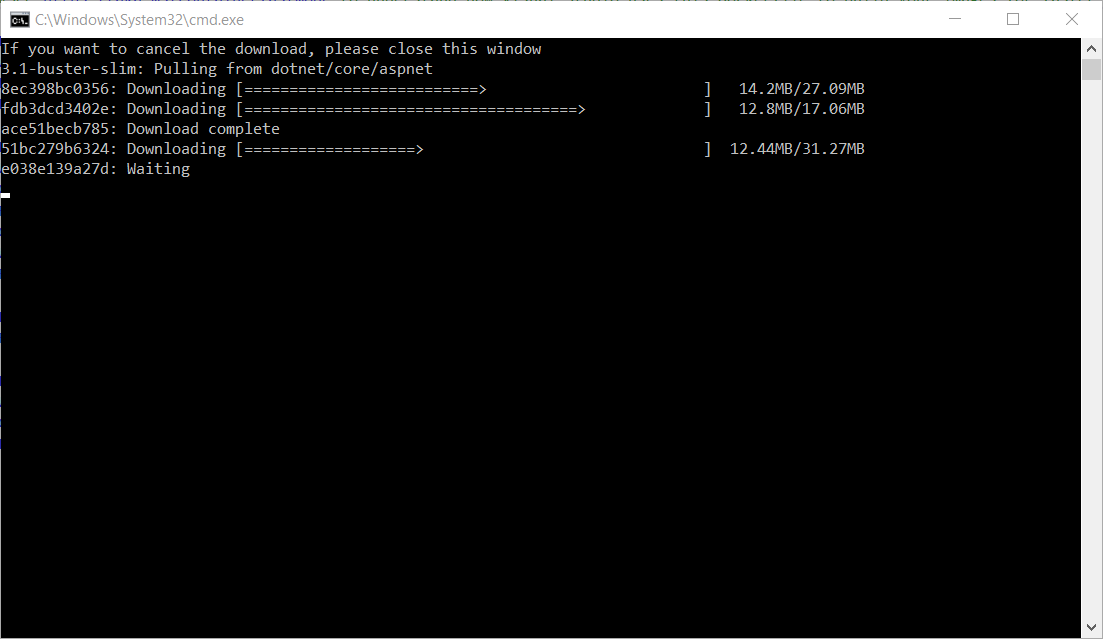
1. Add controller extension that will render partial view to string  
   
2. Extension code as follow

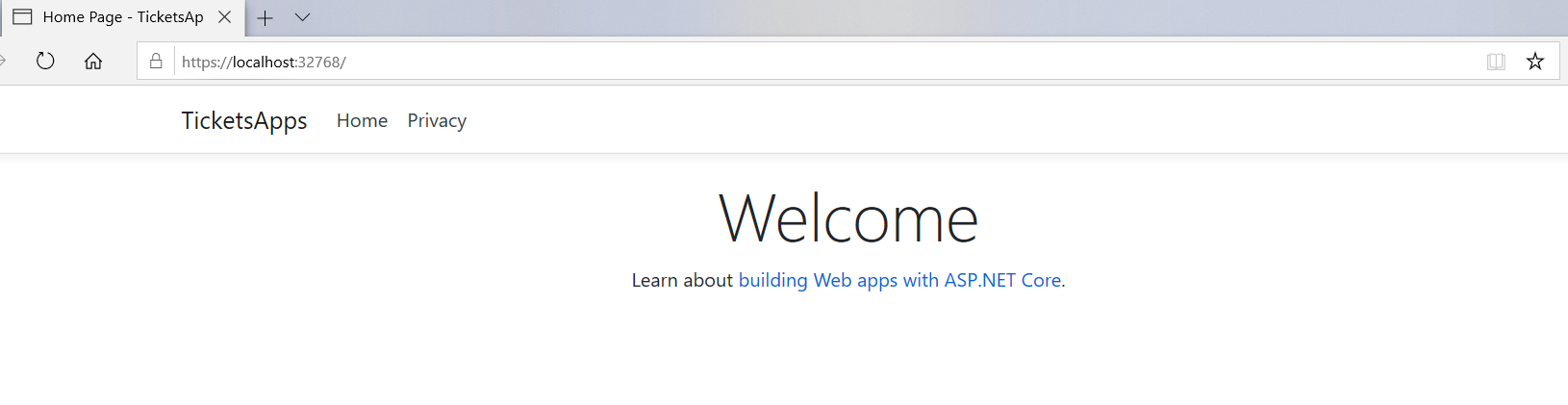
|  |
| --- |
| public static class ControllerExtensions  {  public static async Task<string> RenderViewAsync<TModel>(this Controller controller, string viewName, TModel model, bool partial = false)  {  if (string.IsNullOrEmpty(viewName))  {  viewName = controller.ControllerContext.ActionDescriptor.ActionName;  }  controller.ViewData.Model = model;  using (var writer = new StringWriter())  {  IViewEngine viewEngine = controller.HttpContext.RequestServices.GetService(typeof(ICompositeViewEngine)) as ICompositeViewEngine;  ViewEngineResult viewResult = viewEngine.FindView(controller.ControllerContext, viewName, !partial);  if (viewResult.Success == false)  {  return $"A view with the name {viewName} could not be found";  }  ViewContext viewContext = new ViewContext(  controller.ControllerContext,  viewResult.View,  controller.ViewData,  controller.TempData,  writer,  new HtmlHelperOptions()  );  await viewResult.View.RenderAsync(viewContext);  return writer.GetStringBuilder().ToString();  }  }  } |

1. Now run and file ticket information then click save  
   
2. View ticket   
   
3. To download ticket as pdf click download you will get Pdf contain the ticket

## Advanced options

# Chapter 3: Work with Dockers

Run into Linux container  




Run into windows container

# Chapter 4: Working with Pdf Document

## Open Pdf

## Merge Pdf

## Add Header Or footer to Pdf

## Pdf security

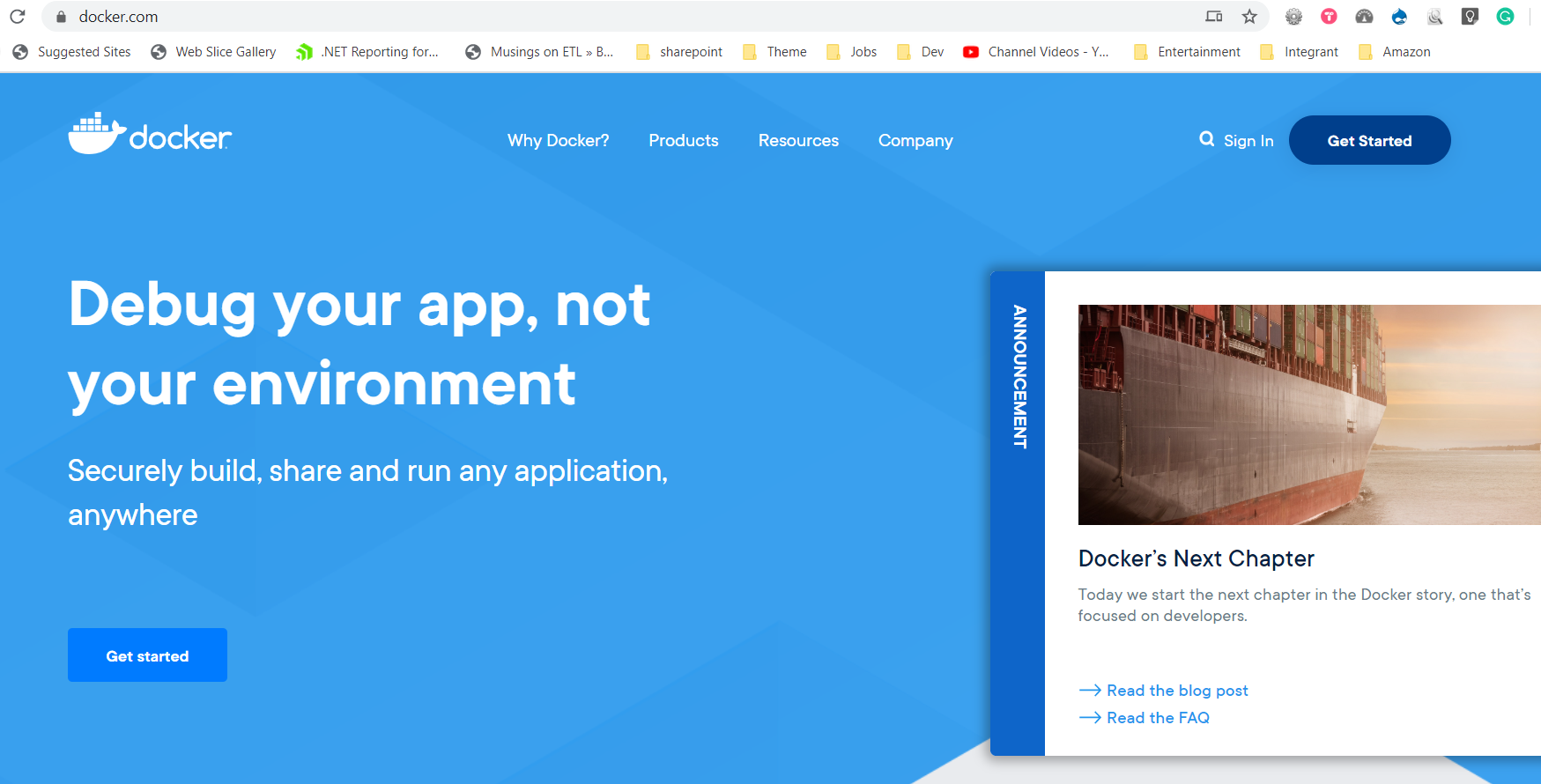
## Pdf extraction and conversions

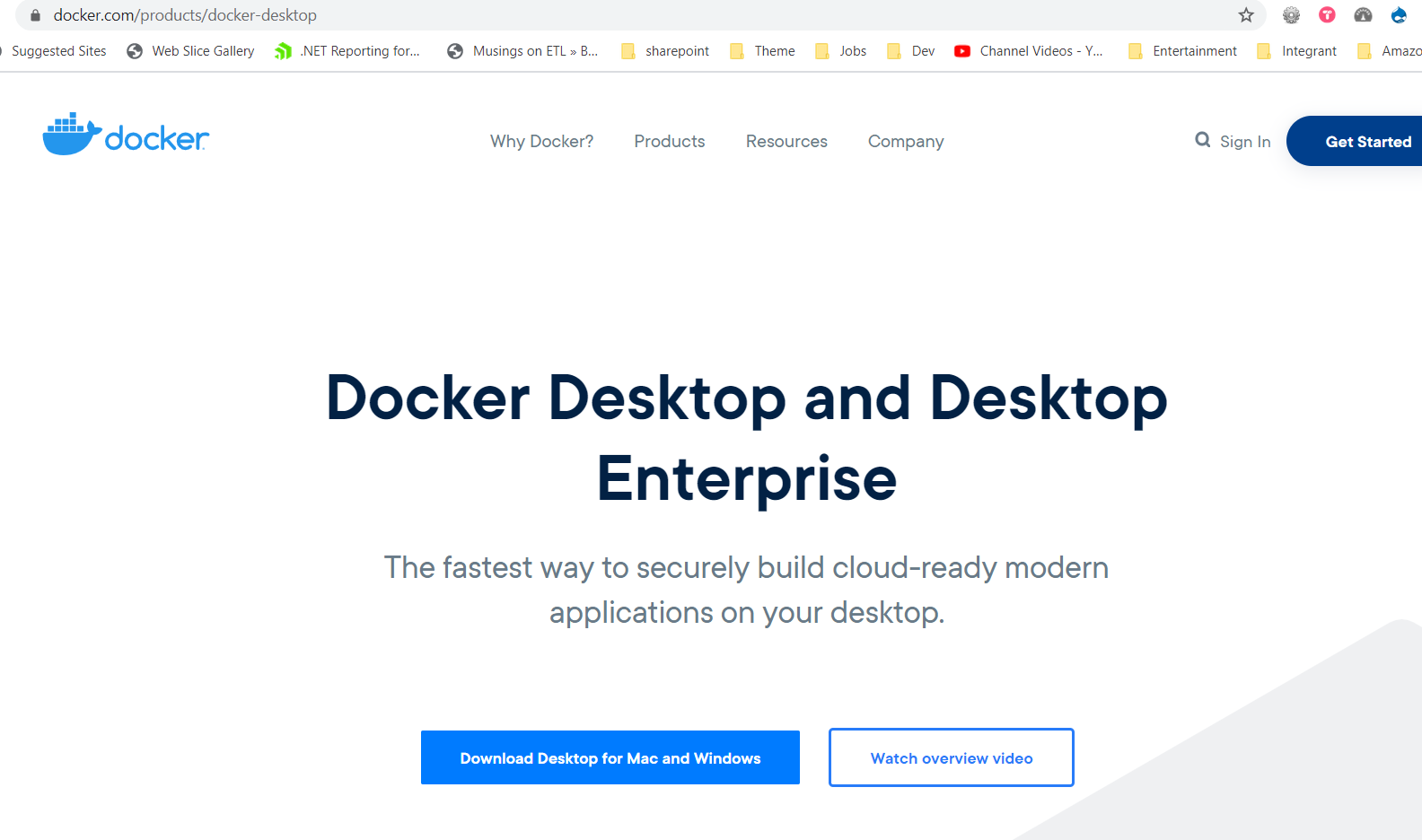
# Summary

# Appendix (A)

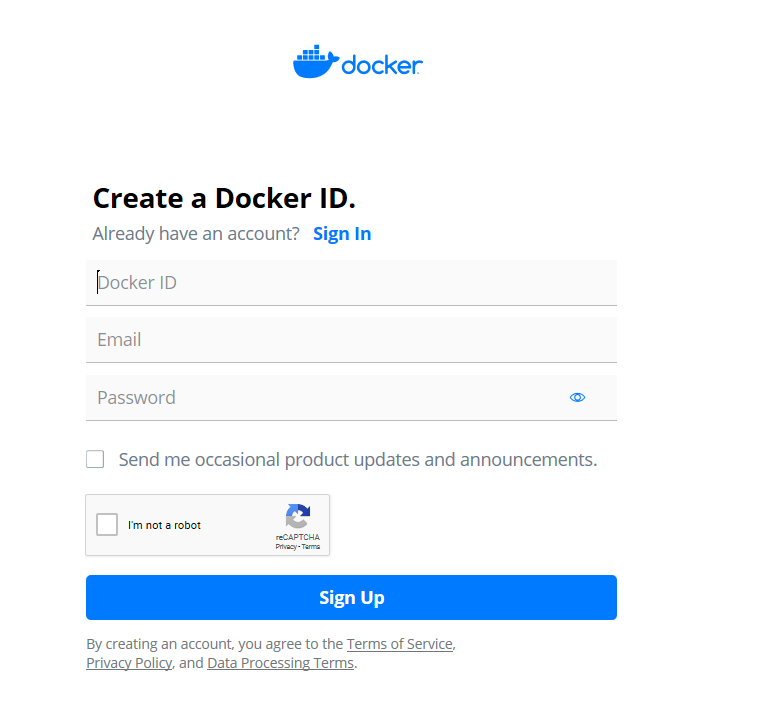
What is Docker

Install Docker

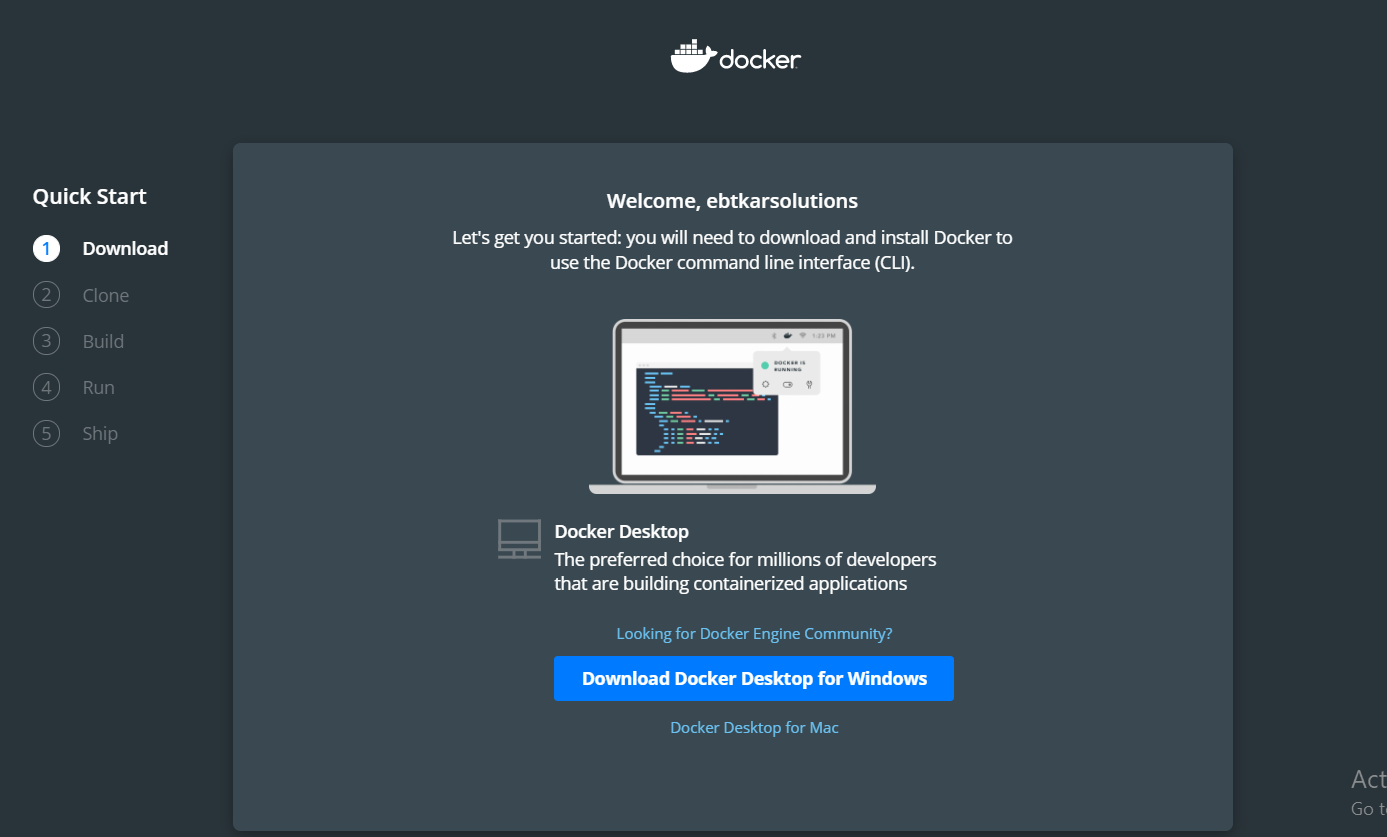
Go to Docker website (<https://www.docker.com/>)  


Click Get started   


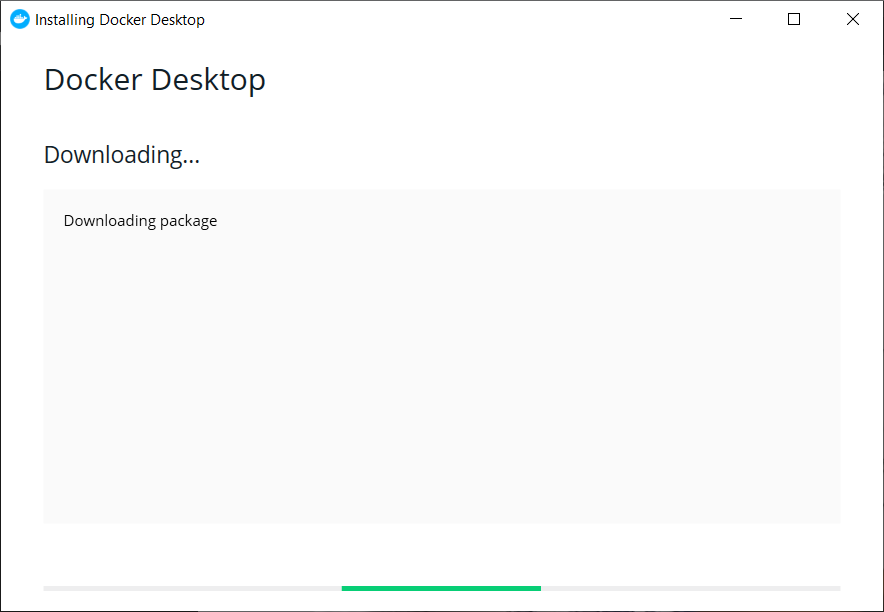
Click download for mac and windows



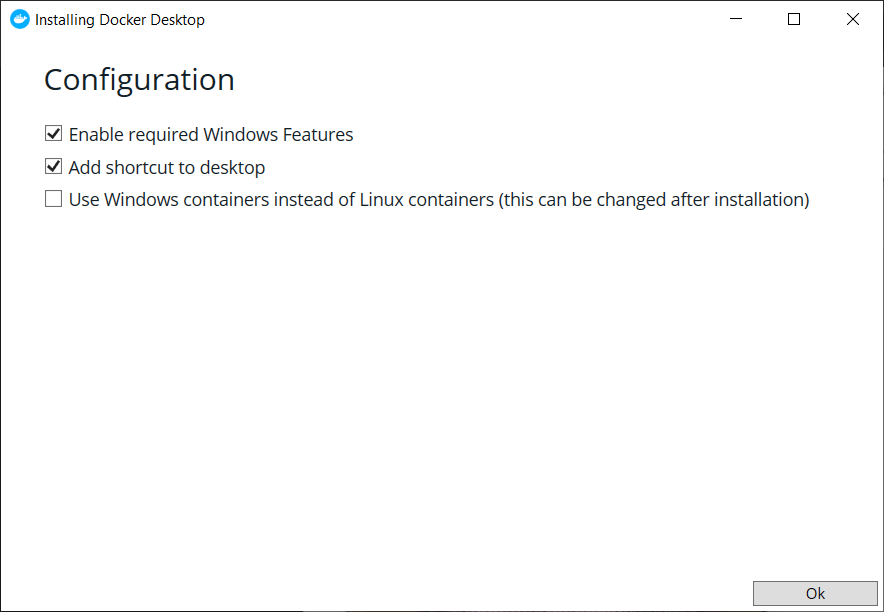
Signup for free then login



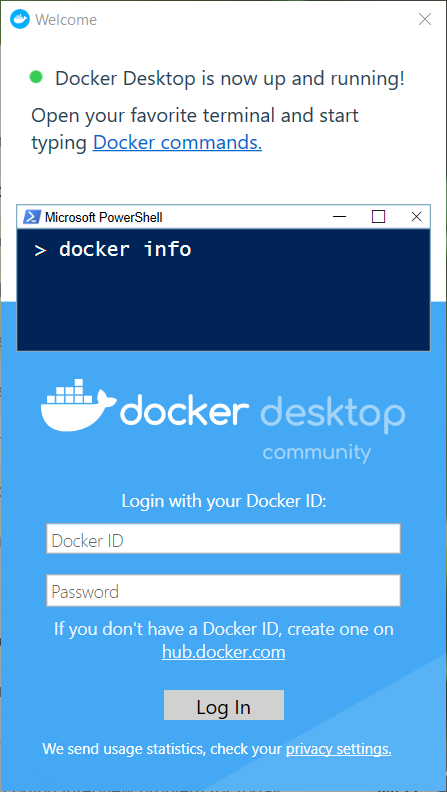
Download Docker for windows



Start installing Docker.

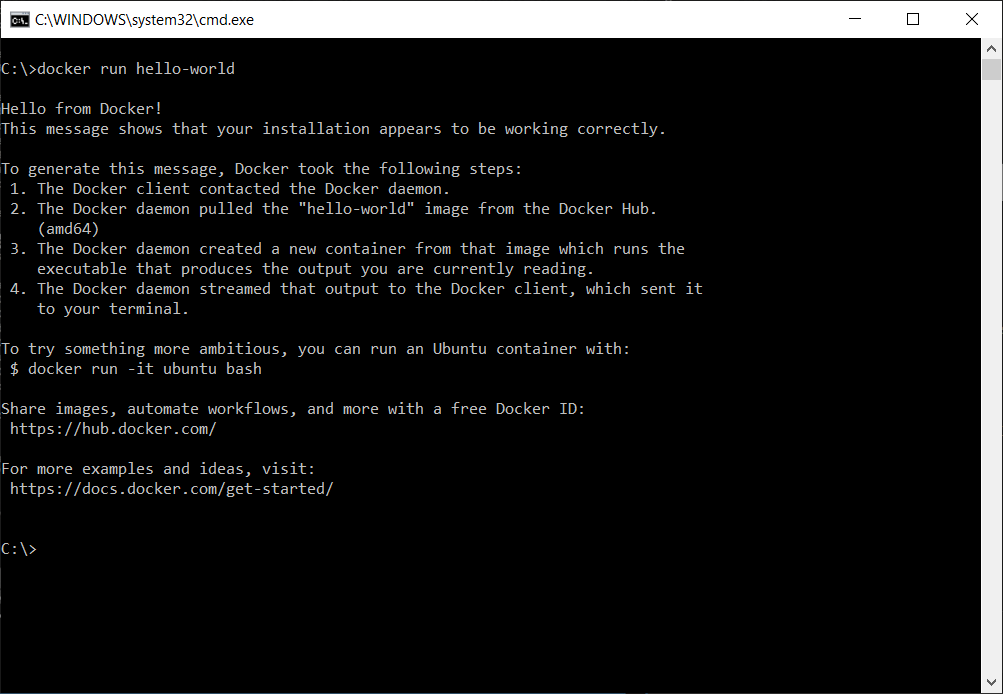


It will require restart and after restart machine login to Docker



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

Docker run hello-world



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.)