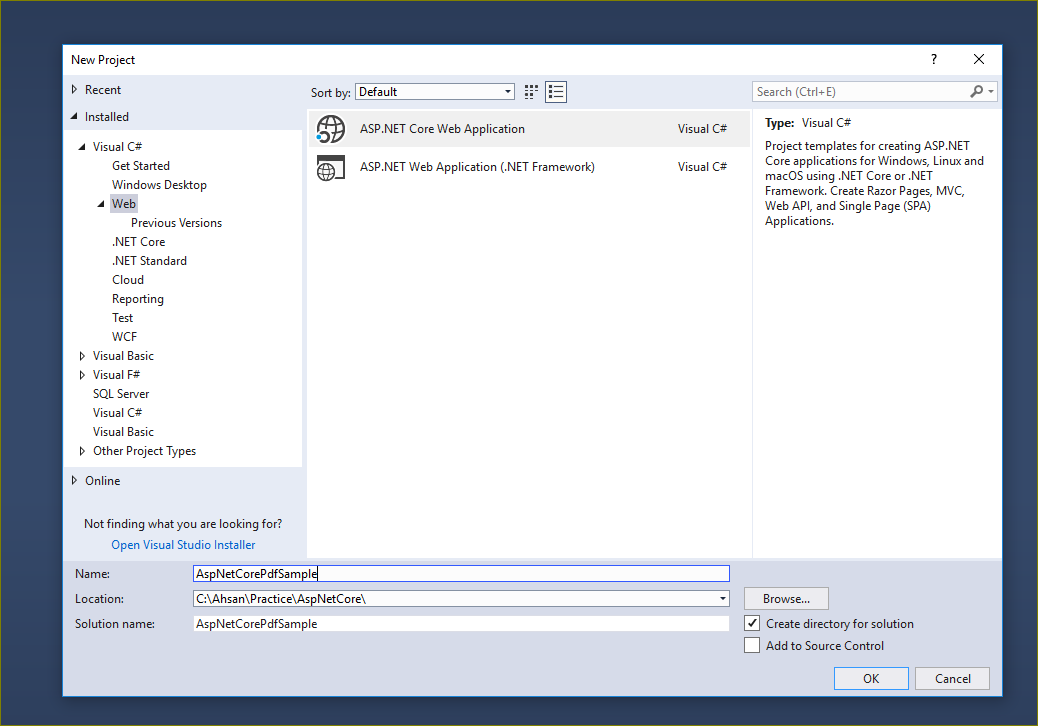
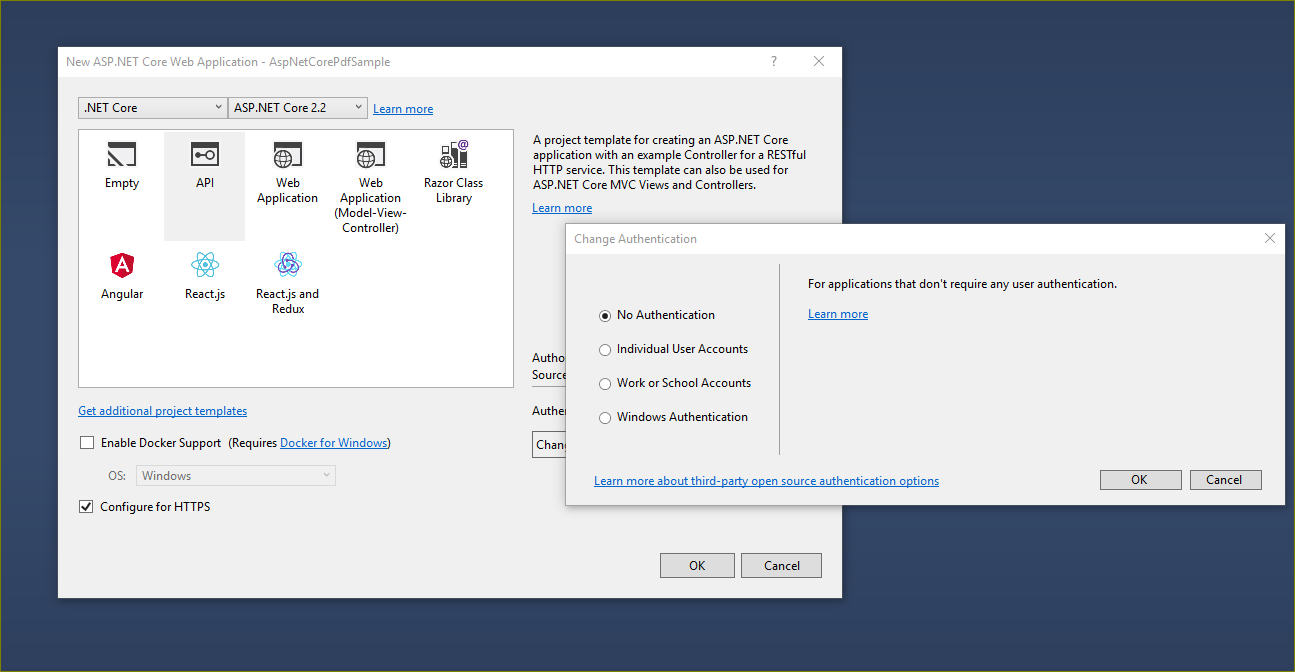
* Basic Project Setup
* Add External Library (DinkToPdf, Razor Light)
* Prepare dummy data for PDF Document
* Design Razor View and ass external CSS (bootstrap)
* Preview the VIEW before generate PDF
* Saving the PDF document in local storage
* Show the PDF document to browser
* Conslusion

Basic Project Setup

First, we will create a new ASP.Net Core 2.2 Web API Project named “AspNetCorePdfSample”



Default configuration:



Delete the default Controller and add a new Controller for PDF functionality:

namespace AspNetCorePdfSample.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class PdfController : ControllerBase

{

[HttpGet("Generate")]

public async Task<IActionResult> Generate()

{

return Ok("API is working fine...!");

}

}

}

Now, we will need to add some external libraries which we need to generate the PD Document.

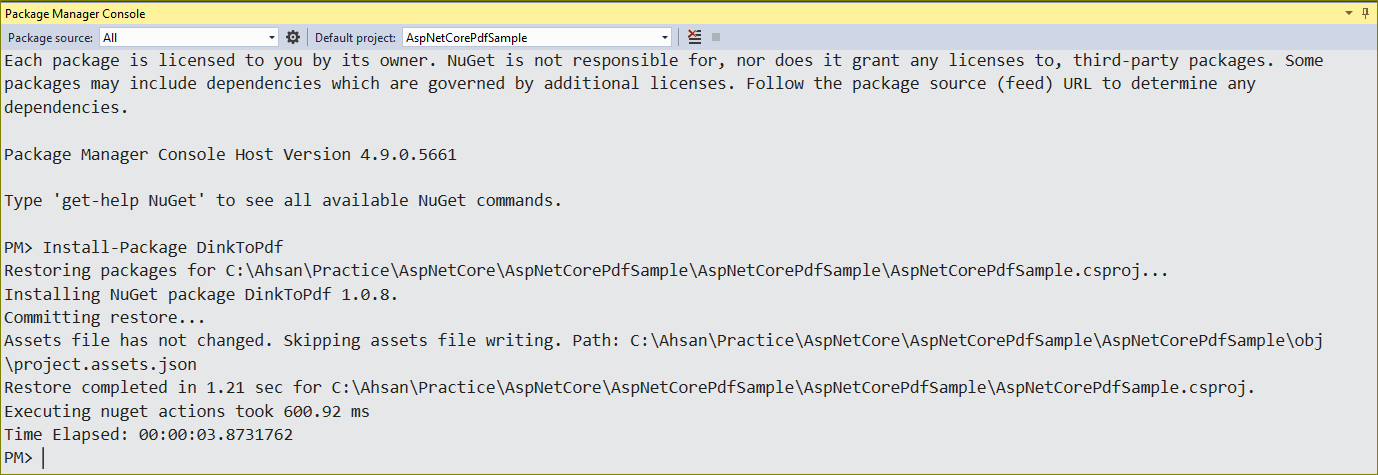
**Add External Library**

**DinkToPDF**

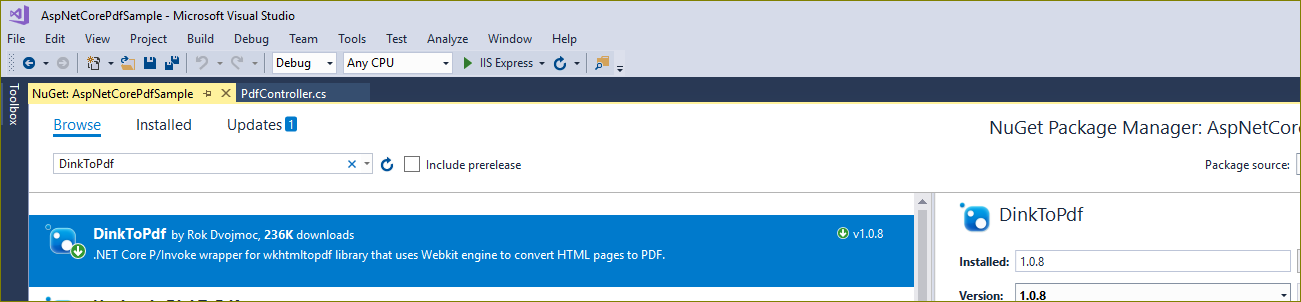
DinkToPDF is a .NET Core P/Invoke wrapper for **wkhtmltopdf library** that uses Webkit engine to convert HTML pages to PDF. The good thing of this library is that we could use our own **CSS** file to design our HTML page and convert it as PDF document.

Let’s install the DinkToPdf library to our project:

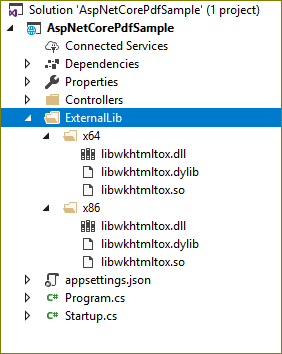
Form the Package manager Console: PM> Install-Package DinkToPdf



Or search for DinkToPdf inside the Nuget Package window:



After installation successful, we need one more step to do before we start using the DinkToPdf Library. We will download the native PDF dlls(**libwkhtmltox) and add both version of files we will add to our project. Create a new Folder named “ExternalLib” and paste downloaded files into it.**



https://github.com/rdvojmoc/DinkToPdf/tree/master/v0.12.4

#### ****Razor Light****

#### https://github.com/toddams/RazorLight

Now we will add another library to our project to design our HTML view and use for generating pdf documents. In this case, will use add “Razor Light”. Razor is a powerful library for templating HTML documents and has lots of features for rendering HTML content. Instead of pulling a big razor library we will use the light version of the Razor library since we will need only the rendering feature to convert that generated HTML content to a PDF document.

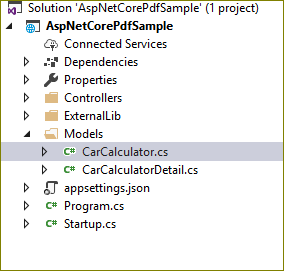
Form the Package manager Console:

PM> Install-Package RazorLight -Version 2.0.0-beta1

**Prepare dummy data for PDF Document**

You can use the real data from your DB or any other data source, but for simplicity we will use simple DTO. Lets Begin…

Firs we will add two classes in the model folder:



CarCalculator.cs

namespace AspNetCorePdfSample.Models

{

public class CarCalculator

{

public string CustomerName { get; set; }

public string CustomerEmail { get; set; }

public List<CarCalculatorDetail> CarDetails { get; set; }

public string CSSLink { get; set; }

}

}

CarCalculatorDetail.cs

namespace AspNetCorePdfSample.Models

{

public class CarCalculatorDetail

{

public int Option { get; set; }

public int VariantId { get; set; }

public string VariantDesc { get; set; }

public string VariantImageString { get; set; }

public string SeriesDesc { get; set; }

public int FinancingProductId { get; set; }

public string FinancingProduct { get; set; }

public string RRP { get; set; }

public int DownPaymentPercentage { get; set; }

public string DownPayment { get; set; }

public int Tenure { get; set; }

public int Mileage { get; set; }

public int CampaignId { get; set; }

public decimal Rate { get; set; }

public string monthlyInstallment { get; set; }

public string LastInstallment { get; set; }

public string GFV { get; set; }

}

}

Now, we will create a static repository for mock data named “CarRepository.cs” and add some dummy data to the newly created repository as follows:

namespace AspNetCorePdfSample.Repository

{

}

**Design Razor View and ass external CSS (bootstrap)**

In this section, we will design our HTML page with Razor template.

First, we will add a new “controller” PreviewController and add an “action“ ShowTemplate as follows

namespace AspNetCorePdfSample.Controllers

{

public class PreviewController : Controller

{

public IActionResult ShowTemplate()

{

var model = CarRepository.GetCarCalculatorData();

return View(model);

}

}

}

Now, we will add the view ~/Views/Preview/ShowTemplate.cshtml as follows

@model AspNetCorePdfSample.Models.CarCalculator

<html>

<head>

<style>

body {

font-family: "Arial";

font-size: 14px;

}

.smallFont {

font-size: 12px;

}

.leftMarin {

margin: 3px !important;

}

.odd {

background-color: #b8d1f3;

}

.even {

background-color: #dae5f4;

}

.imgContainer {

width: 20%;

height: 200px;

max-height: 400px;

}

.imgContainer img {

width: 100%;

}

</style>

<link rel="stylesheet" href="@Model.CSSLink" />

<link href="~/Content/css/bootstrap.css" rel="stylesheet" />

</head>

<body>

<h4>Dear @Model.CustomerName </h4>

<p>

Thank you for your interest in enquiring a Hire Purchase Financing from BMW Group Financial Services. We are pleased to offer the financing quote hereinafter.

</p>

<hr />

<div class="row smallFont">

@foreach (var car in Model.CarDetails)

{

<div class="col-4">

<div class="card">

<div class="card-header">

<h4 class="card-title ">@car.SeriesDesc</h4>

<p class="card-category">@car.VariantDesc </p>

</div>

<div class="card-body">

<div class="row">

<div class="col imgContainer" style="width:20%;">

<img alt="@car.VariantDesc" src="@car.VariantImageString" align=left hspace=12 />

</div>

</div>

<!--<hr />-->

<div class="row even">

<div class="col-7">

<span>RRP :</span>

</div>

<div class="col-5">

<span>@car.RRP </span>

</div>

</div>

<!--<hr />-->

<div class="row odd">

<div class="col-7">

<span>Down Payment :</span>

</div>

<div class="col-5">

<span>@car.DownPayment </span>

</div>

</div>

<!--<hr />-->

<div class="row even">

<div class="col-7">

<span>Down Payment (%) :</span>

</div>

<div class="col-5">

<span>@car.DownPaymentPercentage </span>

</div>

</div>

<!--<hr />-->

<div class="row odd">

<div class="col-7">

<span>Term :</span>

</div>

<div class="col-5">

<span>@car.Tenure </span>

</div>

</div>

<!--<hr />-->

<div class="row even">

<div class="col-7">

<span>Mileage Options :</span>

</div>

<div class="col-5">

<span>@car.Mileage </span>

</div>

</div>

<!--<hr />-->

<div class="row odd">

<div class="col-7">

<span>Financial Product :</span>

</div>

<div class="col-5">

<span>@car.FinancingProduct </span>

</div>

</div>

<!--<hr />-->

<div class="row even">

<div class="col-7">

<span>Monthly Payment :</span>

</div>

<div class="col-5">

<span>@car.monthlyInstallment </span>

</div>

</div>

<!--<hr />-->

<div class="row odd">

<div class="col-7">

<span>Guaranteed Future Value :</span>

</div>

<div class="col-5">

<span>@car.GFV </span>

</div>

</div>

</div>

</div>

</div>

<br />

}

</div>

<br />

<br />

<br />

<div class="row">

<div class="col">

<p>

Marketing advs.

</p>

</div>

</div>

<div class="row">

<div class="col">

<p>

Link to the e-calculator.

</p>

</div>

</div>

</body>

</html>.

Before we run the application we will add our bootstrap.css in the project.

To view this page in the browser we need add MVC routing in our startup.cs file as below:

public Startup(IConfiguration configuration, IHostingEnvironment hostingEnvironment)

{

Configuration = configuration;

\_hostingEnvironment = hostingEnvironment;

}

public IConfiguration Configuration { get; }

private readonly IHostingEnvironment \_hostingEnvironment;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

// app.UseMvc();

app.UseStaticFiles();

// This will add "css" as another valid static content location

app.UseStaticFiles(new StaticFileOptions()

{

FileProvider = new PhysicalFileProvider(

Path.Combine(\_hostingEnvironment.ContentRootPath, @"Content")),

RequestPath = new PathString("/Content")

});

app.UseMvc(routes =>

{

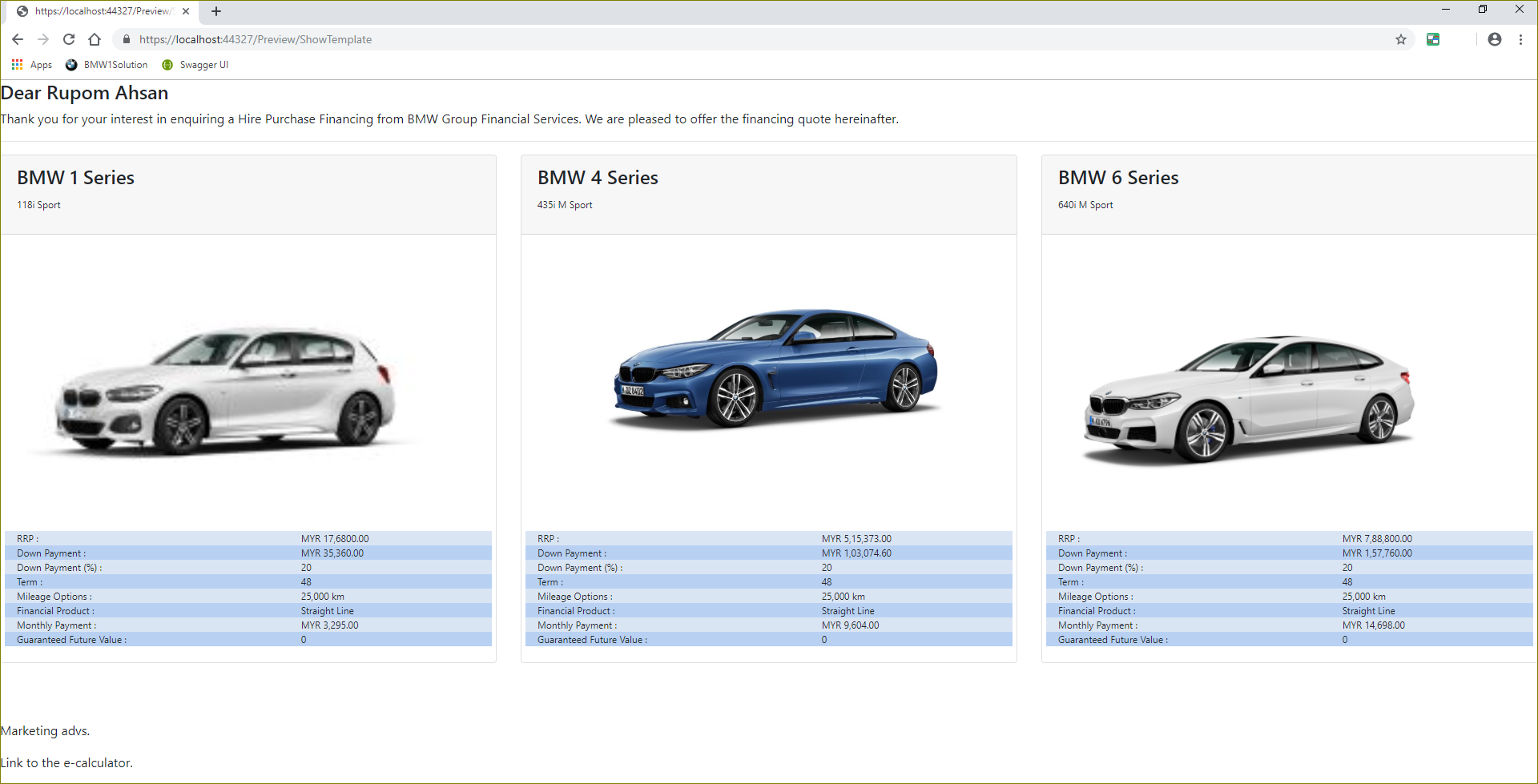
routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");

});

}

Now, If we run the application and browse the following url we should see the below Screen:

URL: <https://localhost:44327/Preview/ShowTemplate>



So now we already have the Design ready for generate the PDF Document.

**Implement PDF Functions**

To implement all things together, we need to do few things.

First, to load the native dll’s for DinkToPdf library, we will create an extension class named “**CustomAssemblyLoadContext**” which will be use to load dependent dlls.

namespace AspNetCorePdfSample.Models

{

public class CustomAssemblyLoadContext : AssemblyLoadContext

{

public IntPtr LoadUnmanagedLibrary(string absolutePath)

{

return LoadUnmanagedDll(absolutePath);

}

protected override IntPtr LoadUnmanagedDll(String unmanagedDllName)

{

return LoadUnmanagedDllFromPath(unmanagedDllName);

}

protected override Assembly Load(AssemblyName assemblyName)

{

throw new NotImplementedException();

}

}

}

Then, we need to modify our Startup.cs in the **ConfigureServices** section, need to add following codes:

services.AddSingleton(typeof(IConverter), new SynchronizedConverter(new PdfTools()));

var processSufix = "x86";

if (Environment.Is64BitProcess && IntPtr.Size == 8)

{

processSufix = "x64";

}

var context = new CustomAssemblyLoadContext();

context.LoadUnmanagedLibrary(Path.Combine(\_hostingEnvironment.ContentRootPath, $"ExternalLib\\{processSufix}\\libwkhtmltox.dll"));

And for Razor service:

services.AddScoped<IRazorLightEngine>(sp =>

{

var engine = new RazorLightEngineBuilder()

.UseFilesystemProject(Path.GetDirectoryName(Assembly.GetEntryAssembly().Location))

.UseMemoryCachingProvider()

.Build();

return engine;

});

Finally, Lets create a service to generate the PDF documents.

namespace AspNetCorePdfSample.Models

{

public interface IPDFService

{ Task<byte[]> GeneratePdf(string contentRootPath, CarCalculator carCalculator);

}

}

And implement the service as below:

namespace AspNetCorePdfSample.Models

{

public class PDFService : IPDFService

{

private readonly IRazorLightEngine \_razorEngine;

private readonly IConverter \_pdfConverter;

public PDFService(IRazorLightEngine razorEngine, IConverter pdfConverter)

{

\_razorEngine = razorEngine;

\_pdfConverter = pdfConverter;

}

public async Task<byte[]> GeneratePdf(string contentRootPath, CarCalculator carCalculator)

{

// var model = Data.CarRepository.GetCars();

carCalculator.CSSLink = Path.Combine(contentRootPath, $"Content\\css\\bootstrap.css");

var templatePath = Path.Combine(contentRootPath, $"ReportFiles\\Templates\\Ecalculator.cshtml");

string fileName = "CarComparison\_" + DateTime.Now.ToString("yyyyMMddHHmmssfff") + ".pdf";

string filePath = Path.Combine(contentRootPath, "ReportTemplates", "PdfFiles", "ECalculator", fileName);

string template = await \_razorEngine.CompileRenderAsync(templatePath, carCalculator);

var globalSettings = new GlobalSettings

{

ColorMode = ColorMode.Color,

Orientation = Orientation.Portrait,

PaperSize = PaperKind.A4,

Margins = new MarginSettings() { Top = 10, Bottom = 10, Left = 10, Right = 10 },

DocumentTitle = "Car Calculator Document"

};

var objectSettings = new ObjectSettings

{

PagesCount = true,

HtmlContent = template,

WebSettings = { DefaultEncoding = "utf-8" },

HeaderSettings = { FontName = "Arial", FontSize = 12, Line = true, Center = "Sample of Car Calculator PDF document " },

FooterSettings = { FontName = "Arial", FontSize = 12, Line = true, Right = "Page [page] of [toPage]" }

};

var pdf = new HtmlToPdfDocument()

{

GlobalSettings = globalSettings,

Objects = { objectSettings }

};

byte[] file = \_pdfConverter.Convert(pdf);

return file;

// \_pdfConverter.Convert(pdf);

// return filePath;

}

}

}

Now, We will need to add this service to our Startup.cs class so we have access to it across the application.

services.AddScoped<IPDFService, PDFService>();

Now its time to modify our PdfController/ Generate() function to see the actual result.

namespace AspNetCorePdfSample.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class PdfController : ControllerBase

{

private readonly IPDFService \_pdfService;

private readonly IHostingEnvironment \_hostingEnvironment;

public PdfController(IPDFService pdfService, IHostingEnvironment hostingEnvironment)

{

\_pdfService = pdfService;

\_hostingEnvironment = hostingEnvironment;

}

[HttpGet("Generate")]

public async Task<IActionResult> Generate()

{

var file = await \_pdfService.GeneratePdf(\_hostingEnvironment.ContentRootPath);

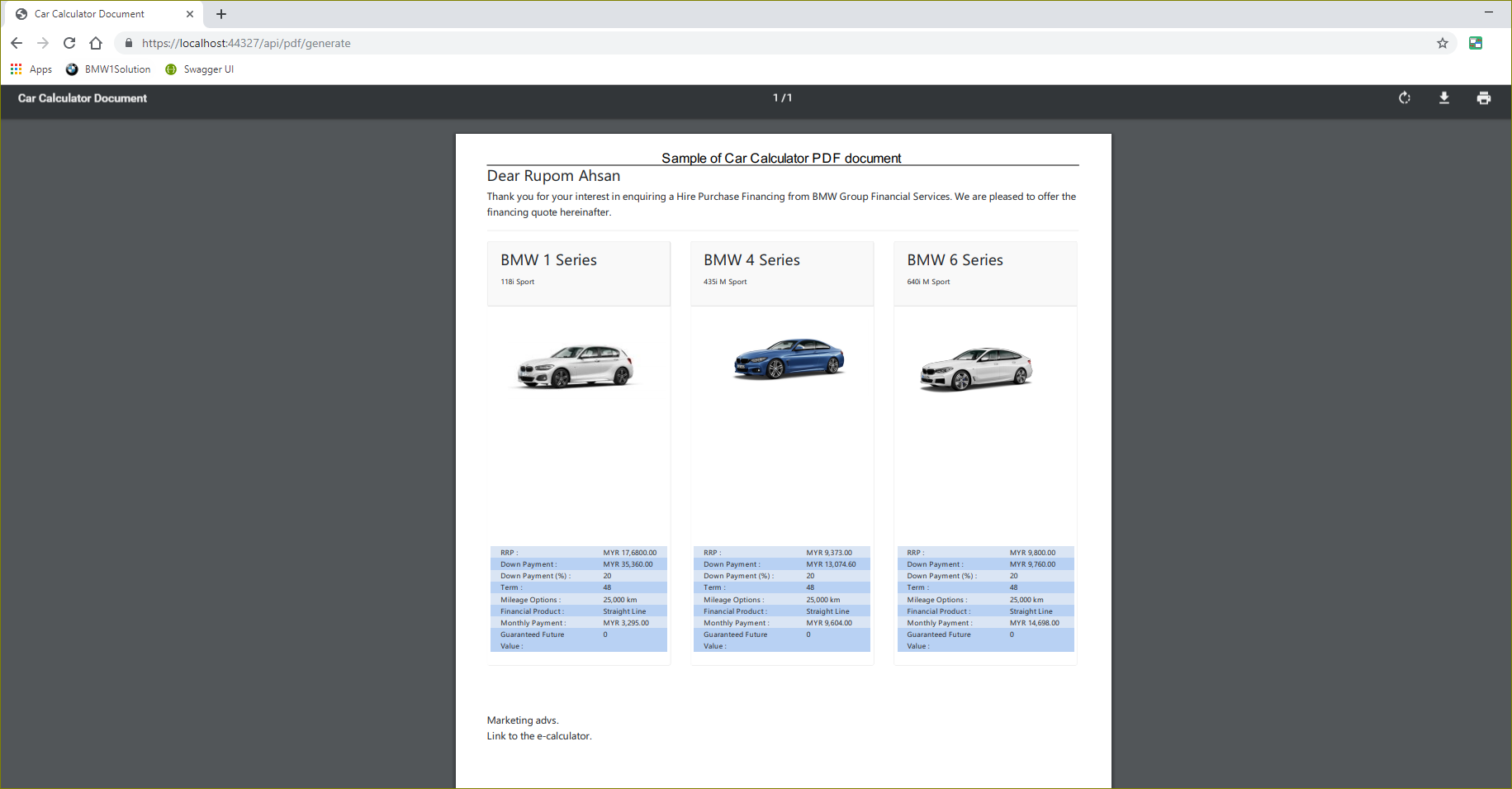
return File(file, "application/pdf");

}

}

}

Now we should see the below result after run the project:



Conclusion:

I hope you enjoyed the article.

Now for deployment you may need to add the below changes in you project settings:

<MvcRazorExcludeRefAssembliesFromPublish>false</MvcRazorExcludeRefAssembliesFromPublish>