MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
FEB 1, 2021 3:00 AM PST

How to create PDF documents in ASP.NET Core 5

Take advantage of the DinkToPdf and wkhtmltopdf libraries to generate PDF documents from HTML templates in ASP.NET Core 5.

When working on web applications in ASP.NET Core 5 or ASP.NET Core MVC 5, you will often need to generate and display PDF documents to the user. Such documents might include invoices, salary slips, reports, etc.

This article presents a discussion of how we can use the DinkToPdf library to generate PDF documents in ASP.NET Core 5 applications. DinkToPdf is a .NET Core P/Invoke wrapper to the wkhtmltopdf library, which renders HTML to PDF (and other formats) using the Qt WebKit rendering engine.

[Also on InfoWorld: The most valuable software developer skills to get hired now]

To work with the code examples provided in this article, you should have Visual Studio 2019 installed in your system. If you don't already have a copy, you can download Visual Studio 2019 here.

Create an ASP.NET Core MVC 5 project in Visual Studio 2019

First off, let's create an ASP.Net Core project in Visual Studio 2019. Following these steps should create a new ASP.NET Core MVC 5 project in Visual Studio 2019.

1. Launch the Visual Studio IDE.
2. Click on “Create new project.”
3. In the “Create new project” window, select “ASP.NET Core Web App (Model-View-Controller)” from the list of templates displayed.
4. Click Next.
5. In the “Configure your new project” window, specify the name and location for the new project.
6. Optionally check the “Place solution and project in the same directory” check box, depending on your preferences.
7. Click Next.
8. In the “Additional Information” window shown next, select .NET 5.0 as the target framework from the drop-down list at the top. Leave the “Authentication Type” set to None (default).
9. Ensure that the check boxes “Enable Docker,” “Configure for HTTPS,” and “Enable Razor runtime compilation” are unchecked as we won’t be using any of those features here.
10. Click Create.

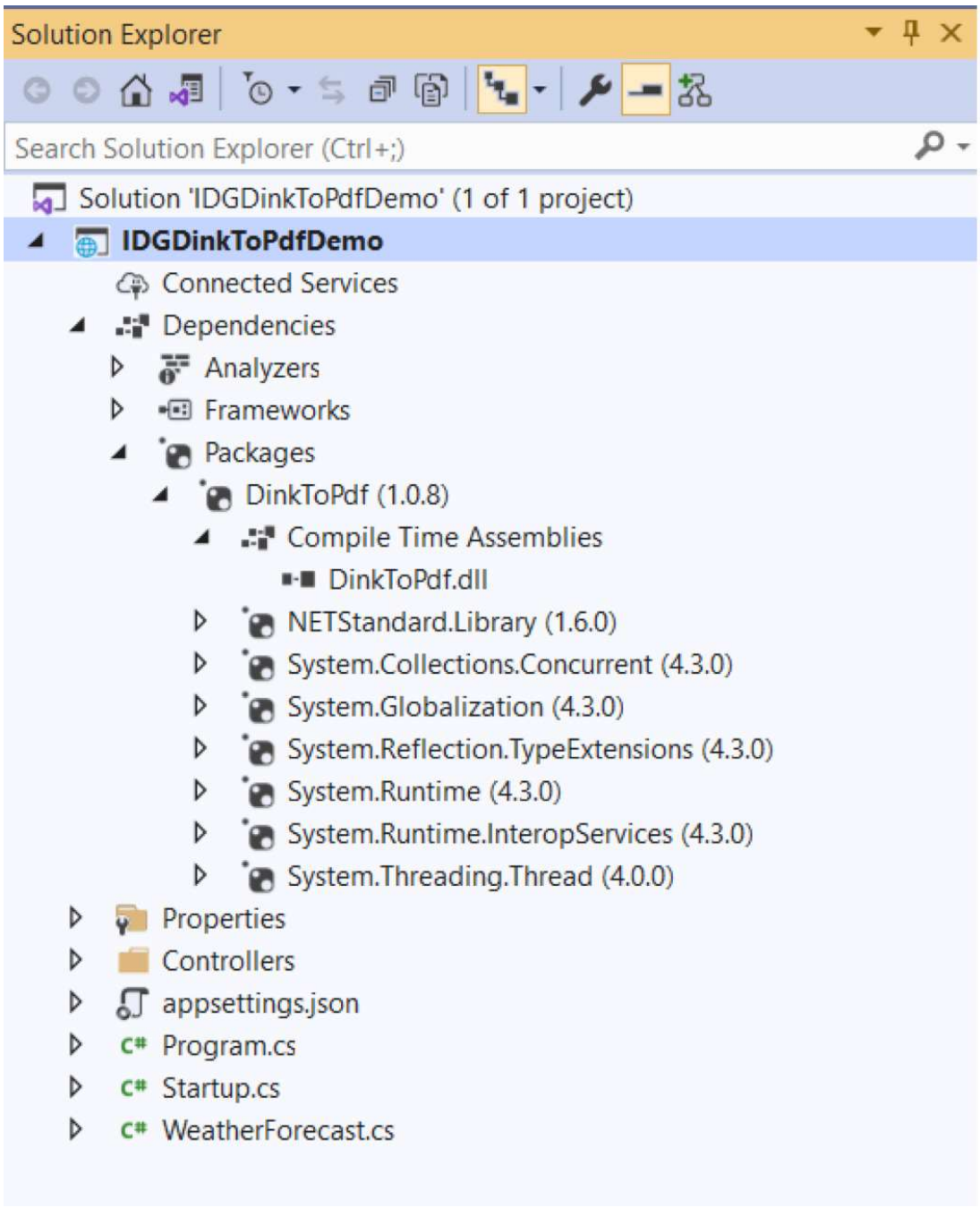
Following these steps will create a new ASP.NET Core MVC 5.0 project. We’ll use this project in the subsequent sections in this article.

Install the DinkToPdf NuGet package

The DinkToPdf library is available as a NuGet package. To get started working with the DinkToPdf library, you must install it from NuGet. You can either install it via the NuGet Package Manager or by using the following command at the NuGet Package Manager Console window.

```
Install-Package DinkToPdf
```

Once you've installed this library, you can verify the installation by locating the DinkToPdf.dll library as shown in Figure 1 below.



IDG

Figure 1. DinkToPdf NuGet package installation.

Register the DinkToPdf library with the IoC container

Now that the DinkToPdf library has been installed on your project, you must register the library with the built-in IoC container. To do this, you can write the following code in the ConfigureServices method.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton(typeof(IConverter),
    new SynchronizedConverter(new PdfTools()));
    services.AddControllers();
}
```

To make the above code snippet work, you should include the following libraries in the Startup.cs file:

```
using DinkToPdf;
using DinkToPdf.Contracts;
```

Create a ReportService class in ASP.NET Core 5

Now that DinkToPdf has been configured, the next step is to create the ReportService class. This class will contain the logic for generating the PDF document. We'll first create an interface named IReportService that will be extended by the ReportService class. Create a file called IReportService.cs and enter the following code.

```
public interface IReportService
{
  public byte[] GeneratePdfReport();
}
```

The ReportService class extends the IReportService interface and implements the GeneratePdfReport method as shown below.

```
public class ReportService: IReportService
{
   private readonly IConverter _converter;
   public ReportService(IConverter converter)
   {
       _converter = converter;
   }
   public byte[] GeneratePdfReport()
   {
      throw new NotImplementedException();
   }
}
```

Note how the IConverter instance has been injected using constructor injection.

Implement a GeneratePdfReport method in ASP.NET Core 5

The following code snippet shows the GeneratePdfReport method that contains all of the necessary code for generating the report.

```
public byte[] GeneratePdfReport()
{
   var html = $@"
   <!DOCTYPE html>
   <html lang=""en"">
   <head>
       This is the header of this document.
   </head>
  <body>
  <h1>This is the heading for demonstration purposes only.</h1>
  <p>This is a line of text for demonstration purposes only.</p>
  </body>
  </html>
  ";
GlobalSettings globalSettings = new GlobalSettings();
globalSettings.ColorMode = ColorMode.Color;
globalSettings.Orientation = Orientation.Portrait;
globalSettings.PaperSize = PaperKind.A4;
 globalSettings.Margins = new MarginSettings { Top = 25, Bottom = 25 };
 ObjectSettings objectSettings = new ObjectSettings();
 objectSettings.PagesCount = true;
 objectSettings.HtmlContent = html;
 WebSettings webSettings = new WebSettings();
 webSettings.DefaultEncoding = "utf-8";
 HeaderSettings headerSettings = new HeaderSettings();
 headerSettings.FontSize = 15;
 headerSettings.FontName = "Ariel";
 headerSettings.Right = "Page [page] of [toPage]";
 headerSettings.Line = true;
 FooterSettings footerSettings = new FooterSettings();
 footerSettings.FontSize = 12;
 footerSettings.FontName = "Ariel";
 footerSettings.Center = "This is for demonstration purposes only.";
 footerSettings.Line = true;
 objectSettings.HeaderSettings = headerSettings;
 objectSettings.FooterSettings = footerSettings;
 objectSettings.WebSettings = webSettings;
 HtmlToPdfDocument htmlToPdfDocument = new HtmlToPdfDocument()
 {
      GlobalSettings = globalSettings,
      Objects = { objectSettings },
 };
  return _converter.Convert(htmlToPdfDocument);
}
```

Create a ReportController class in ASP.NET Core 5

While the GeneratePdfReport method of the ReportService class builds the report data as a byte array, the actual report file is created in the ReportController class as shown in the code listing given below.

```
[Route("api/[controller]")]
[ApiController]
public class ReportController : ControllerBase
    {
        private readonly IReportService _reportService;
        public ReportController(IReportService reportService)
        {
            _reportService = reportService;
        }
        [HttpGet]
        public IActionResult Get()
        {
            var pdfFile = _reportService.GenerateReport();
            return File(pdfFile,
            "application/octet-stream", "SimplePdf.pdf");
        }
    }
```

When you run the application and hit the HttpGet endpoint of the ReportController class, a report file will be created and downloaded in your computer. The generated report from our example is shown in Figure 2 below.

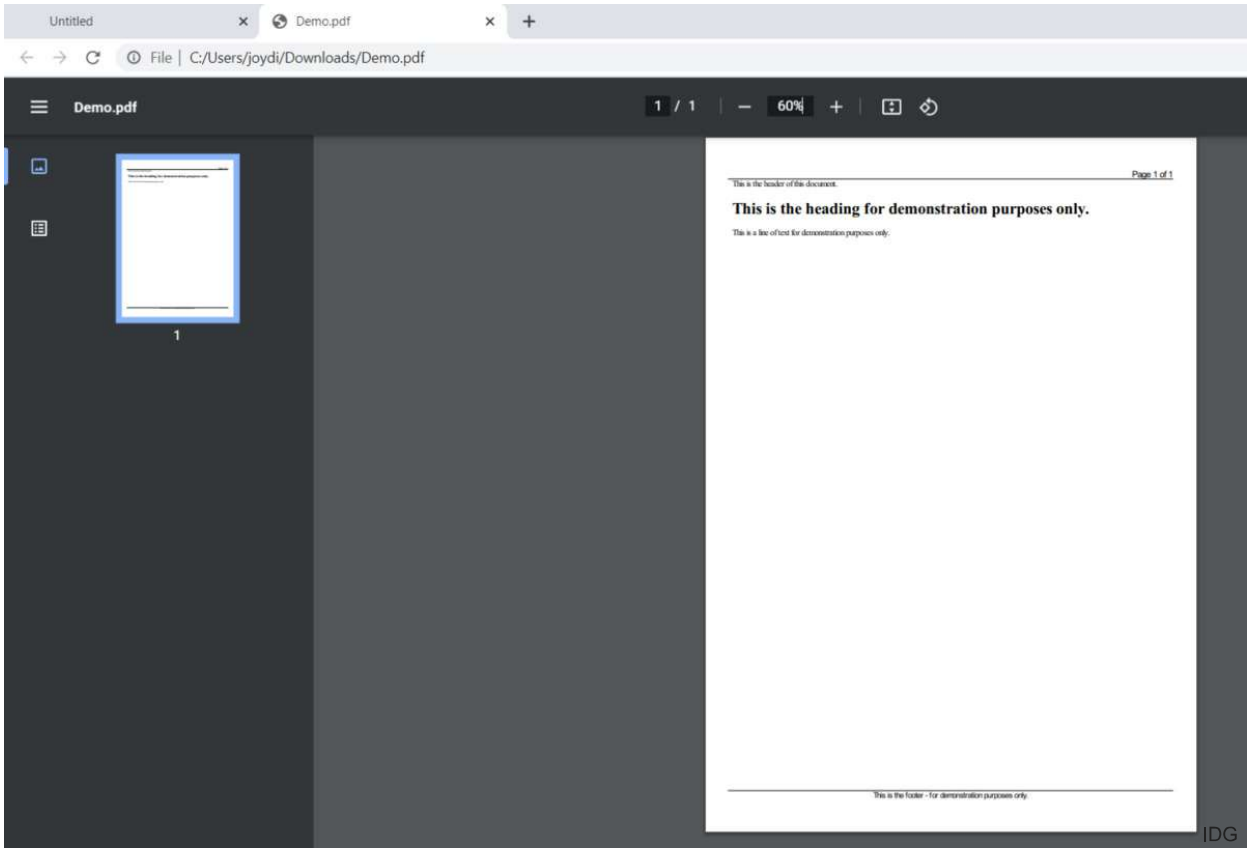


Figure 2. Our example PDF report.

There is no built-in reporting framework in ASP.NET Core 5 or ASP.NET Core MVC 5, so if we want to generate PDF documents, we'll need to take advantage of a third-party library. DinkToPdf is one of many such libraries available.

DinkToPdf is a .NET Core wrapper for wkhtmltopdf that can be used to convert an HTML template to a PDF document. You can learn more about DinkToPdf on GitHub.

How to do more in ASP.NET Core:

- How to use immutable objects in ASP.NET Core MVC 5
- Dependency injection best practices for ASP.NET Core MVC 5
- How to use security headers in ASP.NET Core MVC 5
- How to handle unknown actions in ASP.NET Core 5 MVC
- How to overload action methods in ASP.NET Core 5 MVC
- How to use multiple implementations of an interface in ASP.NET Core
- How to use IHttpClientFactory in ASP.NET Core

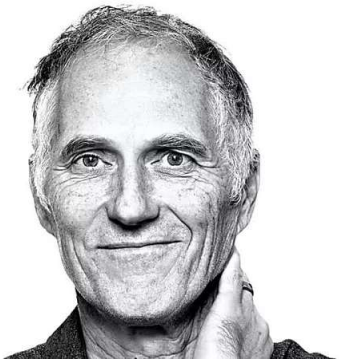
- How to use the ProblemDetails middleware in ASP.NET Core
- How to create route constraints in ASP.NET Core
- How to manage user secrets in ASP.NET Core
- How to build gRPC applications in ASP.NET Core
- How to redirect a request in ASP.NET Core
- How to use attribute routing in ASP.NET Core
- How to pass parameters to action methods in ASP.NET Core MVC
- How to use API Analyzers in ASP.NET Core
- How to use route data tokens in ASP.NET Core
- How to use API versioning in ASP.NET Core
- How to use Data Transfer Objects in ASP.NET Core 3.1
- How to handle 404 errors in ASP.NET Core MVC
- How to use dependency injection in action filters in ASP.NET Core 3.1
- How to use the options pattern in ASP.NET Core
- How to use endpoint routing in ASP.NET Core 3.0 MVC
- How to export data to Excel in ASP.NET Core 3.0

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow

YOU MAY ALSO LIKE

Recommended by



Tim O'Reilly: the golden age of the programmer is over

6 ways Alibaba Cloud challenges AWS, Azure, and GCP

Prisma ORM for Node.js is ready for production

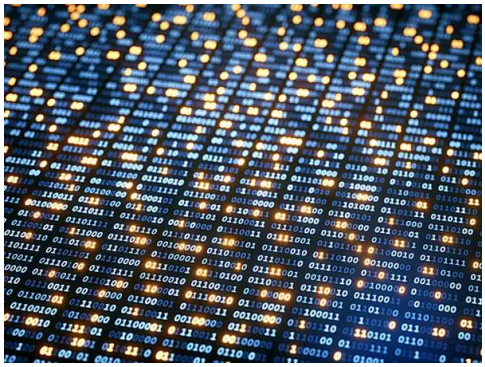
2 common cloud computing predictions for 2021 are wrong

Red Hat Enterprise Linux takes aim at edge computing

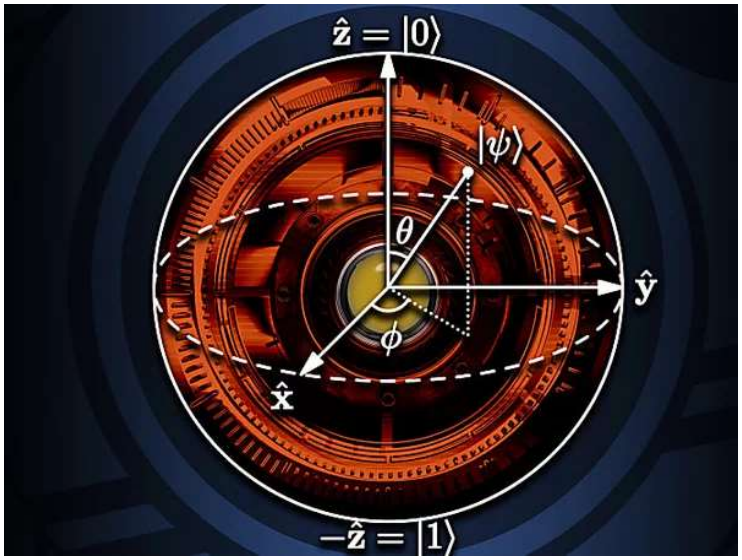
10 top-notch libraries for C++ programming

Edge computing archetypes are emerging,

3 enterprise AI success stories



What is a computational storage drive? Much-



Amazon Braket: Get started with quantum computing

The 24 highest paying developer roles in 2020