Web PDF viewer implemented as single page web application using AngularJS and Web API

Written by Apitron Documentation Team

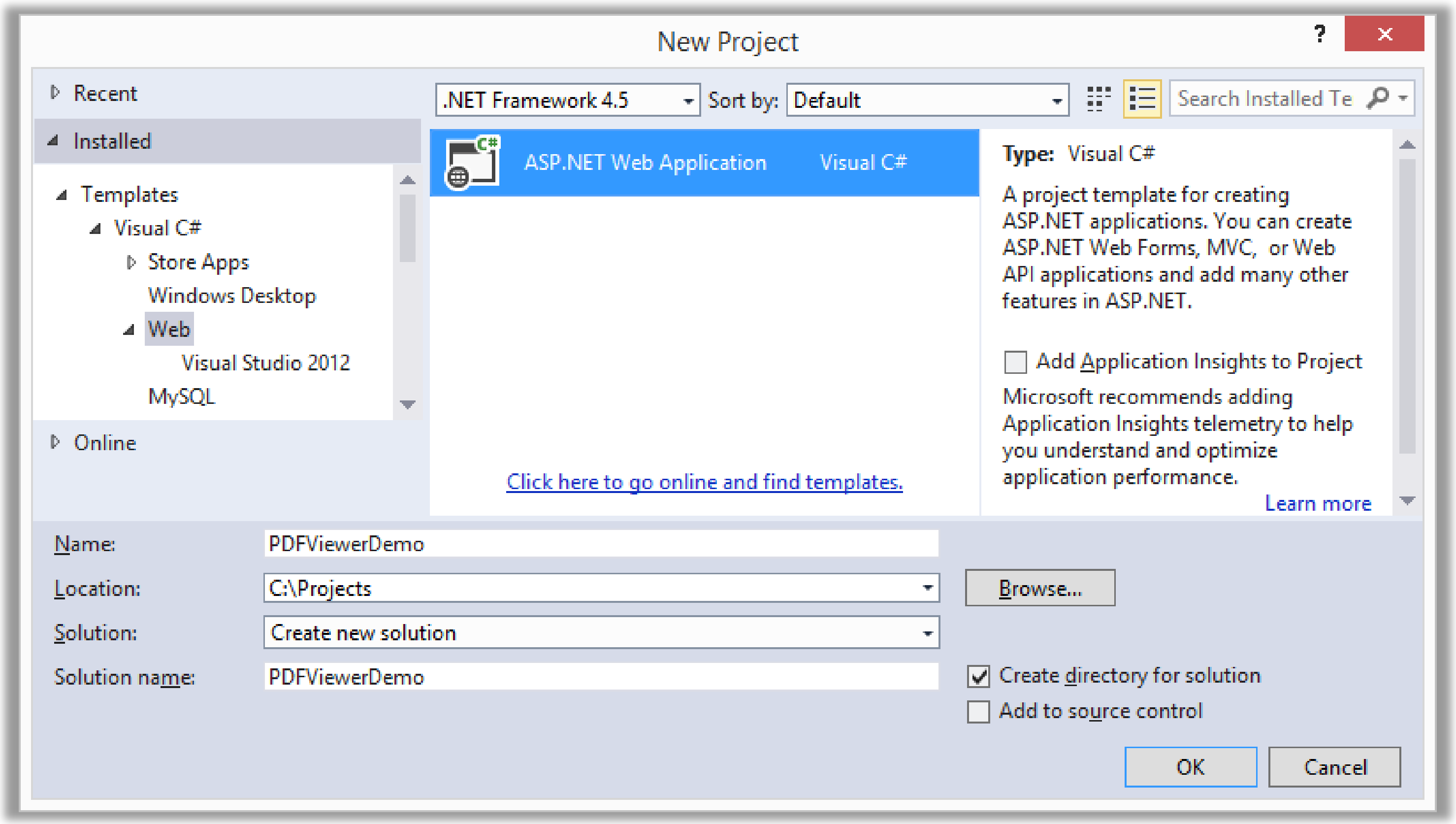
Introduction

There's a trend to implement web applications as so-called *Single Page Applications* or *SPA* for short. These are apps that do most of their job in asynchronous and user-responsive way providing an experience similar to desktop apps, avoiding the need for page reloading.

In this article, we'll show how to create a simple PDF viewer implemented as single page app using ASP.NET Web API for the server side and AngularJS for the client side. We've also used a bit of CSS to make it more interesting.

Creating the project

Open Visual Studio and select File -> New -> Project... -> Web -> ASP .NET Web Application as shown in the illustration below:



.NET Framework 4.5 Sort by: Default

ASP.NET Web Application Visual C#

Click here to go online and find templates.

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

☐ Add Application Insights to Project

Microsoft recommends adding Application Insights telemetry to help you understand and optimize application performance. Learn more

Name: PDFViewerDemo

Location: C:\Projects

Solution: Create new solution

Solution name: PDFViewerDemo

☑ Create directory for solution

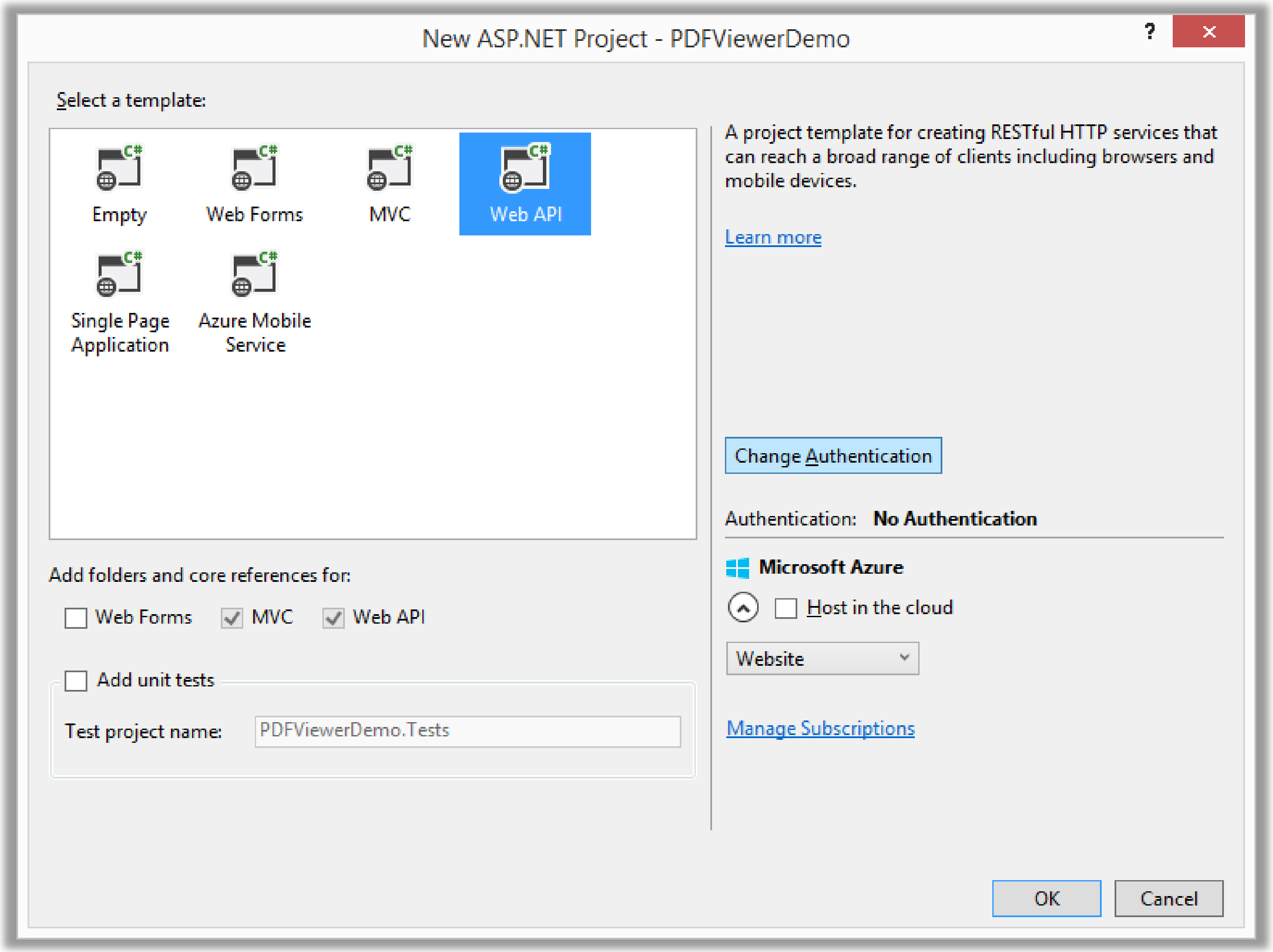
☐ Add to source control

Pic. 1 Creating new project

Click *OK* when you’re ready, and you’ll be redirected to template selection wizard.

Template selection

We'll use the Web API template as we are going to create a very simple UI based exclusively on HTML / CSS and AngularJS. In this example we didn't use any authentication, so turned it off by selecting the *No Authentication* option.


New ASP.NET Project - PDFViewerDemo

Select a template:

Empty | Web Forms | MVC | Web API | Single Page Application | Azure Mobile Service

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Learn more

Change Authentication

Authentication: **No Authentication**

Microsoft Azure

☐ Host in the cloud

Website

Manage Subscriptions

Add folders and core references for:

☐ Web Forms ✓ MVC ✓ Web API

☐ Add unit tests

Test project name: PDFViewerDemo.Tests

OK Cancel

Pic. 2 Finalizing project creation

Click *OK* when you're ready to finish the project creation.

Adding packages

Install **AngularJS core** and **Apitron PDF Rasterizer for .NET** packages via NuGet package manager.

Server side code

The RenderingController, shown below, returns document info and renders pages by request.

```
public class RenderingController : ApiController
{
    private static readonly string filePath;
    static RenderingController()
    {
        filePath = HttpContext.Current.Server.MapPath("~/App_Data/Apitron_Pdf_Kit_in_Action.pdf");
    }

    /// <summary>
    /// Gets the rendered page as encoded image.
    /// </summary>
    /// <param name="id">Page index, zero-based.</param>
    /// <returns>String containing encoded image, or null if the call fails.</returns>
    public string GetRenderedPage(int id)
    {
        using (Document doc = new Document(File.OpenRead(filePath)))
        {
            if (id >= 0 && doc.Pages.Count > id)
            {
                Bitmap bm = doc.Pages[id].Render(new Resolution(72, 72), new RenderingSettings());

                if (bm != null)
                {
                    using (MemoryStream ms = new MemoryStream())
                    {
                        bm.Save(ms, ImageFormat.Png);
                        return string.Format("data:image/png;base64,{0}",
                                 Convert.ToBase64String(ms.ToArray()));
                    }
                }
            }
        }
        return null;
    }

    /// <summary> Returns document info. </summary>
    /// <returns>Valid document info object, or null if the call fails.</returns>
    public DocumentInfo GetFileInfo()
    {
        try
        {
            using (Document doc = new Document(File.OpenRead(filePath)))
            {
                return new DocumentInfo() {PageCount = doc.Pages.Count};
            }
        }
        catch (Exception e)
        {
            Debug.WriteLine(e.Message);
            return null;
        }
    }
}
```

In order to properly map requests to our controller, change the default route registered in WebApiConfig class as folllows:

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

UI template

The html provided below is located in the root of the project and set as startup page for it. It references the CSS stylesheet and scripts needed to run the AngularJS app.

```
<!DOCTYPE html>
<html ng-app="mainApp" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script src="/Scripts/angular.min.js"></script>
    <script src="/Scripts/app/app.js"></script>
    <script src="/Scripts/app/renderingController.js"></script>
    <link rel="stylesheet" type="text/css" href="Content/styles.css">
</head>
    <body ng-controller="pageByPageRenderingController" ng-init="setup()">
            <div class="navbarContainer">
                <ul>
                    <li ng-click="loadFile()">Load file</li>
                    <li ng-click="renderPrev()">Prev</li>
                    <li ng-click="renderNext()">Next</li>
                    <li onclick="alert('Apitron PDF Rasterizer usage sample,
                       implemented using WebAPI and AngularJS.')">About</li>
                </ul>
            </div>
            <div class="viewArea">
                <span id="aligner">{{errorText}}</span>
                <img ng-src="/images/loading.gif" class="{{progressBarClass}}" />
                <img ng-src="{{imageSource}}" class="{{pageViewClass}} shadow" />
            </div>
    </body>
</html>
```

AngularJS part

This part consists of *app.js* and *renderingController.js* files located in “/Scripts/app” project folder.

app.js - contains the code needed to configure our app’s module that depends on *renderingController* module that performs the actual interaction with our Web API methods.

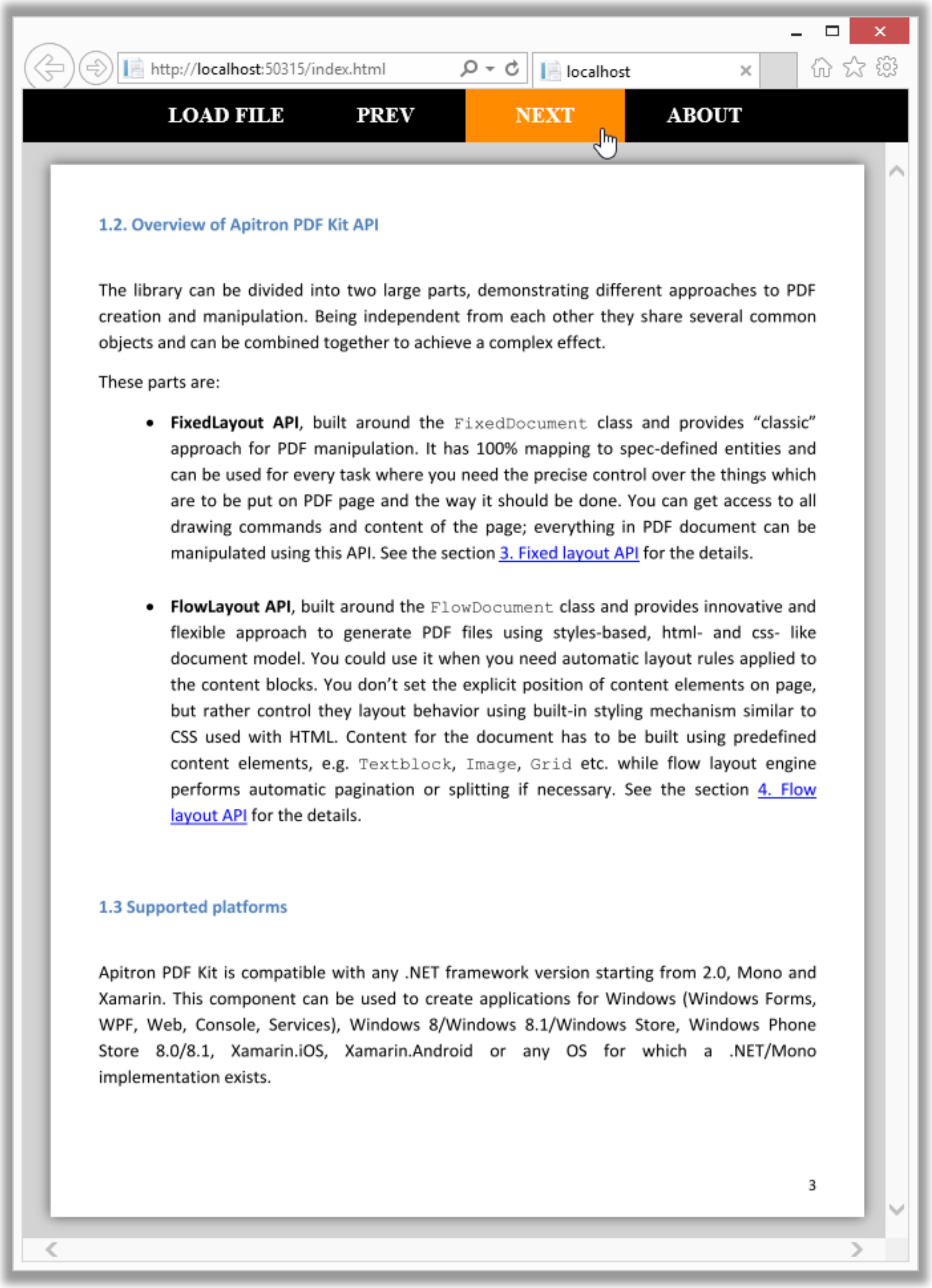
```
var mainApp = angular.module('mainApp', ['renderingController']);
```

renderingController.js - contains the code for *pageByPageRenderingController* object which implements the methods referenced in UI template, e.g. *setup(), loadFile()* etc. The complete code for this controller can be found in corresponding sample project and downloaded from our GitHub repo.

Results

We kept the app simple, and as it can be seen from the RenderingController's code, we didn't use any caching and implemented page rendering using the simplest way: open the file and render the requested page for every request.

The image shown below demonstrates the resulting PDF viewer single page application:



1.2. Overview of Apitron PDF Kit API

The library can be divided into two large parts, demonstrating different approaches to PDF creation and manipulation. Being independent from each other they share several common objects and can be combined together to achieve a complex effect.

These parts are:

- **FixedLayout API**, built around the `FixedDocument` class and provides "classic" approach for PDF manipulation. It has 100% mapping to spec-defined entities and can be used for every task where you need the precise control over the things which are to be put on PDF page and the way it should be done. You can get access to all drawing commands and content of the page; everything in PDF document can be manipulated using this API. See the section 3. Fixed layout API for the details.

- **FlowLayout API**, built around the `FlowDocument` class and provides innovative and flexible approach to generate PDF files using styles-based, html- and css- like document model. You could use it when you need automatic layout rules applied to the content blocks. You don't set the explicit position of content elements on page, but rather control they layout behavior using built-in styling mechanism similar to CSS used with HTML. Content for the document has to be built using predefined content elements, e.g. `Textblock`, `Image`, `Grid` etc. while flow layout engine performs automatic pagination or splitting if necessary. See the section 4. Flow layout API for the details.

1.3 Supported platforms

Apitron PDF Kit is compatible with any .NET framework version starting from 2.0, Mono and Xamarin. This component can be used to create applications for Windows (Windows Forms, WPF, Web, Console, Services), Windows 8/Windows 8.1/Windows Store, Windows Phone Store 8.0/8.1, Xamarin.iOS, Xamarin.Android or any OS for which a .NET/Mono implementation exists.

3

Pic. 3 PDF Viewer single page app

Conclusion

If you plan to implement your web app as a single page app and need to integrate PDF preview or PDF processing capabilities, then Apitron's .NET PDF components such as Apitron PDF Rasterizer or Apitron PDF Kit can be easily employed to achieve the best results.

In this article we have shown how to implement basic pdf preview, but it's also possible to implement full featured pdf viewer with loading of the document bookmarks, support for internal navigation, and so on. You can also read and create document fields, create PDF forms, extract text, sign documents and do whatever you want with them using our PDF components.

If you have any questions regarding our products, just let us know and we'll be happy to answer them.