**ASP.NET Core MVC ile Excel Dışa Aktarma (ClosedXML) Part 119.6.2020**

Bu makalede Asp.Net web uygulamalarında excel import ve export işlemlerine ihtiyaç duyduğumuz durumlar için kullanılabilecek ClosedXML kütüphanesinden bahsetmek istiyorum. Bu kütüphane ile türkçe kaynaklarda anlatılan ve bilinen yöntemlerden çok daha farklı şekilde excel export işlemleri ve import işlemleri yapabilirsiniz. Excelin bir web sayfası formatında değil de çalışma kitabı formatında dışa aktarılmasını sağlayabilirsiniz.

Bu konuyu bir makale serisi şeklinde basit örneklerden başlayarak daha gelişmiş ve daha karmaşık örneklere doğru ilerleyerek anlatacağım.

ClosedXML kütüphanesini gerek masaüstü geliştirilen uygulamalarda, gerek Asp.Net WebForm, Asp.Net MVC, Asp.Net Core uygulamarında kullanabilirsiniz. Bu harika kütüphane ile aklınıza gelebilecek her türlü excel dosyasını kolay bir şekilde oluşturabilirsiniz.

ClosedXML github repo linki; <https://github.com/ClosedXML/ClosedXML>

Serinin bu ilk makalesinde basitçe bir excel export işlemi yapacağız ve JavaScript ile download edeceğiz.

Ve yine bu makalede exceldeki hücrelere değer ve formül atamayı, hücre birleştirmeyi, hücre biçimlendirmeyi ve hizalamayı, hücre arka plan ve yazı rengini, yazı font ve boyutlarını oluşturmayı, satır yüksekliği, sütun genişliği belirtmeyi, hücrelere kenarlık vermeyi ve hücrede metni kaydır işlemlerini uygulamayı anlatacağım.

Öncelikle oluşturduğunuz projeye ClosedXML'i nuget package üzerinden yüklemelisiniz, bunun için Package Manager Console'da Install-Package ClosedXML ﻿komutunu çalıştırabilir veya Manage Nuget Packages üzerinden ClosedXML yazarak aratıp install diyebilirsiniz.

Kütüphaneyi projemize ekledikten sonra işe koyulabiliriz. Ben bu makale için Asp.Net Core MVC ile bir örnek proje oluşturdum ve HomeController'a aşağıdaki gibi bir method tanımladım. Makalenin tamamını okumak istemeyenler en alta inerek tüm kod parçasının olduğu bölüme bakabilir :)

[HttpGet]

[Route("export/simple")]

public IActionResult ExportSimpleExcel() {

    return Ok();

}

Burada bir get methodu oluşturdum ve /export/simple olarak route belirttim.

Şimdi bir excel çalışma kitabı oluşturup içerisine bir sayfa ekleyelim.

[HttpGet]

[Route("export/simple")]

public IActionResult ExportSimpleExcel() {

    byte[] excelFile;

    // Excel dosyası (excel çalışma kitabı) oluşturuyoruz

using (var workbook = new XLWorkbook()) {

// Çalışma kitabına bir sayfa ekliyoruz, sayfa ismini istediğimiz gibi verebiliriz

var worksheet = workbook.Worksheets.Add("Sayfa 1");

}

    return Ok(excelFile);

}

Ardından A1 hücresine Hello World klişesini yazalım

[HttpGet]

[Route("export/simple")]

public IActionResult ExportSimpleExcel() {

    byte[] excelFile;

    // Excel dosyası (excel çalışma kitabı) oluşturuyoruz

using (var workbook = new XLWorkbook()) {

// Çalışma kitabına bir sayfa ekliyoruz, sayfa ismini istediğimiz gibi verebiliriz

var worksheet = workbook.Worksheets.Add("Sayfa 1");

         // A1 hücresine değer atıyoruz

worksheet.Cell("A1").Value = "Hello world";

}

    return Ok(excelFile);

}

Burada worksheet.Cell metodu string olarak doğrudan hücre adını alabiliyor, ardından Value özelliğine veri girişini sağlıyoruz. Bir başka veri girişi sağlama yöntemi de SetValue metodu kullanmaktır. ClosedXML kütüphanesinde bir hücreye herhangi bir işlem uygularken doğrudan ilgili özelliğe değer ataması da yapılabilir veya o özellik için sağlanan set methodları da kullanılabilir.

Ek olarak Cell metodu satır indeksi ve sütun indeksi olarak da parametre alabiliyor. Bunu da aşağıdaki gibi örnekleyelim;

[HttpGet]

[Route("export/simple")]

public IActionResult ExportSimpleExcel() {

    byte[] excelFile;

    // Excel dosyası (excel çalışma kitabı) oluşturuyoruz

using (var workbook = new XLWorkbook()) {

// Çalışma kitabına bir sayfa ekliyoruz, sayfa ismini istediğimiz gibi verebiliriz

var worksheet = workbook.Worksheets.Add("Sayfa 1");

         // A1 hücresine değer atıyoruz

worksheet.Cell("A1").Value = "Hello world";

// Hücrelere alternatif olarak bu şekilde de değer atanabilir, ilk parametre satır index, ikinci parametre sütun index

worksheet.Cell(1, 3).SetValue(100); // C1

worksheet.Cell(2, 3).SetValue("200"); // C2

}

    return Ok(excelFile);

}

Gördüğünüz gibi SetValue methodu ile değer ataması sağladık ve satır sütun indekslerini kullanarak hücreyi belirledik.   
  
Şimdi de hücreleri biçimlendirelim ve formatlayalım, ardından bir hücreye de formül verelim.

[HttpGet]

[Route("export/simple")]

public IActionResult ExportSimpleExcel() {

    byte[] excelFile;

    // Excel dosyası (excel çalışma kitabı) oluşturuyoruz

using (var workbook = new XLWorkbook()) {

// Çalışma kitabına bir sayfa ekliyoruz, sayfa ismini istediğimiz gibi verebiliriz

var worksheet = workbook.Worksheets.Add("Sayfa 1");

         // A1 hücresine değer atıyoruz

worksheet.Cell("A1").Value = "Hello world";

// Hücrelere alternatif olarak bu şekilde de değer atanabilir, ilk parametre satır index, ikinci parametre sütun index

worksheet.Cell(1, 3).SetValue(100); // C1

worksheet.Cell(2, 3).SetValue("200"); // C2

         // Birden fazla hücreye tek seferde bu şekilde de veri tipi verilebilir, C sütunu ilk 4 satır

worksheet.Range(worksheet.Cell(1, 3), worksheet.Cell(4, 3)).DataType = XLDataType.Number;

// Hücreyi biçimlendiriyoruz, binlik ayraçları ve virgülden sonraki hane sayısını belirtmiş olduk, C1

worksheet.Cell(1, 3).Style.NumberFormat.Format = "#,##0.00";

// SetFormat metodu ile de aynı işlem yapılabilir, C sütunu ilk 4 satır

worksheet.Range(worksheet.Cell(1, 3), worksheet.Cell(4, 3)).Style.NumberFormat.SetFormat("#,##0.00");

// C4 hücresine formül tanımlıyoruz, C1 VE C2 yi topla diyoruz

worksheet.Cell("C4").SetFormulaA1("=SUM(C1:C2)");

// Bu metod ise sayfadaki tüm formüllerin tekrar hesaplanmasını sağlıyor

worksheet.RecalculateAllFormulas();

}

    return Ok(excelFile);

}

Yukarıda da bahsettiğim gibi bu işlemlerin her biri için set methodları da kullanılabilir, örneğin C1 için Formatlama işlemini doğrudan özelliğe atama yaparak sağladık. C1-C4 aralığı için ise SetFormat methodunu kullanarak format ataması sağladık. Bu arada Range methodunu da kullandık, Range methodu ise birden fazla hücre seçmek için kullanılıyor.

Yine devamında C4 hücresine bir toplama formülü tanımladık. Burada "=C1+C2" şeklinde de toplama işlemi sağlanabilir, exceldeki hemen hemen tüm formülleri bu kütüphane desteklemektedir, formül konusunu da sonraki makalelerde daha derinlemesine inceleyeceğiz.

Sanıyorum ki konu anlaşılmaya başlanmıştır :) Şimdi çok uzatmadan diğer kod parçalarını da aşağıda paylaşayım, her satırda zaten yorum olarak kodun ne iş yaptığını belirttim, sonrasında Html ve JavaScript tarafındaki download işlemi için kullandığımız kod parçasını paylaşıp makaleyi sonlandırayım.

Aşağıdaki kod parçasında makalenin başında bahsettiğim tüm örnekler bulunmakta ve açıklamaları yazmaktadır. Çalışma Kitabını en son MemoryStream'e basıp oradan byte array olarak response ediyoruz. Client tarafında ise base64 olarak alıp download ediyoruz.

[HttpGet]

[Route("export/simple")]

public IActionResult ExportSimpleExcel() {

byte[] excelFile;

// Excel dosyası (excel çalışma kitabı) oluşturuyoruz

using (var workbook = new XLWorkbook()) {

// Çalışma kitabına bir sayfa ekliyoruz, sayfa ismini istediğimiz gibi verebiliriz

var worksheet = workbook.Worksheets.Add("Sayfa 1");

// A1 hücresine değer atıyoruz

worksheet.Cell("A1").Value = "Hello world";

// Hücrelere alternatif olarak bu şekilde de değer atanabilir, ilk parametre satır index, ikinci parametre sütun index

worksheet.Cell(1, 3).SetValue(100); // C1

worksheet.Cell(2, 3).SetValue("200"); // C2

// Hücrenin veri tipini belirtiyoruz, C1

worksheet.Cell(1, 3).DataType = XLDataType.Number;

// Birden fazla hücreye tek seferde bu şekilde de veri tipi verilebilir, C sütunu ilk 4 satır

worksheet.Range(worksheet.Cell(1, 3), worksheet.Cell(4, 3)).DataType = XLDataType.Number;

// Hücreyi biçimlendiriyoruz, binlik ayraçları ve virgülden sonraki hane sayısını belirtmiş olduk, C1

worksheet.Cell(1, 3).Style.NumberFormat.Format = "#,##0.00";

// SetFormat metodu ile de aynı işlem yapılabilir, C sütunu ilk 4 satır

worksheet.Range(worksheet.Cell(1, 3), worksheet.Cell(4, 3)).Style.NumberFormat.SetFormat("#,##0.00");

// C4 hücresine formül tanımlıyoruz, C1 VE C2 yi topla diyoruz

worksheet.Cell("C4").SetFormulaA1("=SUM(C1:C2)");

// Bu metod ise sayfadaki tüm formüllerin tekrar hesaplanmasını sağlıyor

worksheet.RecalculateAllFormulas();

// Burada iki hücreyi birleştiriyoruz (A4 ve B4)

IXLRange totalCell = worksheet.Range(worksheet.Cell(4, 1), worksheet.Cell(4, 2)).Merge();

totalCell.Value = "TOPLAM";

// Hücreye arka plan rengi veriyoruz, FromColor ya da FromArgb fonksiyonu ile de özel renkler oluşturulabilir

totalCell.Style.Fill.SetBackgroundColor(XLColor.Black);

// Alternatif kullanım

//totalCell.Style.Fill.BackgroundColor = XLColor.FromColor(Color.Black);

// Hücrenin fontunu kalın yapıyoruz

totalCell.Style.Font.SetBold();

// Alternatif kullanım

// totalCell.Style.Font.Bold = true;

// Hücrenin yazı rengini, tipini ve boyutunu belirtiyoruz

totalCell.Style.Font.SetFontColor(XLColor.FromColor(Color.White));

totalCell.Style.Font.SetFontName("Tahoma");

totalCell.Style.Font.SetFontSize(9);

// Hücredeki veriyi yatay olarak sağa yasladık

totalCell.Style.Alignment.SetHorizontal(XLAlignmentHorizontalValues.Right);

// Hücredeki veriyi dikey olarak ortaladık

totalCell.Style.Alignment.SetVertical(XLAlignmentVerticalValues.Center);

// Metni Kaydır olarak belirttik

totalCell.Style.Alignment.SetWrapText();

// C4 Hücresine kenarlık oluşturduk ve kenarlık rengi belirttik

worksheet.Cell("C4").Style.Border.SetOutsideBorder(XLBorderStyleValues.Thick);

worksheet.Cell("C4").Style.Border.SetOutsideBorderColor(XLColor.Black);

// 4. Satır, 3. sütun yani C4 ü yatayda sağa yasladık

worksheet.Cell(4,3).Style.Alignment.SetHorizontal(XLAlignmentHorizontalValues.Right);

// C4 hücresini dikeyde ortaladık

worksheet.Cell("C4").Style.Alignment.SetVertical(XLAlignmentVerticalValues.Center);

// C4 hücresini bold yaptık

worksheet.Cell("C4").Style.Font.SetBold();

// C4 hücresinin rengini html hex color code ile kırmızı olarak belirttik

worksheet.Cell("C4").Style.Font.SetFontColor(XLColor.FromHtml("#FF0000"));

// A1 hücresine noktalı alt kenarlık oluşturduk

worksheet.Cell("A1").Style.Border.SetBottomBorder(XLBorderStyleValues.Dotted);

// A sütununun genişliğinin içeriğe göre ayarlanmasını sağladık

worksheet.Column("A").AdjustToContents();

// 3. Sütunun yani C sütununun genişliğini belirtiyoruz

worksheet.Column(3).Width = 20;

// 4. Satırın yüksekliğini belirtiyoruz

worksheet.Row(4).Height = 30;

using (MemoryStream memoryStream = new MemoryStream()) {

// Dosyayı bir path belirterek de kaydedebiliriz, biz burada stream olarak kaydedeceğiz

workbook.SaveAs(memoryStream);

// Stream olarak kaydettiğiniz dosyayı byte array olarak alıyoruz, bunu client'a response olarak döneceğiz

excelFile = memoryStream.ToArray();

}

}

// Dosyamızı byte array formatında response olarak dönüyoruz

return Ok(excelFile);

// Eğer JavaScript değil de direkt .net ile dosyayı indirmek istiyorsanız bu alternatifi kullanabilirsiniz.

// return File(excelFile, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");

}

Html ve JavaScript tarafı; Index.cshtml

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

<h1 class="display-4">Welcome</h1>

<p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>

@\*<a href="/export/simple" class="btn btn-primary">Export Excel (.NET)</a> \*@

<button onclick="exportExcel()" class="btn btn-primary">Export Excel (JS)</button>

</div>

@section Scripts {

<script>

function exportExcel() {

fetch('/export/simple', { method: 'GET' })

.then(response => {

return response.json()

}).then(data => {

downloadFile(data);

}).catch(e => {

console.log('error', e);

});

}

function downloadFile(base64) {

let bytes = base64ToByteArray(base64);

let blob = new Blob([bytes], { type: "application/octet-stream" });

let link = document.createElement('a');

link.href = window.URL.createObjectURL(blob);

let fileName = "simple.xlsx";

link.download = fileName;

link.click();

}

function base64ToByteArray(base64) {

let binaryString = window.atob(base64);

let len = binaryString.length;

let bytes = new Uint8Array(len);

for (let i = 0; i < len; i++) {

bytes[i] = binaryString.charCodeAt(i);

}

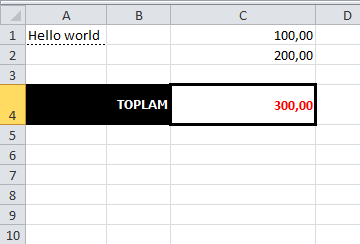
return bytes.buffer;

}

</script>

}

Excel çıktı görüntüsü ise şu şekilde;



Devam serisinde bir listeyi veya tabloyu nasıl export ederiz onu anlatacağım, sonraki serilerde ise mevcut bir exceli içe aktarmayı anlatacağım. Belki veritabanı üzerinden içe ve dışa aktarma örnekleri de yapabiliriz. Hatta birden fazla sayfa oluşturma veya daha karmaşık rapor örneklerini nasıl oluştururuz onları da anlatmayı hedefliyorum. Örneğin yıl yıl ve ay ay hangi üründen ne kadar satış yapılmış bunları sütunlar halinde güzelce oluşturup daha derinlemesine çalışmalar nasıl yapılır bunu kavrayabiliriz :)  
  
Umarım anlaşılır ve faydalı bir yazı olmuştur, makalenin devam serilerinde görüşmek üzere :)