##### ASP.NET Core ile ClosedXML Kullanımı (Liste Dışa Aktarma) - Part 3 6.9.2020

Biraz geç de olsa .NET ile ClosedXML serisine devam ediyoruz.

Daha önce ClosedXML kullanımı üzerine iki makale yazmıştım. İlk makalede basitçe excel dışa aktarma örnekleri yapmıştık, basit desem de aslında o makalede ClosedXML ile yapılabilecek bir çok şeyi yaptık. Hücreleri farklı alternatiflerle doldurmayı yani değer set etmeyi, hücreleri formatlandırmayı (biçimlendirmeyi), formül oluşturmayı, birden fazla hücrelerin merge edilmesi (birleştirilmesi), fontlarını düzenlemeyi, arka plan rengi, yazı rengi, kenarlık verilmesi, hücredeki verinin dikey ve yatay konumlandırılması, sütun ve satırların boyutlandırılması veya içeriğe göre genişletilmesinin sağlanması. Aslında excelde yapılabilecek bir çok şeyi biz ClosedXML kütüphanesini kullanarak kod ile yapmıştık.

İlk makaleye [**şuradan**](http://enestas.net/Detay/asp-net-core-mvc-excel-export-islemleri-closedxml-part-1) ulaşabilirsiniz. Bu makaleye başlamadan önce ilk makaleyi okumanız daha faydalı olacaktır.

İkinci makalede ise excel içe aktarmayı yapmıştık, basit bir tablo excelimiz vardı ve bunu içeriye aktarıp DataTable olarak almıştık, sonrasında json olarak ekranda göstermiştik. İkinci makale için de [**buraya**](http://enestas.net/Detay/asp-net-core-mvc-ile-excelden-veri-okuma-closedxml-part-2) tıklayınız :)

Bence ilk iki makaleden yola çıkarak ihtiyacınıza ve senaryonuza uygun şekilde export ve import işlemlerini geliştirebilirsiniz.

Bu makalede ise aslında ilk makaleye göre daha basit export işlemler yapacağız diyebilirim. Basitçe bir ürün listemiz olacak, bu sefer bu ürün listesini dışa aktaracağız.İlk örnekte tüm listeyi dışa aktaracağız ve listedeki veriden bağımsız olarak fiyat \* stok formülünü yedireceğimiz ek bir sütunumuz daha olacak. Bu sütunun en altında da sütun toplamı için bir formül olacak.

İkinci örnekte ise aynı listeyi marka bazlı gruplandırıp, excelde her marka için ayrı sayfalar oluşturarak aynı işlemleri gerçekleştireceğiz. Bu serinin son makalesinde ise daha detaylı ve biraz daha farklı bir örnek hazırlayacağım. Tabi o makaleden önce Google Drive senkronizasyonu üzerine bir makale hazırlayacağım için biraz gecikebilir :)

Şimdi gelelim ilk örneğimize. Öncelikle elimizde aşağıdaki gibi basitçe bir ürün modelimiz olsun.

Product.cs

public class Product {

public int IDProduct { get; set; }

public string Brand { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public decimal Stock { get; set; }

}

Sonrasında Controller'ımıza export için bir method açalım ve bir ürün listesi oluşturalım.

[HttpGet]

[Route("export/simple-list")]

public IActionResult ExportSimpleListExcel() {

byte[] excelFile;

List<Product> products = new List<Product> {

new Product { IDProduct = 1, Brand = "Apple", Name = "IPhone XS 256GB Space Gray", Price = 10000, Stock = 10 },

new Product { IDProduct = 2, Brand = "Apple", Name = "IPhone XS 256GB Gold", Price = 10500, Stock = 5 },

new Product { IDProduct = 3, Brand = "Samsung", Name = "Galaxy Note 20 Ultra 256GB", Price = 12999, Stock = 9 },

new Product { IDProduct = 4, Brand = "Huawei", Name = "P40 Pro 256GB", Price = 11999, Stock = 6 },

new Product { IDProduct = 5, Brand = "Xiaomi", Name = "Mi 10 256GB 5G", Price = 9999, Stock = 2 }

};

}

Ardından ClosedXML XLWorkbook sınıfını kullanarak excel dosyamızı oluşturalım ve ilk satıra sütun başlıklarımızı yazalım.

// 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");

// ilk satıra tablo başlığımızı oluşturuyoruz

worksheet.Cell(1, 1).SetValue("IDProduct");

worksheet.Cell(1, 2).SetValue("Marka");

worksheet.Cell(1, 3).SetValue("Adı");

worksheet.Cell(1, 4).SetValue("Fiyat");

worksheet.Cell(1, 5).SetValue("Stok");

worksheet.Cell(1, 6).SetValue("Toplam Değeri");

// ilk satıra stil veriyoruz ( CellsUsed sadece dolu hücrelere işlem yapmamızı sağlar)

worksheet.Row(1).CellsUsed().Style.Font.SetBold();

worksheet.Row(1).CellsUsed().Style.Font.SetFontSize(12);

worksheet.Row(1).CellsUsed().Style.Fill.SetBackgroundColor(XLColor.LightGray);

}

Excel dosyamıza bir sayfa ekledik, ilk satırımızda sütun başlıklarımızı oluşturduk ve ilk satırımızın fontunu kalın ve 12pt yaptık, sonrasında LightGray olarak da arka plan rengi belirledik.

Şimdi aşağıdaki tek satırlık komutla ürünlerimizi ekleyelim.

// ikinci satırdan itibaren ürün listemizi veriyoruz

worksheet.Cell(2, 1).InsertData(products);

Cell(2,1) ile hücreyi aldık ve InsertData metoduna products listemizi geçtik. Böylece tek komutla sayfamızın 2. satırının ilk sütunundaki (A2) hücreden itibaren ürün listemizi eklemesini sağladık.

Şimdi Fiyat ve Stok sütunundaki değerlerimizi formatlayalım ve ardından toplam değer sütunumuzun formüllerini yedirelim.

// 4. ve 5. sütunları yani fiyat ve stok sütunlarını number olarak formatlıyoruz, (2. satırdan itibaren eklenen ürün sayısı kadar)

worksheet.Column(4).Cells(2, products.Count + 1).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00");

worksheet.Column(5).Cells(2, products.Count + 1).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00");

// fiyat ve stok sütunlarının harf karşılıklarını alıyoruz

string priceColumnLetter = worksheet.Column(4).ColumnLetter();

string stockColumnLetter = worksheet.Column(5).ColumnLetter();

// exceldeki satırlarımızı 2. satırdan itibaren (yani ürün satırlarımızı) dönüyoruz

int i, n = worksheet.Rows().Count();

for (i = 2; i <= n; i++) {

// aktif satıra toplam değeri hesaplayacak formülü oluşturuyoruz.

// örneğin =D2\*E2 (i= aktif satır)

string totalFormula = $"={priceColumnLetter}{i}\*{stockColumnLetter}{i}";

worksheet.Cell(i, 6).SetFormulaA1(totalFormula);

}

// toplam değer sütununu formatlıyoruz

worksheet.Column(6).Cells(2, products.Count + 2).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00").Font.SetBold();

// elimizdeki toplam ürün değerini hesaplayan formülü de ekliyoruz

string totalColumnLetter = worksheet.Column(6).ColumnLetter();

string subTotalFormula = $"=SUM({totalColumnLetter}2:{totalColumnLetter}{n})";

worksheet.Cell(n + 1, 6).SetFormulaA1(subTotalFormula).Style.Font.SetFontSize(12);

// tüm formülleri hesaplatıyoruz

worksheet.RecalculateAllFormulas();

// sütunların içeriğe göre otomatik genişletilmesini sağlıyoruz

worksheet.Columns().AdjustToContents();

Excel sayfamızdaki yapacağımız işlemler bu kadar, sadece export etmek kaldı. Export işleminin de dahil olduğu metodun tam hali ise şu şekilde;

[HttpGet]

[Route("export/simple-list")]

public IActionResult ExportSimpleListExcel() {

byte[] excelFile;

List<Product> products = new List<Product> {

new Product { IDProduct = 1, Brand = "Apple", Name = "IPhone XS 256GB Space Gray", Price = 10000, Stock = 10 },

new Product { IDProduct = 2, Brand = "Apple", Name = "IPhone XS 256GB Gold", Price = 10500, Stock = 5 },

new Product { IDProduct = 3, Brand = "Samsung", Name = "Galaxy Note 20 Ultra 256GB", Price = 12999, Stock = 9 },

new Product { IDProduct = 4, Brand = "Huawei", Name = "P40 Pro 256GB", Price = 11999, Stock = 6 },

new Product { IDProduct = 5, Brand = "Xiaomi", Name = "Mi 10 256GB 5G", Price = 9999, Stock = 2 }

};

// 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");

// ilk satıra tablo başlığımızı oluşturuyoruz

worksheet.Cell(1, 1).SetValue("IDProduct");

worksheet.Cell(1, 2).SetValue("Marka");

worksheet.Cell(1, 3).SetValue("Adı");

worksheet.Cell(1, 4).SetValue("Fiyat");

worksheet.Cell(1, 5).SetValue("Stok");

worksheet.Cell(1, 6).SetValue("Toplam Değeri");

// ilk satıra stil veriyoruz ( CellsUsed sadece dolu hücrelere işlem yapmamızı sağlar)

worksheet.Row(1).CellsUsed().Style.Font.SetBold();

worksheet.Row(1).CellsUsed().Style.Font.SetFontSize(12);

worksheet.Row(1).CellsUsed().Style.Fill.SetBackgroundColor(XLColor.LightGray);

// ikinci satırdan itibaren ürün listemizi veriyoruz

worksheet.Cell(2, 1).InsertData(products);

// 4. ve 5. sütunları yani fiyat ve stok sütunlarını number olarak formatlıyoruz, (2. satırdan itibaren eklenen ürün sayısı kadar)

worksheet.Column(4).Cells(2, products.Count + 1).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00");

worksheet.Column(5).Cells(2, products.Count + 1).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00");

// fiyat ve stok sütunlarının harf karşılıklarını alıyoruz

string priceColumnLetter = worksheet.Column(4).ColumnLetter();

string stockColumnLetter = worksheet.Column(5).ColumnLetter();

// exceldeki satırlarımızı 2. satırdan itibaren (yani ürün satırlarımızı) dönüyoruz

int i, n = worksheet.Rows().Count();

for (i = 2; i <= n; i++) {

// aktif satıra toplam değeri hesaplayacak formülü oluşturuyoruz.

// örneğin =D2\*E2 (i= aktif satır)

string totalFormula = $"={priceColumnLetter}{i}\*{stockColumnLetter}{i}";

worksheet.Cell(i, 6).SetFormulaA1(totalFormula);

}

// toplam değer sütununu formatlıyoruz

worksheet.Column(6).Cells(2, products.Count + 2).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00").Font.SetBold();

// elimizdeki toplam ürün değerini hesaplayan formülü de ekliyoruz

string totalColumnLetter = worksheet.Column(6).ColumnLetter();

string subTotalFormula = $"=SUM({totalColumnLetter}2:{totalColumnLetter}{n})";

worksheet.Cell(n + 1, 6).SetFormulaA1(subTotalFormula).Style.Font.SetFontSize(12);

// tüm formülleri hesaplatıyoruz

worksheet.RecalculateAllFormulas();

// sütunların içeriğe göre otomatik genişletilmesini sağlıyoruz

worksheet.Columns().AdjustToContents();

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 actionresult için belirtilen route'u tetikleyerek dosyayı indirmek istiyorsanız bu alternatifi kullanabilirsiniz.

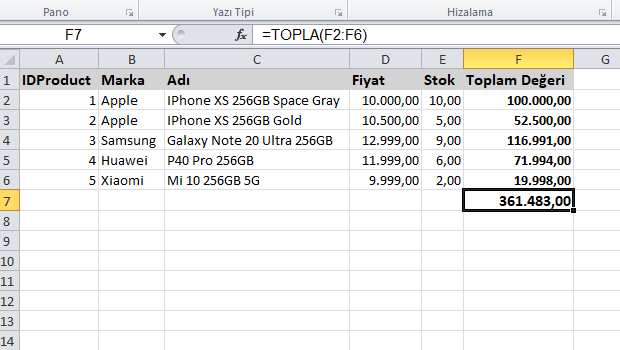
return File(excelFile, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");

}

Index.cshtml sayfamızdan bu methodu basitçe şu şekilde çağırabiliriz.

<a href="/export/simple-list" class="btn btn-primary">Export Excel Simple List</a>

Excel çıktımız ise görseldeki gibi olacaktır.



Bu makalede aslında ilk makaleden farklı olarak kullandığımız bir kaç komut var. **InsertData(), ColumnLetter()** ve **CellUsed()**. Aslında ilk makalede çok daha fazla komuta değinmiştik.

Ürünlerimizi marka marka ayrı sayfalar halinde almak da çok basit. Önce ürünlerimizi markaya göre gruplayacağız ve sonrasında bu grupları dönerek yukarıdaki işlemlerin aynısı yapacağız. Ama kod tekrarı yapmamak adına öncesinde yukarıdaki işlemlerin bir kısmını ayrı bir metod haline getirelim.

public void CreateAndFillWorksheet(IXLWorkbook workbook, string sheetName, List<Product> products) {

// Çalışma kitabına bir sayfa ekliyoruz, sayfa ismi parametre olarak geliyor

var worksheet = workbook.Worksheets.Add(sheetName);

// ilk satıra tablo başlığımızı oluşturuyoruz

worksheet.Cell(1, 1).SetValue("IDProduct");

worksheet.Cell(1, 2).SetValue("Marka");

worksheet.Cell(1, 3).SetValue("Adı");

worksheet.Cell(1, 4).SetValue("Fiyat");

worksheet.Cell(1, 5).SetValue("Stok");

worksheet.Cell(1, 6).SetValue("Toplam Değeri");

// ilk satıra stil veriyoruz ( CellsUsed sadece dolu hücrelere işlem yapmamızı sağlar)

worksheet.Row(1).CellsUsed().Style.Font.SetBold();

worksheet.Row(1).CellsUsed().Style.Font.SetFontSize(12);

worksheet.Row(1).CellsUsed().Style.Fill.SetBackgroundColor(XLColor.LightGray);

// ikinci satırdan itibaren parametreden gelen ürün listemizi veriyoruz

worksheet.Cell(2, 1).InsertData(products);

// 4. ve 5. sütunları yani fiyat ve stok sütunlarını number olarak formatlıyoruz, (2. satırdan itibaren eklenen ürün sayısı kadar)

worksheet.Column(4).Cells(2, products.Count + 1).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00");

worksheet.Column(5).Cells(2, products.Count + 1).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00");

// fiyat ve stok sütunlarının harf karşılıklarını alıyoruz

string priceColumnLetter = worksheet.Column(4).ColumnLetter();

string stockColumnLetter = worksheet.Column(5).ColumnLetter();

// exceldeki satırlarımızı 2. satırdan itibaren (yani ürün satırlarımızı) dönüyoruz

int i, n = worksheet.Rows().Count();

for (i = 2; i <= n; i++) {

// aktif satıra toplam değeri hesaplayacak formülü oluşturuyoruz.

// örneğin =D2\*E2 (i= aktif satır)

string totalFormula = $"={priceColumnLetter}{i}\*{stockColumnLetter}{i}";

worksheet.Cell(i, 6).SetFormulaA1(totalFormula);

}

// toplam değer sütununu formatlıyoruz

worksheet.Column(6).Cells(2, products.Count + 2).SetDataType(XLDataType.Number).Style.NumberFormat.SetFormat("#,##0.00").Font.SetBold();

// toplam ürün değerini hesaplayan formülü de ekliyoruz

string totalColumnLetter = worksheet.Column(6).ColumnLetter();

string subTotalFormula = $"=SUM({totalColumnLetter}2:{totalColumnLetter}{n})";

worksheet.Cell(n + 1, 6).SetFormulaA1(subTotalFormula).Style.Font.SetFontSize(12);

// tüm formülleri hesaplatıyoruz

worksheet.RecalculateAllFormulas();

// sütunların içeriğe göre otomatik genişletilmesini sağlıyoruz

worksheet.Columns().AdjustToContents();

}

Böylece ilk örnekteki tek sayfalık metodumuzu ise şu şekilde güncellemiş olduk.

List<Product> products = new List<Product> {

new Product { IDProduct = 1, Brand = "Apple", Name = "IPhone XS 256GB Space Gray", Price = 10000, Stock = 10 },

new Product { IDProduct = 2, Brand = "Apple", Name = "IPhone XS 256GB Gold", Price = 10500, Stock = 5 },

new Product { IDProduct = 3, Brand = "Samsung", Name = "Galaxy Note 20 Ultra 256GB", Price = 12999, Stock = 9 },

new Product { IDProduct = 4, Brand = "Huawei", Name = "P40 Pro 256GB", Price = 11999, Stock = 6 },

new Product { IDProduct = 5, Brand = "Xiaomi", Name = "Mi 10 256GB 5G", Price = 9999, Stock = 2 }

};

[HttpGet]

[Route("export/simple-list")]

public IActionResult ExportSimpleListExcel() {

byte[] excelFile;

// Excel dosyası (excel çalışma kitabı) oluşturuyoruz

using (var workbook = new XLWorkbook()) {

CreateAndFillWorksheet(workbook, "Sayfa 1", products);

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 actionresult için belirtilen route'u tetikleyerek dosyayı indirmek istiyorsanız bu alternatifi kullanabilirsiniz.

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

}

Ürünlerimizi marka bazlı gruplandırılmış ve excelde her marka için ayrı sayfa oluşturulmuş haliyle export alan metodumuz da aşağıdaki gibi olacaktır.

[HttpGet]

[Route("export/multipage-list")]

public IActionResult ExportMultiPageListExcel() {

byte[] excelFile;

// ürünlerimizi markaya göre grupluyoruz

List<IGrouping<string, Product>> groupedProducts = products.GroupBy(t => t.Brand).ToList();

// Excel dosyası (excel çalışma kitabı) oluşturuyoruz

using (var workbook = new XLWorkbook()) {

// markalarımızı dönüyoruz

int i, n = groupedProducts.Count;

for (i = 0; i < n; i++) {

CreateAndFillWorksheet(workbook, groupedProducts[i].Key, groupedProducts[i].ToList());

}

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 actionresult için belirtilen route'u tetikleyerek dosyayı indirmek istiyorsanız bu alternatifi kullanabilirsiniz.

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

}

Html tarafından metodun üzerindeki route'u çağırmak yeterli olacaktır.

Ado.net kullanan geliştiriciler eğer DataTable'ı export alacaksa InsertData() metodu yerine InsertTable() metoduna parametre olarak DataTable geçerek aynı işlemleri yapabilirler.

İlk iki makalenin örneklerini de içeren projenin tam halini [**şuradan**](http://enestas.net/upload/ClosedXMLExample.rar) indirip inceleyebilirsiniz. Proje VS 2019 ve Net Core 3.1 ile kullanarak hazırlanmıştır.