**ASP.NET Core MVC ile Excelden Veri Okuma (ClosedXML) Part 2**

**28.6.2020**

ClosedXML makale serimizin bu ikinci bölümünde excelden veri okuma işlemi sağlayacağız. Bu makaleyi okumadan önce bir önceki makaleyi de okumanızı öneririm. İlk makalede ClosedXML kütüphanesini projemize nasıl ekliyoruz onu da anlatmıştım.

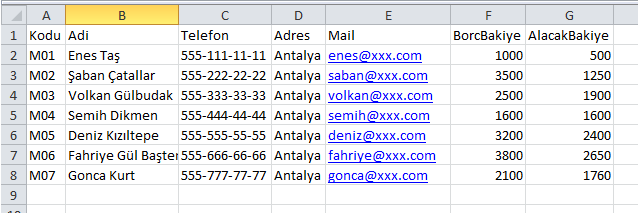
Bir önceki makaleye [şuradan](http://enestas.net/Detay/asp-net-core-mvc-excel-export-islemleri-closedxml-part-1) ulaşabilirsiniz.

ASP.NET uygulamalarında excelden veri okuma işlemleri hakkında OLEDB kullanılarak anlatılan bolca kaynak var veya Microsoft.Office.Interop.Excel komponenti kullanılarak anlatılan kaynaklar mevcut. Ancak bu iki yöntemde de bir takım sorunlarla karşılaşabiliriz. Uygulamamızı barındırdığımız sunucuya bazı kurulumlar yapmak durumunda kalabiliriz. Bu nedenle ben hem ClosedXML kütüphanesini detaylıca anlatmak adına hem de excel veri okuma yazma işlemlerini daha sorunsuz çözebilmek adına bu makale serisini yazmak istedim.

ClosedXML haricinde OpenXML kütüphanesini kullanarak da excel veri okuma yazma işlemleri yapılabilmekte ancak OpenXML kütüphanesinin kullanımının ClosedXML kütüphanesine göre çok daha zor olduğunu belirtmek isterim. ClosedXML de aslında kendi içerisinde OpenXML'i kullanmakta, sadece OpenXML ile yapılan işlemleri kolaylaştırmış bir kütüphane. Bunun da notunu kısaca düşmüş olalım :)

Bu makalede excel dosyamızı hem .Net ile, hem de JavaScript ile post etme işlemini de anlatacağım.  Çünkü projenizdeki yapıya veya duruma göre kullandığınız yöntem değişebilir, bu nedenle ben her iki yöntemi de anlatmayı tercih edeceğim.

Elimizde basitçe bir müşteri exceli olsun ve bu exceli içe aktaralım. Excelimiz şu şekilde;

Projemizin controller tarafındaki post metodumuz aşağıdaki gibidir. (HomeController.cs)

        [HttpPost]

[Route("import/excel")]

public IActionResult ImportExcel(IFormFile file) {

System.Data.DataTable dt = new System.Data.DataTable();

// excel dosyamızı stream'e çeviriyoruz

using (var ms = new MemoryStream()) {

file.CopyTo(ms);

// excel dosyamızı streamden okuyoruz

using (var workbook = new XLWorkbook(ms)) {

var worksheet = workbook.Worksheet(1); // sayfa 1

// sayfada kaç sütun kullanılmış onu buluyoruz ve sütunları DataTable'a ekliyoruz, ilk satırda sütun başlıklarımız var

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

for (i = 1; i <= n; i++) {

dt.Columns.Add(worksheet.Cell(1, i).Value.ToString());

}

// sayfada kaç satır kullanılmış onu buluyoruz ve DataTable'a satırlarımızı ekliyoruz

n = worksheet.Rows().Count();

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

DataRow dr = dt.NewRow();

int j, k = worksheet.Columns().Count();

for (j = 1; j <= k; j++) {

// i= satır index, j=sütun index, closedXML worksheet için indexler 1'den başlıyor, ama datatable için 0'dan başladığı için j-1 diyoruz

dr[j - 1] = worksheet.Cell(i, j).Value;

}

dt.Rows.Add(dr);

}

}

}

// tablomuzu json formatına çeviriyoruz

string json = JsonConvert.SerializeObject(dt);

return Ok(json);

}

Burada yaptığımız işlemi özetlersek; client tarafından upload edilen dosyamızı önce stream'e çeviriyoruz ve sonrasında ClosedXML kütüphanemizdeki XLWorkbook sınıfımızı kullanarak stream'den excel dosyamızı açıyoruz. İstersek upload edilen exceli dosya olarak kaydedip XLWorkbook sınıfına dosya yolunu parametre olarak geçerek de okuma işlemini sağlayabiliriz.

Excel dosyamızın ilk satırında sütun başlıklarımız var, bir DataTable oluşturup bu başlıkları kolon olarak ekliyoruz. Sonrasında da 2. satırdan itibaren tüm satırları dönüp bunları da tablomuza satır olarak yani veri olarak ekliyoruz. Kod içinde de satır satır açıklamaları yazdım.

Excelde veri yazılan kaç sütun olduğunu öğrenmek için **worksheet.Columns.Count()**komutunu kullanıyoruz, kaç satır olduğunu öğrenmek için de **worksheet.Rows().Count()** komutunu kullanıyoruz.   
  
Bu örnekte exceli okuyup bir DataTable'a bastık, bu tabloyu ado.net ile SqlBulkCopy kullanarak veritabanımızdaki bir tabloya toplu olarak aktarabiliriz. SqlBulkCopy ile ilgili bir çok örneği farklı kaynaklardan bulabilirsiniz. SqlBulkCopy işlemindeki önemli detay veritabanındaki tablonuzun kolon başlıkları ile datatable'daki kolon başlıkları ve sıralaması aynı olmalıdır. DataTable haricinde siz de kendi yapınıza uygun olarak okuma işlemleri sağlayabilirsiniz. Bir sınıfınız varsa ve o formatta okuyacaksanız kodunuzu ona göre revize edebilirsiniz.

Ben işlem sonrasında DataTable'ı json'a çevirip response olarak döndüm, hatta DataTable'daki kolonları sınıfınıza uygun olarak tanımlayıp, excel okumasını da uygun şekilde revize ettikten sonra json'a çevirdiğiniz tabloyu tekrar sınıfınız için bir list tanımlayıp ona deserialize edebilirsiniz de.

View tarafındaki kod bloğumuz ise aşağıdaki şekilde; (Index.cshtml)

@{

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

}

<div class="text-center">

<h1 class="display-4">Import Excel</h1>

<form enctype="multipart/form-data" action="/import/excel" method="post">

<input type="file" name="file" />

<button type="submit" class="btn btn-primary">Import Excel</button>

</form>

</div>

Eğer JavaScript kullanarak post etmeniz gerekiyorsa, aşağıdaki script'i eklemeniz yeterli;

<script>

form.onsubmit = async function (e) {

e.preventDefault();

const formData = new FormData(form);

fetch('/import/excel', { method: 'POST', body: formData })

.then(response => {

return response.json()

}).then(data => {

console.log(data);

}).catch(e => {

console.log('error', e);

});

}

</script>

Ya da form kullanmadan post işlemi sağlayacağım diyorsanız bir örnek daha paylaşayım;

@{

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

}

<div class="text-center">

<h1 class="display-4">Import Excel</h1>

<input type="file" name="file" id="file" />

<button onclick="importExcel()" class="btn btn-primary">Import Excel</button>

</div>

@section Scripts {

<script>

async function importExcel() {

const file = await getFile();

const formData = new FormData();

formData.append('file', file);

fetch('/import/excel', { method: 'POST', body: formData })

.then(response => {

return response.json()

}).then(data => {

console.log(data);

}).catch(e => {

console.log('error', e);

});

}

async function getFile() {

const file = document.querySelector('#file').files[0];

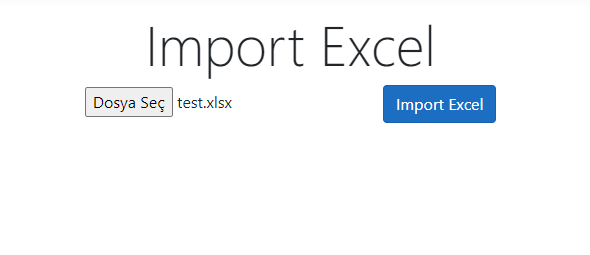
return file;

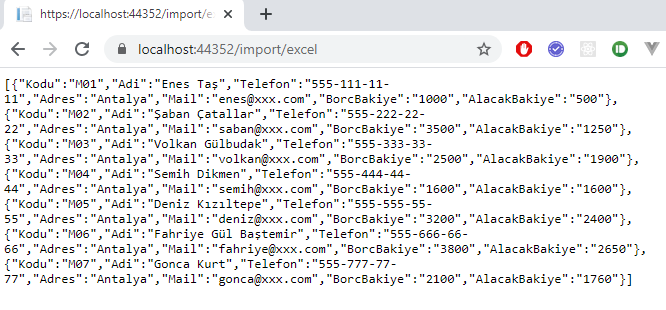
}

</script>

}

Bunun dışında JavaScript ile dosyanızı base64 formatına çevirip o şekilde de import edebilirsiniz ama bunun için örnek paylaşmaya gerek duymadım, yine de ihtiyacınız olan yapıya göre kaynak bulamadığınızda yorum yazıp destek istemekten çekinmeyin :)   
  
İşlem örnek görüntülerini de aşağıdaki şekilde paylaşıyorum;



  
  
Umarım ihtiyacı olanlara faydalı ve anlaşılır bir kaynak oluşturabilmişimdir :) Bu makalede sadece ClosedXML ile excelden veri okumayı değil aynı zamanda bilmeyenler için 3 farklı şekilde file upload işlemini de öğrenmiş olduk diye düşünüyorum. Bir önceki makalede de excel export ile birlikte aslında yine hem .net core mvc hem de JavaScript ile file download işlemlerini de öğrenmiş olduk.   
  
Bir sonraki makalede inşallah daha canlı ve detaylı örneklerle konuyu tamamlayacağım ve proje olarak da kaynak kodlarını paylaşacağım.