بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

آموزش مرتب سازی، فیلتر کردن و صفحه بندی با استفاده از EF در یک برنامه ی ASP.NET MVC

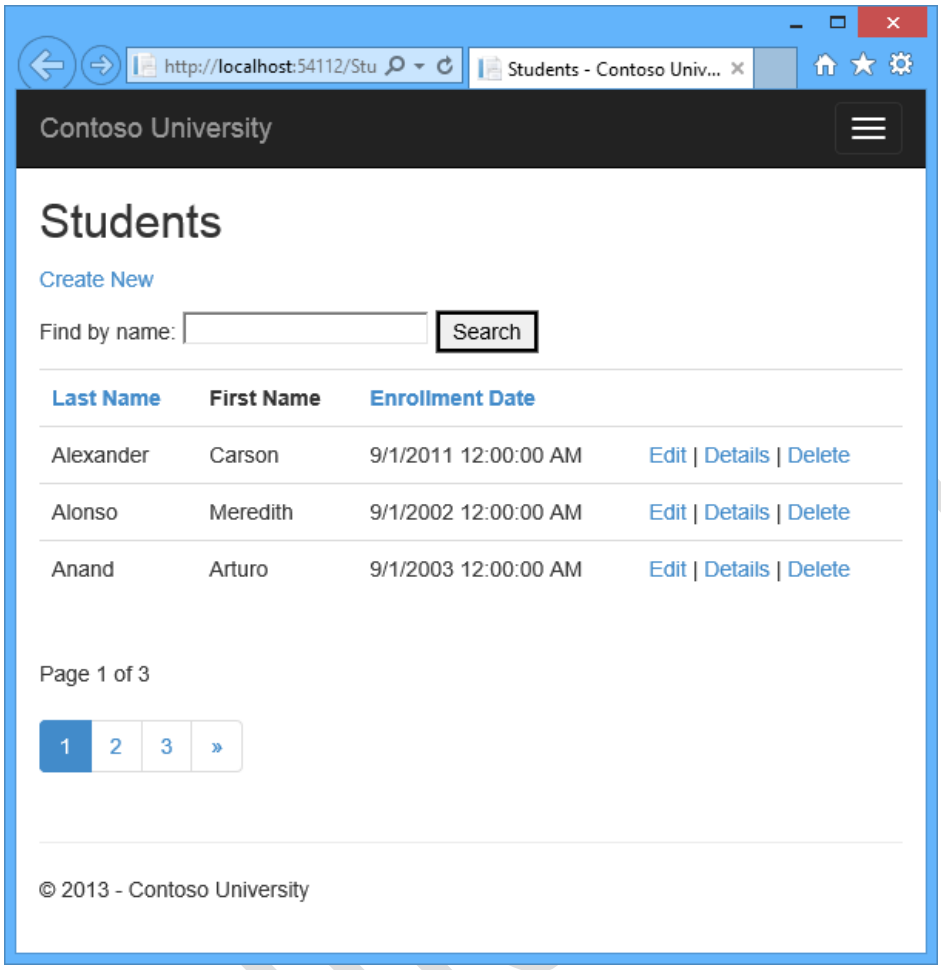
مدرس : مهندس افشین رفوآ

دوره آموزش MVC

آموزش مرتب سازی، فیلتر کردن و صفحه بندی با استفاده از EF در یک برنامه ی ASP.NET MVC

در فصل قبلی تعدادی صفحه ی وب را ویژه ی عملیات ساده ی ایجاد، خواندن، بروز رسانی و حذف (CRUD) برای موجودیت Student پیاده سازی کریدم. در درس حاضر قابلیت های مرتب سازی، فیلتر کردن و صفحه بندی را به صفحه ی Students Index اضافه خواهیم کرد. همچنین یک صفحه ایجاد می کنیم که عملیات ساده ی گروه بندی را اجرا می کند.

تصویر زیر، نسخه ی کامل برنامه را پس از انجام تمام عملیات به نمایش می گذارد. سرستون هایی که مشاهده می کنید، در واقع لینک هایی هستند که کاربر با کلیک بر روی آن ها می تواند فیلتر را بر اساس ستون مربوطه انجام دهد. کلیک مکرر بر روی سرستون باعث می شود ستون بین دو حالت ترتیب نمایش نزولی و صعودی تغییر حالت دهد.



افزودن لینک های مرتب سازی به صفحه ی Students Index

جهت اضافه کردن قابلیت مرتب سازی به صفحه ی Student Index، بایستی متد Index از کنترلگر Student را تغییر داده و کدهایی را به Student Index view اضافه کنید.

افزودن قابلیت مرتب سازی به متد Index

در فایل Controllers\StudentController.cs، کد زیر را جایگزین متد Index کنید:

```
public ActionResult Index(string sortOrder)
{
 ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
 ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";
 var students = from s in db.Students
           select s;
 switch (sortOrder)
 {
  case "name_desc":
```

```
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }
    return View(students.ToList());
}
```

کد بالا، یک پارامتر sortOrder از query string موجود در URL بازیابی می کند. مقدار query string توسط ASP.NET MVC به عنوان یک پارامتر در اختیار action method مورد نظر قرار می گیرد. پارامتر مورد نظر یک رشته است که اسم آن یا "Name" و یا "Date" خواهد بود که در صورت تمایل یک زیرخط (underscore) و نیز یک رشته "desc" که ترتیب نمایش آیتم ها را نزولی تعریف می کند به دنبال آن درج می گردد. ترتیب مرتب سازی، به صورت پیش فرض بر روی نمایش صعودی تنظیم می شود.

اولین باری که صفحه ی Index درخواست می شود، هیچ query string ای وجود ندارد. فیلدهای students به ترتیب صعودی بر اساس LastName نمایش داده می شوند که حالت پیش فرض می باشد؛ همان طور که fall-through case در ساختمان switch تعریف می کند. هنگامی که کاربر بر روی یک لینک سرستون کلیک می کند، مقدار sortOrder صحیح در query string قرار می گیرد.

دو متغیر ViewBag به این منظور بکار می روند تا view بتواند لینک های سرستون را با مقادیر مناسب query string تنظیم کند:

```
ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";
```

به این دستورات، دستورات سه گانه (به انگلیسی ternary statements) گفته می شود. اولین دستور مشخص می کند که اگر پارامتر null sortOrder یا تهی بود، ViewBag.NameSortParm باید روی "name_desc"; تنظیم شود و در غیر این صورت، بر روی رشته ی خالی تنظیم شود. این دو دستور به view اجازه می دهند که لینک های سرستون را به ترتیب زیر تنظیم کند:

Current sort order	Last Name Hyperlink	Date Hyperlink
Last Name ascending	descending	ascending
Last Name descending	ascending	ascending
Date ascending	ascending	descending
Date descending	ascending	ascending

متد مورد نظر با استفاده از LINQ to Entities، ستونی که بر اساس آن مرتب سازی صورت می گیرد را تعیین می کند. کد قبل از دستور switch، یک متغیر IQueryable ایجاد می کند، سپس آن متغیر را در دستور switch اصلاح کرده و متد ToList را بعد از دستور switch صدا می زند. هنگامی که متغیرهای IQueryable را ایجاد و اصلاح می کنید، هیچ query ای به پایگاه داده ارسال نمی شود. Query مورد نظر تا زمانی که شما شی IQueryable را با فراخوانی متدی همچون ToList به مجموعه (collection) تبدیل نکرده اید، اجرا نمی شود. باتوجه به آنچه گفته شد، کد جاری صرفا یک query ای را تولید می کند که تا رسیدن به دستور return View اجرا نمی شود.

به جای نوشتن چندین دستور مختلف LINQ ویژه ی هر ترتیب مرتب سازی (ascending و descending)، می توانید یک دستور LINQ به صورت پویا (dynamic) ایجاد کنید.

افزودن لینک های سرستون به Student Index View

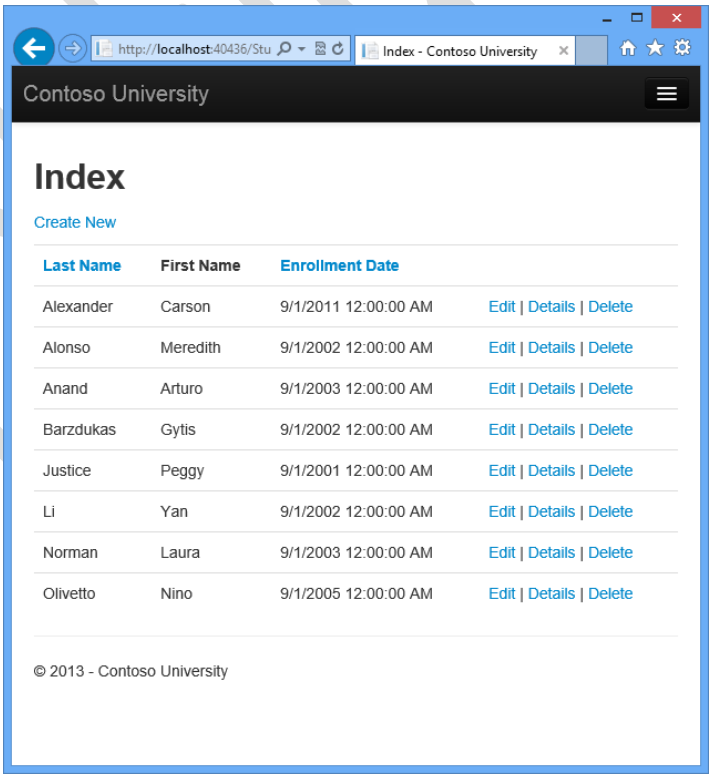
فایل Views\Student\Index.cshtml را باز کرده و کد رنگی شده ی زیر را جایگزین المان های <tr> و <th> برای heading row (سطر عنوان) کنید:

```
<p>
 @Html.ActionLink("Create New", "Create")
```

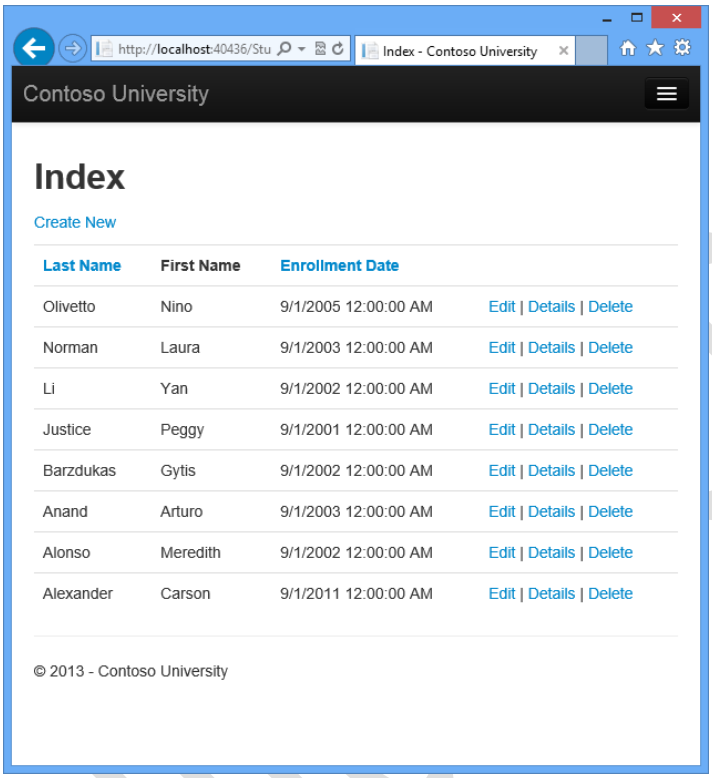
```
</p>
<table>
    <tr>
        <th>
            @Html.ActionLink("Last Name", "Index", new { sortOrder = ViewBag.NameSortParm })
        </th>
        <th>First Name
        </th>
        <th>
            @Html.ActionLink("Enrollment Date", "Index", new { sortOrder = ViewBag.DateSortParm })
        </th>
        <th></th>
    </tr>
@foreach (var item in Model) {
```

این کد با استفاده از اطلاعات موجود در property های شی ViewBag لینک هایی را با مقادیر query string مناسب تنظیم و ایجاد می کند.

صفحه را اجرا کرده و سرستون های Last Name و Enrollment Date را کلیک نمایید تا از کارکرد صحیح مرتب سازی اطمینان حاصل کنید.



پس از کلیک بر روی سرستون Last Name، خواهید دید که دانشجویان بر اساس last name به ترتیب نزولی نمایش داده می شوند.



افزودن یک کادر جستجو به صفحه ی Students Index

به منظور افزودن قابلیت فیلتر کردن به صفحه ی Index، یک کادر متن (textbox) و یک دکمه ی ارسال (submit button) به view افزوده و تغییرات مشابه را در متد Index اعمال کنید.کادر متن به شما اجازه ی وارد کردن یک رشته ی متنی برای جستجو در فیلدهای first name و last name را می دهد.

افزودن قابلیت فیلتر به متد Index

در فایل Controllers\StudentController.cs، کد رنگی شده ی زیر را جایگزین متد Index کنید:

```
public ViewResult Index(string sortOrder, string searchString)
{
```

```
    ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
    ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";
    var students = from s in db.Students
                   select s;
    if (!String.IsNullOrEmpty(searchString))
    {
        students = students.Where(s => s.LastName.Contains(searchString)
                               || s.FirstMidName.Contains(searchString));
    }
    switch (sortOrder)
    {
        case "name_desc":
            students = students.OrderByDescending(s => s.LastName);
            break;
        case "Date":
            students = students.OrderBy(s => s.EnrollmentDate);
            break;
        case "date_desc":
            students = students.OrderByDescending(s => s.EnrollmentDate);
            break;
        default:
            students = students.OrderBy(s => s.LastName);
            break;
    }
    return View(students.ToList());
}
```

در کد بالا، یک پارامتر searchString به متد Index اضافه کرده ایم. مقدار search string (عبارت جستجو) از یک textbox که به Index view اضافه خواهیم کرد، گرفته می شود. همچنین همان طور که در این کد مشاهده می کنید، یک عبارت where به دستور LINQ اضافه کرده ایم که تنها دانشجویانی (student) را گزینش می کند که first name و last name آن ها مشمول عبارت جستجو باشد. دستوری که عبارت where را اضافه می کند، تنها در صورتی اجرا می شود که مقداری برای جستجو وجود داشته باشد.

نکته: در بسیاری از موارد می توانید متد یکسان را برای یک مجموعه جدول (entity set) EF صدا بزنید و یا آن را به عنوان یک متد الحاقی (extension method) برای یک مجموعه ی مقیم در حافظه (in-memory collection) فراخوانی کنید. نتایج حاصل به طور معمول یکسان است، با این حال ممکن است در برخی موارد نتایج مورد نظر کمی متفاوت باشند.

به عنوان مثال، پیاده سازی متد Contains از جانب .NET، هنگامی که رشته ی تهی به آن ارسال می کنید، تمامی سطرها را بازیابی می کند، این درحالی است که Entity Framework provider ویژه ی SQL Server

Compact 4.0 در ازای ارسال رشته ی تهی، هیچ سطری را برنمی گرداند. بنابراین کدی که در مثال فوق مشاهده کردید (که در آن عبارت Where داخل دستور if قرار داده شده)، اطمینان کسب می کند که در تمامی نسخه های SQL Server، نتیجه ی یکسان حاصل گردد. همچنین پیاده سازی متد Contains (توسط چارچوب کاری .NET)، به صورت پیش فرض یک مقایسه ی حساس به کوچک و بزرگی حروف صورت می دهد، در حالی که ارائه دهنده های (providor) Entity Framework SQL Server، به صورت پیش فرض مقایسه ی غیر حساس به کوچک و بزرگی حروف اجرا می کند. از این رو فراخوانی متد ToUpper جهت case-insensitive کردن تست، باعث می شود که با عوض کردن کد برای استفاده از یک repository، کد مورد نظر تغییر نکند، که بجای بازگردانی شی IQueryable، یک مجموعه IEnumerable برمی گرداند. (هنگامی که متد Contains را برای مجموعه ی IEnumerable صدا می زنید، پیاده سازی .NET اجرا می شود، در حالی که اگر متد مذکور را برای شی IQueryable فراخوانی کنید، پیاده سازی database provider انجام می شود.)

مدیریت مقدار null نیز ممکن است در database provider های مختلف یا زمانی که از شی IQueryable استفاده می کنید درمقایسه با زمانی که از مجموعه ی IEnumerable استفاده می کنید، متفاوت باشد.

به عنوان مثال در برخی سناریوها ممکن است شرط Where (مانند این نمونه table.Column != 0)، سطرهایی که مقدار آن ها null می باشد را بازیابی نکند.

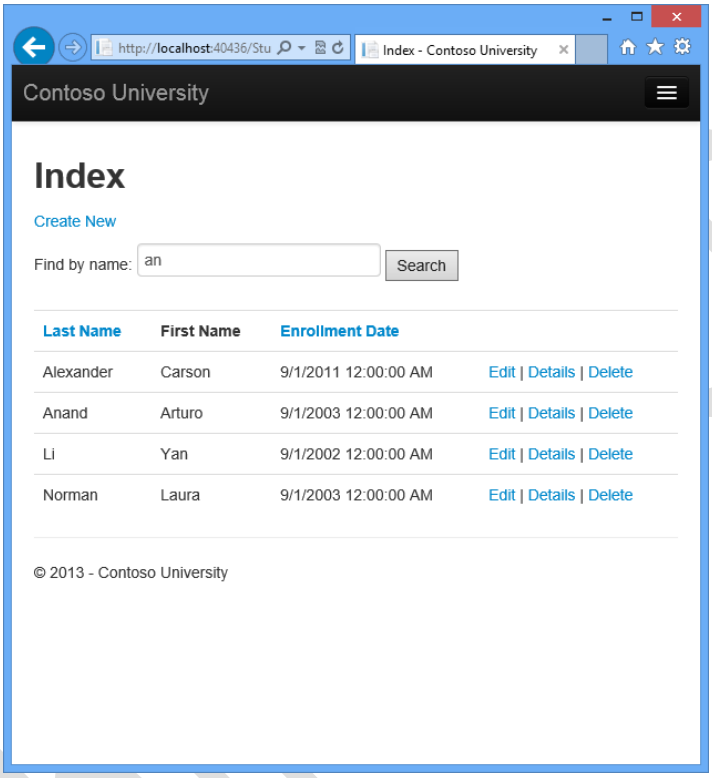
افزودن یک کادر جستجو به Student Index View

فایل Views\Student\Index.cshtml را باز کرده و کد رنگی شده ی زیر را درست قبل از تگ باز table درج کنید تا یک عنوان (caption)، textbox و یک دکمه ی Search ایجاد شود.

```
<p>
   @Html.ActionLink("Create New", "Create")
</p>
@using (Html.BeginForm())
{
   <p>
      Find by name: @Html.TextBox("SearchString")
      <input type="submit" value="Search" />
   </p>
```

```
}
<table>
  <tr>
```

صفحه را اجرا کرده، یک عبارت جستجو وارد کنید، سپس Search را کلیک کرده تا از کارکرد صحیح قابلیت filtering مطمئن شوید.



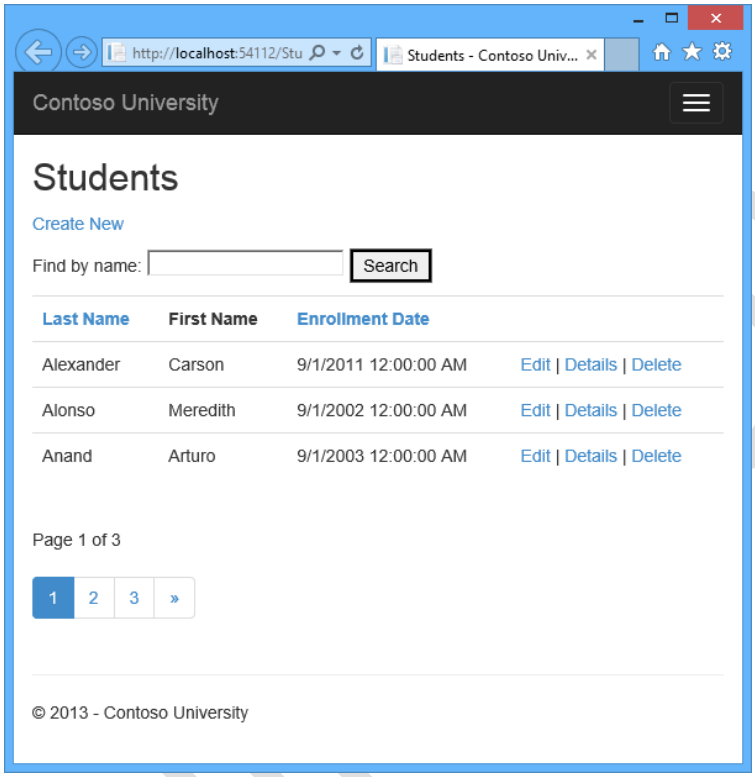
همان طور که مشاهده می کنید، URL شامل عبارت جستجو (search string) "an" نمی باشد، بدین معنی پس از bookmark کردن صفحه، نمی توانید همان لیست فیلترشده ای که در صورت عدم استفاده از bookmark دریافت می کردید، بدست آورید.

افزودن قابلیت صفحه بندی (Paging) به صفحه ی Students Index

جهت افزودن قابلیت صفحه بندی به صفحه ی Students Index، بایستی کار خود را با نصب پکیج PagedList.Mvc NuGet آغاز کنید. سپس تغییراتی را در متد Index ایجاد کرده و لینک های صفحه بندی (paging link) را به Index view اضافه می کنیم. PagedList.Mvc تنها یکی از پکیج های سودمند مرتب

سازی و صفحه بندی برای ASP.NET MVC است، و کاربرد آن دراینجا بدین معنا نیست که استفاده از آن به دیگر گزینه ها، ترجیح داده می شود.

تصویر زیر لینک های صفحه بندی را نمایش می دهد:



نصب پکیج PagedList.MVC NuGet

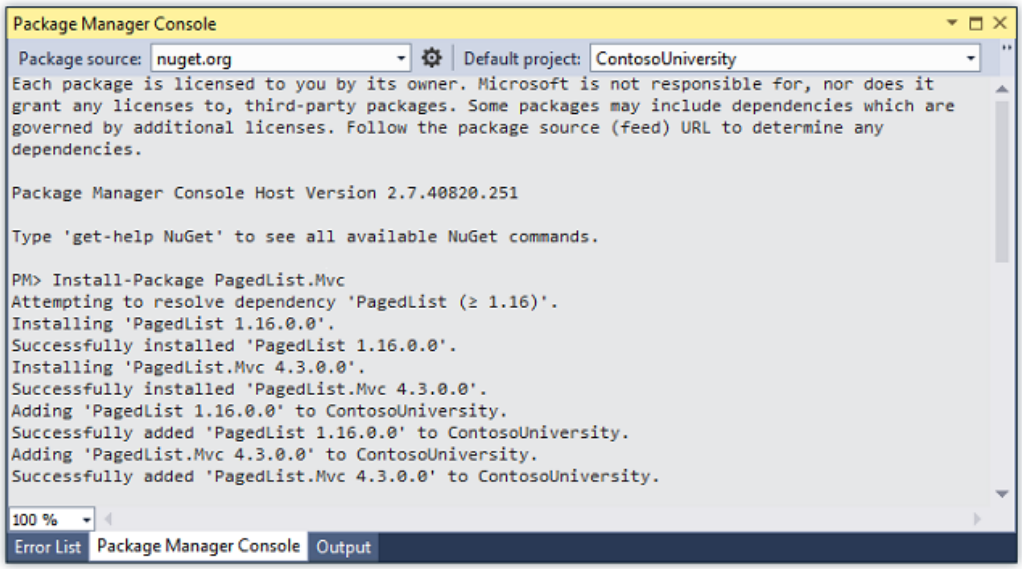
پکیج PagedList.Mvc، خودکار پکیج PagedList را به عنوان یک dependency نصب می کند. پکیج PagedList، یک collection type (نوع مجموعه) PagedList و متدهای الحاقی (extension methods) برای مجموعه های IQueryable و IEnumerable نصب می کند. متدهای الحاقی یک صفحه ی واحد از داده ها از نوع های IQueryable و IEnumerable در مجموعه ی PagedList ایجاد می کند، و مجموعه ی

PagedList نیز با ارائه ی چندین متد و خاصیت (property)، صفحه بندی را آسان می سازد. پکیج PagedList.Mvc همچنین یک paging helper نصب می کند که توسط آن دکمه های صفحه بندی نمایش داده می شوند.

از منوی Tools، Library Package Manager و به دنبال آن Package Manager Console را انتخاب کنید.

در پنجره ی Package Manager Console، پس از مطمئن شدن از صحیح بودن این موارد: Package source = nuget.org و Default project = ContosoUniversity، سپس دستور زیر را وارد کنید:

```
Install-Package PagedList.Mvc
```



پروژه را بازسازی (build) کنید.

افزودن قابلیت صفحه بندی به متد Index

فایل Controllers\StudentController.cs را باز کرده و یک دستور using برای فضای نام (namespace) PagedList اضافه نمایید:

```
using PagedList;
```

کد زیر را جایگزین متد Index کنید:

```
public ViewResult Index(string sortOrder, string currentFilter, string searchString, int? page)
{
  ViewBag.CurrentSort = sortOrder;
  ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
  ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";
  if (searchString != null)
  {
    page = 1;
  }
  else
  {
    searchString = currentFilter;
  }
  ViewBag.CurrentFilter = searchString;
  var students = from s in db.Students
           select s;
  if (!String.IsNullOrEmpty(searchString))
  {
    students = students.Where(s => s.LastName.Contains(searchString)
    || s.FirstMidName.Contains(searchString));
  }
  switch (sortOrder)
  {
    case "name_desc":
      students = students.OrderByDescending(s => s.LastName);
      break;
    case "Date":
      students = students.OrderBy(s => s.EnrollmentDate);
      break;
    case "date_desc":
      students = students.OrderByDescending(s => s.EnrollmentDate);
      break;
    default:  // Name ascending
      students = students.OrderBy(s => s.LastName);
      break;
  }
  int pageSize = 3;
  int pageNumber = (page ?? 1);
  return View(students.ToPagedList(pageNumber, pageSize));
}
```

این کد یک پارامتر page، یک پارامتر ترتیب مرتب سازی جاری، یک پارامتر فیلتر جاری به امضای (ورودی) متد (method signature) اضافه می کند:

```
public ActionResult Index(string sortOrder, string currentFilter, string searchString, int? page)
```

چناچه صفحه اولین بار است که نمایش داده می شود و یا اگر کاربر بر روی لینک مرتب سازی یا صفحه بندی کلیک نکرده باشد، در آن صورت تمامی پارامترها null خواهد بود. در صورتی که کاربر بر روی لینک صفحه بندی کلیک کند، متغیر page، شماره ی صفحه ی مورد نظر را برای نمایش دربرخواهد گرفت.

یکی از خاصیت های ViewBag، ترتیب مرتب سازی جاری (current sort order) را دراختیار view قرار می دهد. با این کار، ترتیب مرتب سازی در لینک های صفحه بندی گنجانده شده و بدین وسیله مانع از تغییر آن (sort order) به هنگام صفحه بندی می شود:

```
ViewBag.CurrentSort = sortOrder;
```

یک خاصیت دیگر، ViewBag.CurrentFilter، رشته ی فیلترکننده جاری (filter string) را به view ارائه می دهد. این مقدار بایستی در داخل لینک های صفحه بندی لحاظ شود تا بتوان تنظیمات فیلترینگ را حین صفحه بندی حفظ کرد، همچنین (filter string) باید پس از نمایش مجدد صفحه به textbox بازگردانده شود. چنانچه عبارت جستجو حین صفحه بندی تغییر یافت، در آن صورت صفحه باید به 1 reset شود، زیرا که فیلتر جدید ممکن است داده های متفاوتی را برای نمایش فراهم کند. هنگامی که مقدار جدیدی داخل textbox وارد شده و بعد از آن دکمه ی submit کلیک می شود، عبارت جستجو تغییر می کند. در چنین وضعیتی، پارامتر searchString تهی (null) نخواهد بود.

```
if (searchString != null)
{
  page = 1;
}
else
{
  searchString = currentFilter;
}
```

در انتهای کد زیر (متد)، مشاهده می کنید که متد الحاقی ToPagedList که به شی IQueryable الصاق شده، student query را به یک صفحه ی واحد از student ها، در یک نوع مجموعه ای (collection type) که

از قابلیت paging پشتیبانی می کند، تبدیل می کند. سپس آن صفحه ی واحد متشکل از فیلدهای student به view فرستاده می شود:

```
 {int pageSize = 3;
int pageNumber = (page ?? 1);
return View(students.ToPagedList(pageNumber, pageSize));
```

متد ToPagedList ، یک pageNumber به عنوان پارامتر می گیرد. دو علامت سوالی که مشاهده می کنید، نشانگر عملگر null-coalescing می باشد. عملگر ذکر شده یک مقدار پیش فرض برای نوع nullable تعریف می کند؛ عبارت (page ?? 1) به این معنی است که اگر صفحه مقداری داشت، مقدار page را برگردان و در غیر این صورت 1 را بازیابی کن.

افزودن لینک صفحه بندی به Student Index View

به فایل Views\Student\Index.cshtml مراجعه کرده و کد زیر را جایگزین کد جاری کنید. تغییرات اعمال شده با رنگ زرد هایلایت شده:

```
@model PagedList.IPagedList<ContosoUniversity.Models.Student>
@using PagedList.Mvc;
<link href="~/Content/PagedList.css" rel="stylesheet" type="text/css" />
@{
  ViewBag.Title = "Students";
}
<h2>Students</h2>
<p>
  @Html.ActionLink("Create New", "Create")
</p>
@using (Html.BeginForm("Index", "Student", FormMethod.Get))
{
  <p>
    Find by name: @Html.TextBox("SearchString", ViewBag.CurrentFilter as string)
    <input type="submit" value="Search" />
  </p>
}
<table class="table">
  <tr>
    <th>
```

```
        @Html.ActionLink("Last Name", "Index", new { sortOrder = ViewBag.NameSortParm,
currentFilter=ViewBag.CurrentFilter })
      </th>
      <th>
        First Name
      </th>
      <th>
        @Html.ActionLink("Enrollment Date", "Index", new { sortOrder = ViewBag.DateSortParm,
currentFilter=ViewBag.CurrentFilter })
      </th>
      <th></th>
    </tr>
@foreach (var item in Model) {
    <tr>
      <td>
        @Html.DisplayFor(modelItem => item.LastName)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.FirstMidName)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.EnrollmentDate)
      </td>
      <td>
        @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
        @Html.ActionLink("Details", "Details", new { id=item.ID }) |
        @Html.ActionLink("Delete", "Delete", new { id=item.ID })
      </td>
    </tr>
}
</table>
<br />
Page @(Model.PageCount < Model.PageNumber ? 0 : Model.PageNumber) of @Model.PageCount
@Html.PagedListPager(Model, page => Url.Action("Index",
  new { page, sortOrder = ViewBag.CurrentSort, currentFilter = ViewBag.CurrentFilter }))
```

دستور @model که در بالای صفحه درج شده، نشانگر این است که اکنون view بجای شی List، یک شی PagedList می گیرد. دستور using در PagedList.Mvc، امکان دسترسی به دکمه های صفحه بندی را به MVC helper می دهد.

این کد یک نسخه ی overload شده از متد الحاقی BeginForm را بکار می برد که به آن اجازه می دهد FormMethod.Get را به عنوان پارامتر ورودی بکار ببرد.

```
@using (Html.BeginForm("Index", "Student", FormMethod.Get))
{
  <p>
    Find by name: @Html.TextBox("SearchString", ViewBag.CurrentFilter as string)
    <input type="submit" value="Search" />
  </p>
}
```

BeginForm پیش فرض، داده های فرم را با روش POST ارسال می کند، بدین معنی که پارامترها داخل بدنه ی HTTP message فرستاده می شوند و نه درون بدنه ی URL به صورت query string. در حالی که اگر بجای HTTP POST، HTTP GET بکار ببرید، داده های فرم به صورت query string در داخل URL فرستاده می شوند که به کاربر اجازه می دهد URL را bookmark کند. رهنمودهای W3C بر این تاکیید می کند که در شرایطی که action مورد نظر منجر به بروز رسانی نمی شود، تا حد امکان از روش ارسال GET استفاده کنید.

Textbox با عبارت جستجوی جاری مقداردهی شده به گونه ای که هنگامی که شما بر روی یک صفحه ی جدید کلیک می کنید، می توانید search string کنونی را ببینید.

```
Find by name: @Html.TextBox("SearchString", ViewBag.CurrentFilter as string)
```

لینک سرستون با استفاده از query string، عبارت جستجو (search string) فعلی را به controller ارسال نموده تا کاربر بتواند نتایج فیلتر را مرتب کند:

```
@Html.ActionLink("Last Name", "Index", new { sortOrder=ViewBag.NameSortParm,
currentFilter=ViewBag.CurrentFilter
```

صفحه ی جاری به ضمیمه ی تعداد کل صفحات نمایش داده شده است.

```
Page @ (Model.PageCount < Model.PageNumber ? 0 : Model.PageNumber) of @Model.PageCount
```

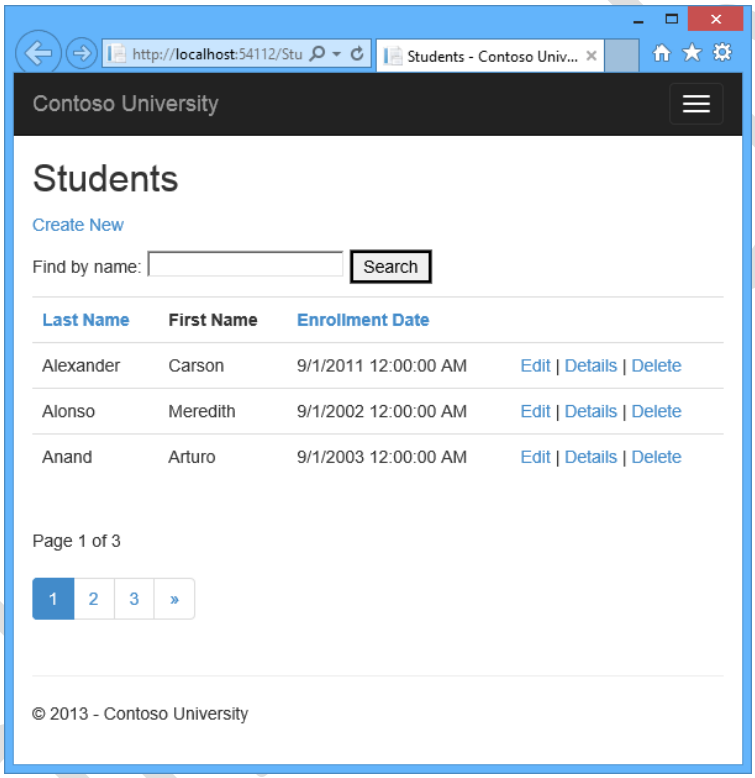
اگر هیچ صفحه ای برای نمایش وجود نداشته باشد، "Page 0 of 0" نمایش داده می شود. (در چنین شرایطی شماره ی صفحه از تعداد صفحات بزرگتر می باشد، زیرا Model.PageNumber برابر 1 و Model.PageCount برابر با 0 می باشد.)

دکمه های صفحه بندی توسط PagedListPager helper نمایش داده می شوند:

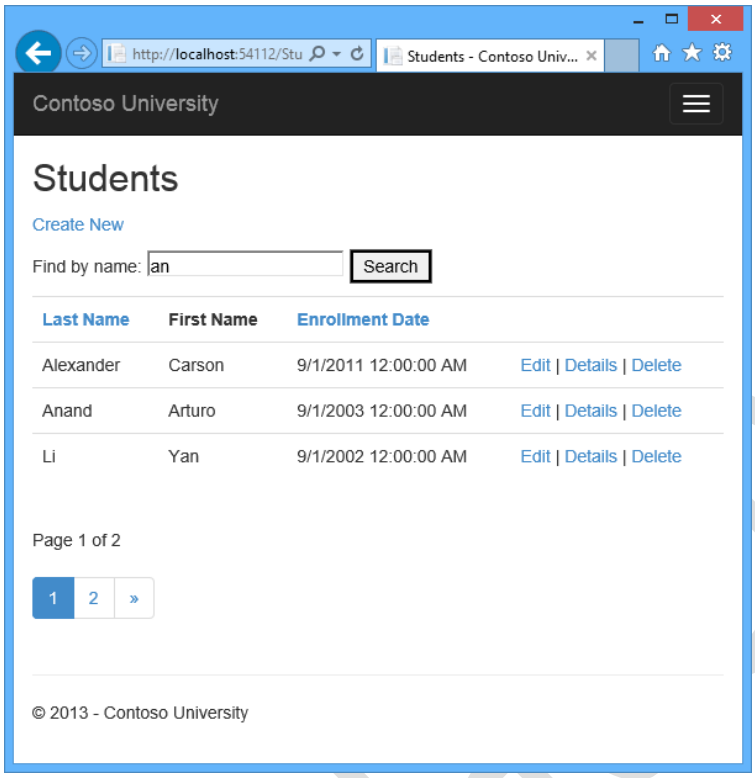
```
@Html.PagedListPager(Model, page => Url.Action("Index", new { page }))
```

PagedListPager helper تعدادی گزینه همچون URL و styling ارائه می دهد که شما می توانید سفارشی تنظیم کنید.

صفحه را اجرا کنید.



لینک های صفحه بندی را با ترتیب های مرتب سازی مختلف امتحان کنید و از عملکرد صحیح صفحه بندی اطمینان حاصل نمایید. سپس یک عبارت جستجو (search string) وارد کرده و مجددا paging را تکرار کنید تا مطمئن شوید که paging با مرتب سازی (sorting) و امکان فیلترینگ (filtering) نیز درست کار می کند.



ایجاد یک صفحه به نام About که آمار و جزئیات مربوط به دانشجویان را نمایش می دهد

در این صفحه تعداد دانش آموزانی که برای هر تاریخ ثبت نام (enrollment date) کرده اند، نمایش داده می شود. این لازمه ی گروه بندی و انجام محاسبات ساده بر روی گروه ها می باشد. برای انجام این کار، لازم است اقدامات زیر را انجام دهید:

یک کلاس view model ویژه ی داده هایی که باید به view ارسال شود، ایجاد کنید.

متد About را در کنترلگر Home اصلاح کنید.

در مرحله ی آخر About view را اصلاح کنید.

ایجاد View Model

یک پوشه به نام ViewModels در پوشه ی پروژه ایجاد کنید. حال به داخل آن پوشه یک فایل کلاس EnrollmentDateGroup.cs اضافه نموده و کد زیر را جایگزین کد template زیر کنید:

```
    using System;
using System.ComponentModel.DataAnnotations;
namespace ContosoUniversity.ViewModels
{
  public class EnrollmentDateGroup
  {
    [DataType(DataType.Date)]
    public DateTime? EnrollmentDate { get; set; }

    public int StudentCount { get; set; }
  }
}
```

اصلاح کنترلگر Home

دستورهای using زیر را به بالای محتویات فایل HomeController.cs اضافه کنید:

```
using ContosoUniversity.DAL;
using ContosoUniversity.ViewModels;
```

یک متغیر کلاس درست پس از کاراکتر باز "{ " ویژه ی database context اضافه کنید.

```
public class HomeController : Controller
{
  private SchoolContext db = new SchoolContext();
```

متد About را با کد زیر جایگزین کنید:

```
public ActionResult About()
{
  IQueryable<EnrollmentDateGroup> data = from student in db.Students
        group student by student.EnrollmentDate into dateGroup
        select new EnrollmentDateGroup()
        {
          EnrollmentDate = dateGroup.Key,
          StudentCount = dateGroup.Count()
        };
  return View(data.ToList());
}
```

دستور LINQ موجودیت های student را بر اساس تاریخ ثبت نام (enrollment date) گروه بندی می کند، تعداد موجودیت ها را در هر گروه محاسبه کرده، سپس نتایج را درون یک مجموعه EnrollmentDateGroup view model object ذخیره می کند.

یک متد Dispose اضافه نمایید:

```
protected override void Dispose(bool disposing)
    {
      db.Dispose();
      base.Dispose(disposing);
    }
```

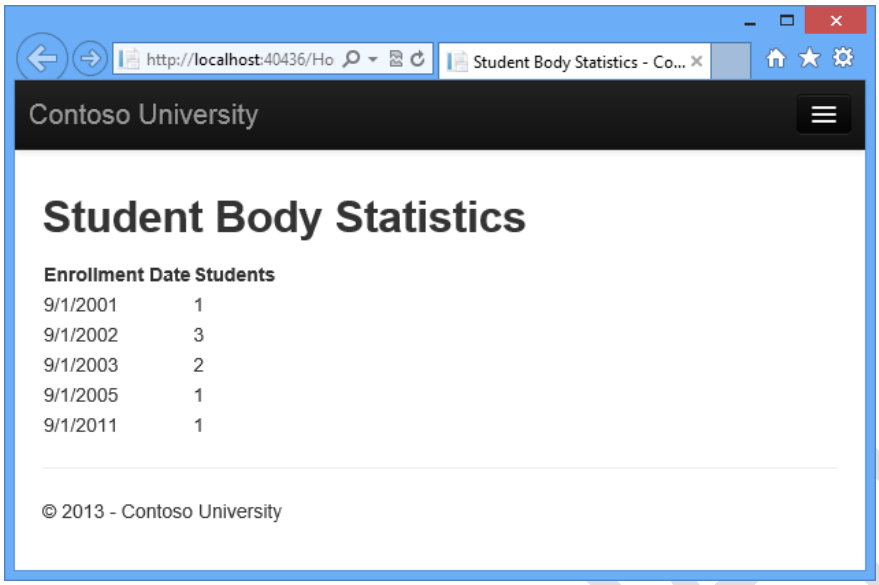
اصلاح About View

کد زیر را جایگزین را با کد موجود در فایل Views\Home\About.cshtml جایگزین کنید:

```
@model IEnumerable<ContosoUniversity.ViewModels.EnrollmentDateGroup>
@{
  ViewBag.Title = "Student Body Statistics";
}
<h2>Student Body Statistics</h2>
<table>
   <tr>
     <th>
       Enrollment Date
     </th>
     <th>
       Students
     </th>
   </tr>
   @foreach (var item in Model)
   {
     <tr>
       <td>
         @Html.DisplayFor(modelItem => item.EnrollmentDate)
       </td>
       <td>
         @item.StudentCount
       </td>
     </tr>
   }
</table>
```

برنامه را اجرا کرده و لینک About را کلیک کنید. تعداد دانشجویان که در هر تاریخ ثبت نام ثبت نام کرده اند

در جدول به نمایش گذاشته می شود:



چکیده

در این فصل کارهایی همچون ایجاد یک data model ، پیاده سازی عملیات ساده ی CRUD و قابلیت های مرتب سازی، فیلتر کردن، صفحه بندی و گروه بندی پرداختیم. در مبحث بعدی با بسط data model مباحث پیچیده تری را تشریح خواهیم کرد.