**Server Side Pagination With Datatables**

03 September 2019

|  |
| --- |
| **Goals:**  **In "ASP.NET CORE Web Application",**   * Create list of objects to be used as test data. * Display records in datatables * Use page ids of datatables to fetch data from data source. |

|  |
| --- |
| **Keys:**  Paging, Pagination  Datatables  Create sample objects in startup.  Static keyword usage. |

|  |
| --- |
| **Assumptions:**  Your project is already open in Visual Studio.  Only the Details page will be ready at the end of the example. Create, Update and Delete pages have been ignored to make video shorter. |

|  |
| --- |
| **Scenario:**  Prepare sample data to be displayed in tables.  Make necessary changes to enable pagination via datatables and ajax calls.  Fetch and Display example students with paging feature.  Listen RUD buttons events and determine id of clicked button of datatable records. |

|  |
| --- |
| **Result:**  Now the result of the example will be displayed. |

**Summary of Steps**

[**Part 1: Prepare Sample Data** 3](#_Toc18391466)

[1. Create Student class as an example object class 3](#_Toc18391467)

[2. Create Data folder in project. 3](#_Toc18391468)

[3. Create StudentContext class in Data folder as if a data source to keep Student objects. 3](#_Toc18391469)

[4. Update Startup.cs to initialize data at application start 3](#_Toc18391470)

[**Part 2: Preparing Controller and View** 4](#_Toc18391471)

[1. Create StudentsController 4](#_Toc18391472)

[2. Create Index View of Students 4](#_Toc18391473)

[3. **Install latest dataTables files** as client-side library 4](#_Toc18391474)

[**4.** **Add dataTables CSS into Students/Index view** 4](#_Toc18391475)

[5. **Update view to display table using dataTables.** 4](#_Toc18391476)

[6. Add dataTables.js into Scripts section of the view file 4](#_Toc18391477)

[7. Add js code to initialize datatable into section Scripts 5](#_Toc18391478)

[8. Add link to Home page to be able to display Students/Index page 5](#_Toc18391479)

[9. Update Details Action and Create Details View of Students 5](#_Toc18391480)

[10. RunProject. 5](#_Toc18391481)

[**Part 3: Use Server Side Paging Feature In Datatable** 5](#_Toc18391482)

[1. Stop the application if its still running. 5](#_Toc18391483)

[2. Create action in StudentsController to get filtered records related to paging requests. 5](#_Toc18391484)

[3. Update Students/Index to display student list in Home page 6](#_Toc18391485)

[4. Optional: Update site.css to enable loading spinner during data fetching calls. 10](#_Toc18391486)

[5. RunProject. 11](#_Toc18391487)

**Steps**

# **Part 1: Prepare Sample Data**

We will create Student class and StudentContext to keep sample Student objects in StudentList. Also we will update startup.cs to create initial students at application startup. The students will be used in datatable in Part 2 and Part 3.

## Create Student class as an example object class

* + - 1. **Right click on Models folder + mouseover to Add + Select Class**
      2. Enter the name of the class as Student
      3. In Models/Student.cs that we have just created, enter the properties of Student class as below

|  |
| --- |
| public class Student  {  public int Id { get; set; }  public string Firstname { get; set; }  public string Lastname { get; set; }  public DateTime CreatedDate { get; set; }  } |

## Create Data folder in project.

* + - 1. Right click on Project folder + mouseover to Add + Select New Folder
      2. Enter the name of the folder as Data

## Create StudentContext class in Data folder as if a data source to keep Student objects.

* + - 1. Right click on Data folder + mouseover to Add + Select class
      2. Enter the name of the class as StudentContext.cs
      3. Replace code of StudentContext class with the code below

|  |
| --- |
| public static class StudentContext  {  //StudentList is static to be able to reach in application scope  public static List<Student> StudentList { get; set; }  //Creates “studentCount” students and adds into student list  public static void InitStudentList(int studentCount)  {  StudentList = new List<Student>();  for (int i = 1; i < studentCount + 1; i++)  {  StudentList.Add(  new Student()  {  Id = i,  Firstname = "Firstname" + i,  Lastname = "Lastname" + i,  CreatedDate = DateTime.Now  }  );  }  }  } |

* + - 1. Add using statement for the Student class.

## Update Startup.cs to initialize data at application start

* + - 1. Open /Startup.cs file
      2. Add the code line below to create 1000 students

|  |
| --- |
| public Startup(IConfiguration configuration)  {  Configuration = configuration;  **Data.StudentContext.InitStudentList(1000);//Add this line to create 1000 students**  } |

# **Part 2: Preparing Controller and View**

## Create StudentsController

* + - 1. Right click on Controllers folder
      2. Select Add / Controller
      3. Select MvcController with read/write actions + Click Add
      4. Enter name of the class as StudentsController
      5. Update the Index action in StudentsController.cs as below

|  |
| --- |
| public ActionResult Index()  {  return View(Data.StudentContext.StudentList);// Send students to view  } |

## Create Index View of Students

* + - 1. Right click on Index action in StudentsController.cs
      2. Select add / view
      3. Select template as List
      4. Select model class as Student + Click Add

## **Install latest dataTables files** as client-side library

* + - 1. right click on wwwroot/lib
      2. select add clientside library
      3. enter datatables@ into Library field.
      4. select latest datatables version (1.10.19 current version in this example)
      5. click Install.

## **Add dataTables CSS into Students/Index view**

* + - 1. Open Views/Students/Index.cshtml
      2. Drag and Drop "**jquery.dataTables.css**" file from wwroot/lib after the table definition.

## **Update view to display table using dataTables.**

* + - 1. Open view file /Views/Students/Index.cshtml
      2. Find table definition.
      3. Update header before the table definition as <h1>Students</h1>
      4. Update id of the table as <table id="studentTable" class="table table-striped">
      5. Add code below to the bottom of the view file.

|  |
| --- |
| @section Scripts{  } |

## Add dataTables.js into Scripts section of the view file

* + - 1. Drag and Drop "**jquery.dataTables.js**" file from wwroot/lib into the section Scripts.

Section Scripts will be like below.

|  |
| --- |
| @section Scripts{  <script src="~/lib/datatables/js/jquery.dataTables.js"></script>  } |

## Add js code to initialize datatable into section Scripts

|  |
| --- |
| @section Scripts{  //script for datatables js.  <script src="~/lib/datatables/js/jquery.dataTables.js"></script>  //script to initialize datatable.  <script>  $(document).ready(function () {  $('#studentTable').DataTable();//studentTable is the id of the table to be displayed as dataTable  });  </script>  } |

## Add link to Home page to be able to display Students/Index page

* + - 1. Open Views/Home/Index.cshtml
      2. Add code below into the page.

|  |
| --- |
| <h1>Datatables With Pagination At Server Side</h1>  <**a** **asp-controller**="Students" **asp-action**="Index">Open Students Page</**a**> |

## Update Details Action and Create Details View of Students

* + - 1. Find the Details action in StudentsController.cs
      2. Replace the Details action in StudentsController.cs as below to display Student data.

|  |
| --- |
| public ActionResult Details(int id)  {  Student s = Data.StudentContext.StudentList.FirstOrDefault(a => a.Id == id);  return View(s);  } |

* + - 1. Right click on Details action in StudentsController
      2. Select add / view
      3. Select template as Details
      4. Select model class as Student + Click Add

## RunProject.

# **Part 3: Use Server Side Paging Feature In Datatable**

## Stop the application if its still running.

## Create action in StudentsController to get filtered records related to paging requests.

* + - 1. Open Controllers/StudentsControllers.cs
      2. Find Index action and delete the parameter that sends StudentList to view.
      3. Add new action to be used by datatable ajax requests.

|  |
| --- |
| //Action to be called by js in details page when search, sort or page numbers clicked  // Search is applied only to Firstname and Lastname properties  public JsonResult GetFilteredItems()  {  System.Threading.Thread.Sleep(2000);//Used to display loading spinner in demonstration, remove this line in production  int draw = Convert.ToInt32(Request.Query["draw"]);  // Data to be skipped ,  // if 0 first "length" records will be fetched  // if 1 second "length" of records will be fethced ...  int start = Convert.ToInt32(Request.Query["start"]);  // Records count to be fetched after skip  int length = Convert.ToInt32(Request.Query["length"]);    // Getting Sort Column Name  int sortColumnIdx = Convert.ToInt32(Request.Query["order[0][column]"]);  string sortColumnName = Request.Query["columns[" + sortColumnIdx + "][name]"];    // Sort Column Direction  string sortColumnDirection = Request.Query["order[0][dir]"];    // Search Value  string searchValue = Request.Query["search[value]"].FirstOrDefault()?.Trim();  // Records Count matching search criteria  int recordsFilteredCount =  Data.StudentContext.StudentList  .Where(a => a.Lastname.Contains(searchValue) || a.Firstname.Contains(searchValue))  .Count();  // Total Records Count  int recordsTotalCount = Data.StudentContext.StudentList.Count();  // Filtered & Sorted & Paged data to be sent from server to view  List<Student> filteredData = null;  if (sortColumnDirection == "asc")  {  filteredData =  Data.StudentContext.StudentList  .Where(a => a.Lastname.Contains(searchValue) || a.Firstname.Contains(searchValue))  .OrderBy(x => x.GetType().GetProperty(sortColumnName).GetValue(x))//Sort by sortColumn  .Skip(start)  .Take(length)  .ToList<Student>();  }  else  {  filteredData =  Data.StudentContext.StudentList  .Where(a => a.Lastname.Contains(searchValue) || a.Firstname.Contains(searchValue))  .OrderByDescending(x => x.GetType().GetProperty(sortColumnName).GetValue(x))  .Skip(start)  .Take(length)  .ToList<Student>();  }  // Send data  return Json(  new {  data = filteredData,  draw = Request.Query["draw"],  recordsFiltered = recordsFilteredCount,  recordsTotal = recordsTotalCount  }  );  } |

## Update Students/Index to display student list in Home page

* + - 1. Open view file Views/Students/Index.cshtml
      2. Find table definition.
      3. Remove table body including <tbody></tbody> tags. (Table body will be provided by ajax request given in next step)
      4. Enter “Actions” text between last(empty) th tags to be used for crud actions <th></th>
      5. Delete script to initialize datatable. We will add different code in next step. Be careful, do not delete the script link to datatables js library.

|  |
| --- |
| @section Scripts{  //script for datatables js.  <script src="~/lib/datatables/js/jquery.dataTables.js"></script>  ~~//script to initialize datatable.~~  ~~<script>~~  ~~$(document).ready(function () {~~  ~~$('#studentTable').DataTable();//studentTable is the id of the table to be displayed as dataTable~~  ~~});~~  ~~</script>~~  } |

* + - 1. Add scripts to initialize and manage ajax requests as below.

This script has been copied from https://datatables.net/examples/server\_side/pipeline.html Although it seems complex, very few lines have been changed. Updated parts have been highlighted with yellow with comments.

|  |
| --- |
| <script>  //  // Pipelining function for DataTables. To be used to the `ajax` option of DataTables  // Copied from https://datatables.net/examples/server\_side/pipeline.html  //  $.fn.dataTable.pipeline = function (opts) {  // Configuration options  var conf = $.extend({  pages: 5, // number of pages to cache. That means action(url) will be called in 1st, 6th, 11th ... pages  url: 'Students/GetFilteredItems', // url to controller action  data: null, // function or object with parameters to send to the server  method: 'GET' // Ajax HTTP method  }, opts);  // Private variables for storing the cache  var cacheLower = -1;  var cacheUpper = null;  var cacheLastRequest = null;  var cacheLastJson = null;  return function (request, drawCallback, settings) {  var ajax = false;  var requestStart = request.start;  var drawStart = request.start;  var requestLength = request.length;  var requestEnd = requestStart + requestLength;  if (settings.clearCache) {  // API requested that the cache be cleared  ajax = true;  settings.clearCache = false;  }  else if (cacheLower < 0 || requestStart < cacheLower || requestEnd > cacheUpper) {  // outside cached data - need to make a request  ajax = true;  }  else if (JSON.stringify(request.order) !== JSON.stringify(cacheLastRequest.order) ||  JSON.stringify(request.columns) !== JSON.stringify(cacheLastRequest.columns) ||  JSON.stringify(request.search) !== JSON.stringify(cacheLastRequest.search)  ) {  // properties changed (ordering, columns, searching)  ajax = true;  }  // Store the request for checking next time around  cacheLastRequest = $.extend(true, {}, request);  if (ajax) {  // Need data from the server  if (requestStart < cacheLower) {  requestStart = requestStart - (requestLength \* (conf.pages - 1));  if (requestStart < 0) {  requestStart = 0;  }  }  cacheLower = requestStart;  cacheUpper = requestStart + (requestLength \* conf.pages);  request.start = requestStart;  request.length = requestLength \* conf.pages;  // Provide the same `data` options as DataTables.  if (typeof conf.data === 'function') {  // As a function it is executed with the data object as an arg  // for manipulation. If an object is returned, it is used as the  // data object to submit  var d = conf.data(request);  if (d) {  $.extend(request, d);  }  }  else if ($.isPlainObject(conf.data)) {  // As an object, the data given extends the default  $.extend(request, conf.data);  }  settings.jqXHR = $.ajax({  "type": conf.method,  "url": conf.url,  "data": request,  "dataType": "json",  "cache": false,  "success": function (json) {  cacheLastJson = $.extend(true, {}, json);  if (cacheLower != drawStart) {  json.data.splice(0, drawStart - cacheLower);  }  if (requestLength >= -1) {  json.data.splice(requestLength, json.data.length);  }  drawCallback(json);  }  });  }  else {  json = $.extend(true, {}, cacheLastJson);  json.draw = request.draw; // Update the echo for each response  json.data.splice(0, requestStart - cacheLower);  json.data.splice(requestLength, json.data.length);  drawCallback(json);  }  }  };  // Register an API method that will empty the pipelined data, forcing an Ajax  // fetch on the next draw (i.e. `table.clearPipeline().draw()`)  // Copied from https://datatables.net/examples/server\_side/pipeline.html  $.fn.dataTable.Api.register('clearPipeline()', function () {  return this.iterator('table', function (settings) {  settings.clearCache = true;  });  });  //  // DataTables initialization  // Copied from https://datatables.net/examples/server\_side/pipeline.html  // Updated according to our data  //  $(document).ready(function () {  $('#studentTable').DataTable({  "processing": true,  "serverSide": true,  "searching": true,  "paging": true,  "ajax": $.fn.dataTable.pipeline({  url: 'Students/GetFilteredItems',  pages:5 //number of pages to cache  }),  //Column definitions are sent to action to be used in sorted column definition  //name parts are assigned as the exact property name to determine sort columns  // render definition has been given to display format of CreatedDate property  "columns": [  // For Student.Id  { "data": "id", "name": "Id" },  // For Student.Firstname  { "data": "firstname", "name": "Firstname" },  // For Student.Lastname  { "data": "lastname", "name": "Lastname" },  // For Student.CreatedDate  { "data": "createdDate", "name": "CreatedDate",  "render": function (data) {  var date = new Date(data);  return date.toLocaleString();  }  },  // Optional: Buttons For Action Listeners  {  'data': null,  'render': function (data, type, row) {  return '<button id="' + row.id + '" onclick="detailsClick(this)">Details</button>'  + '<button id="' + row.id + '" onclick="editClick(this)">Edit</button>'  + '<button id="' + row.id + '" onclick="deleteClick(this)">Delete</button>'  }  }  ],  language: {  processing: '<div class="spinner"></div>', // Optional to use loading spinner. Instead of it you can define a simple string.  zeroRecords: "No matching records found"  }  });  });  //Optional: Details button listener  function detailsClick(obj) {  //var rowID = $(obj).attr('id');  var studentId = $(obj).closest('tr').find('td:first').html();  alert("Id = " + studentId + " for details");  }  // Optional: Edit button listener  function editClick(obj) {  //var rowID = $(obj).attr('id');  var studentId = $(obj).closest('tr').find('td:first').html();  alert("Id = " + studentId + " for edit");  }  // Optional: Delete button listener  function deleteClick(obj) {  //var rowID = $(obj).attr('id');  var studentId = $(obj).closest('tr').find('td:first').html();  alert("Id = " + studentId + " for delete");  }  </script> |

## Optional: Update site.css to enable loading spinner during data fetching calls.

* + - 1. Open /wwwroot/site.css

|  |
| --- |
| /\*  Begin - Style of Centered spinner  Code from https://www.w3schools.com/howto/howto\_css\_loader.asp  \*/  .spinner {  position: absolute;  left: 50%;  top: 50%;  z-index: 1;  width: 150px;  height: 150px;  margin: -75px 0 0 -75px;  border: 16px solid #f3f3f3;  border-radius: 50%;  border-top: 16px solid #3498db;  width: 120px;  height: 120px;  -webkit-animation: spin 2s linear infinite;  animation: spin 2s linear infinite;  }  @-webkit-keyframes spin {  0% {  -webkit-transform: rotate(0deg);  }  100% {  -webkit-transform: rotate(360deg);  }  }  @keyframes spin {  0% {  transform: rotate(0deg);  }  100% {  transform: rotate(360deg);  }  }  /\* Add animation to "page content" \*/  .animate-bottom {  position: relative;  -webkit-animation-name: animatebottom;  -webkit-animation-duration: 1s;  animation-name: animatebottom;  animation-duration: 1s  }  @-webkit-keyframes animatebottom {  from {  bottom: -100px;  opacity: 0  }  to {  bottom: 0px;  opacity: 1  }  }  @keyframes animatebottom {  from {  bottom: -100px;  opacity: 0  }  to {  bottom: 0;  opacity: 1  }  }  /\* End - Style of Centered spinner \*/ |

* + - 1. Add code below to the bottom of the site.css file for loading spinner.

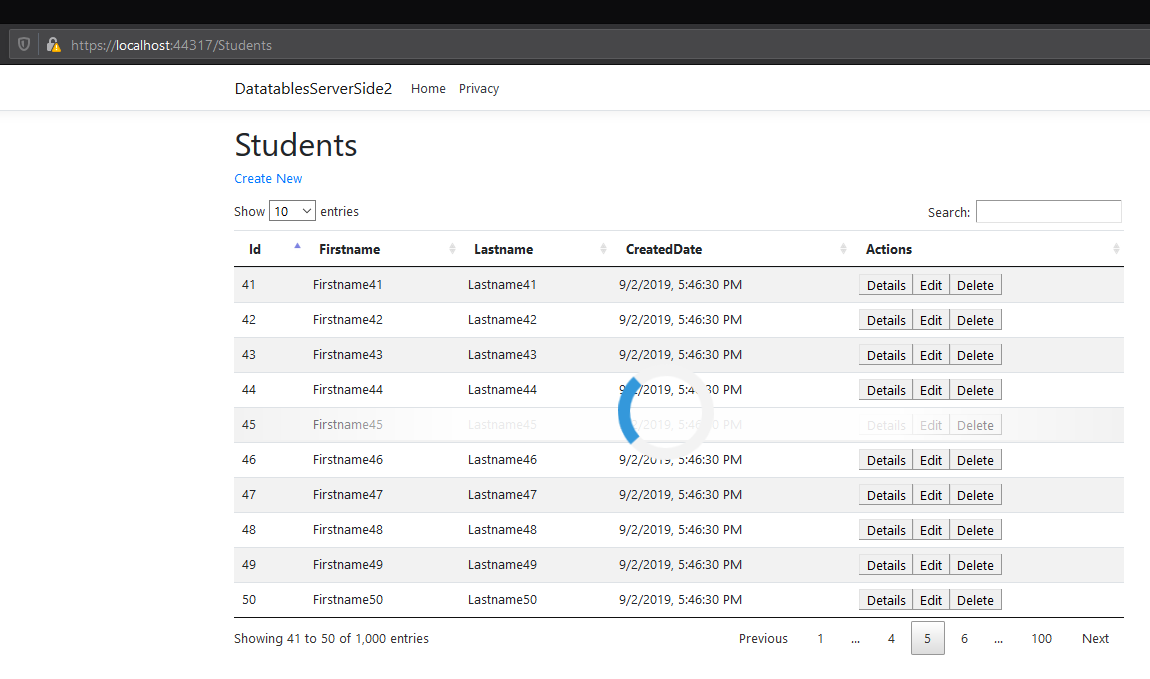
## 

## RunProject.

**Run** the Project by clicking IIS Express ( or by pressing key F5 )

See the result as below:

Figure :Result screen during the data load after click on Page 6



## Update detailsClick js function at Views/Students/Index.cshmtl to display Details page when button clicked.

|  |
| --- |
| //var rowID = $(obj).attr('id');  // Get Id of clicked student  var studentId = $(obj).closest('tr').find('td:first').html();  //Go to details page of student  window.location.replace("Students/Details/" + studentId); |