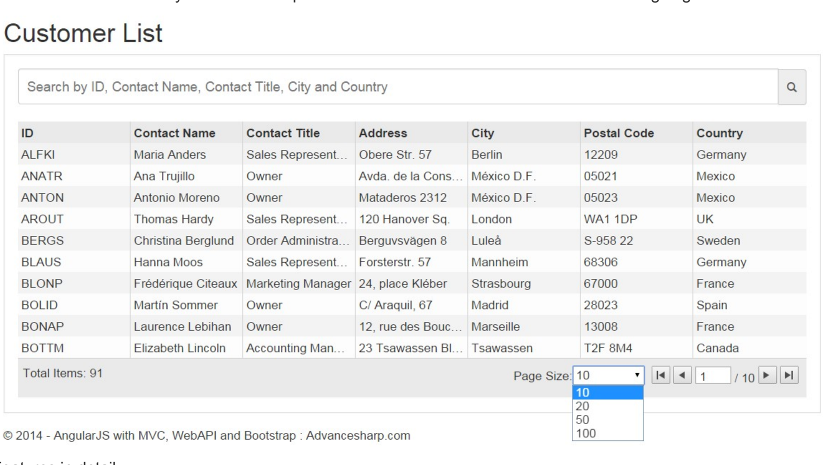
ng-grid search sort paging with AngularJs WebAPI and MVC ASP.Net

ng-grid external filtering *ng-grid angularjs webapi and mvc* *mvc webapi and angularjs* *AngularJS with Web API* *AngularJS, Web API and EntityFramework* *Data-binding in AngularJS by consuming ASP.NET Web API* *Working example of AngularJS* *$resource searching*

We are going to use ng-grid with feature like searching, sorting and paging of records, we will use paginated data, means pull only as many records as we have to show into our ng-grid. We already discussed about the AngularJS application structure in our previous article see ASP.Net MVC Web API AngularJS and Bootstrap application structure. We will use Entity Framework and WebAPI to search record from database through the service and WebAPI which we already created in our previous article. Here is the screen which we are going to create:



Features in detail:

1. Search on load of page, first time
2. Search when user type anything in textbox
3. Sorting on clicking the grid header
4. Paging
5. Change page size to see more or less records

First of all we need to add ng-grid and ng-grid-flexible-height package to our application:

1. Install-Package ng-grid
2. Download ng-grid-flexible-height.js from http://cdnjs.com/libraries/ng-grid and save to scripts folder

Let's add these JavaScript and ng-grid CSS file to our bundle file, open /App_Start/BundleConfig.cs and add following file in indicated bundles:

```
// in angularjs section
"~/Scripts/ng-grid.js",
"~/Scripts/ng-grid-flexible-height.js"

// Content/css section
"~/Content/ng-grid.css"

// Change app.js first line to
var app = angular.module('app', ['ngRoute', 'ui.bootstrap', 'ngGrid']);

```

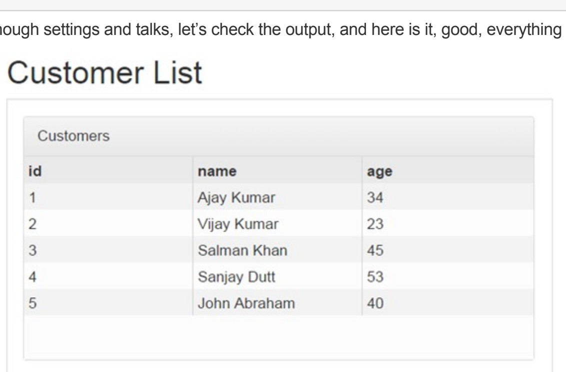
We added some packages and added some JavaScript files to our bundling section so let's first try to check whatever we added is correct by using hard coded array of customer's data, so change the customer controller like this

```
app.controller('CustomerController',
   function ($scope, CustomerService, $routeParams, $log) {
       $scope.hardCodedCustomers = [
                  { id: 1, name: "Ajay Kumar", age: 34 },
                  {id: 2, name: "Vijay Kumar", age: 23},
                  {id: 3, name: "Salman Khan", age: 45},
                  {id: 4, name: "Sanjay Dutt", age: 53},
                  {id: 5, name: "John Abraham", age: 40}];

         $scope.gridOptions = { data: 'hardCodedCustomers' };
}

//And HTML changes
<h1>Customer List</h1>
<div class="gridStyle" ng-grid="gridOptions"></div>

```

Before running the application, let's change the CSS so it can show properly the grid height otherwise it will show in only one line if there is no records. Open ng-grid.css and change class to min-height to 200px

```
.ngViewport {
  overflow: auto;
  min-height: 200px;
}

```

Enough settings and talks, let's check the output, and here is it, good, everything is working as it should be



WebAPI: As we decided to show the customers from northwind database so we will write code to search customers table and return records, if you can see the first image, we have to search the table on different columns, customer Id, contact name, contact title, city and country, if you want to add other columns you can add one we done, we will also need to add the code to get the number of records on the basis of search text.

```
[HttpGet]
public PagedList GetCustomers(string searchtext, int page = 1, int pageSize = 10, string sortBy =
"CustomerID", string sortDirection = "asc")
{
    var pagedRecord = new PagedList();

    pagedRecord.Content = db.Customers
                .Where(x => searchtext == null ||
                        (( x.CustomerID.Contains(searchtext)) ||
                        ( x.ContactName.Contains(searchtext)) ||
                        ( x.ContactTitle.Contains(searchtext)) ||
                        ( x.City.Contains(searchtext)) ||
                        ( x.Country.Contains(searchtext))
                    ))
                .OrderBy(sortBy + " " + sortDirection)
                .Skip((page -1 ) * pageSize)
                .Take(pageSize)
                .ToList();

    // Count
    pagedRecord.TotalRecords = db.Customers
                .Where(x => searchtext == null ||
                        ( (x.CustomerID.Contains(searchtext)) ||
                        (x.ContactName.Contains(searchtext)) ||
                        (x.ContactTitle.Contains(searchtext)) ||
                        (x.City.Contains(searchtext)) ||
                        (x.Country.Contains(searchtext))
                    )).Count();

    pagedRecord.CurrentPage = page;
    pagedRecord.PageSize = pageSize;

    return pagedRecord;
}

```

If you can see closely, I used order by parameter as string `.OrderBy(sortBy + " " + sortDirection)` normally which will not accept, to make it working you need LINQ dynamic query package, so add it in our project by using view => Other Windows => Package Manager Console "**Install-Package System.Linq.Dynamic.Library**". One more thing, PagedList class, we need to create a class In Models folder of MVC, here is the detail

```
public class PagedList
{
    public IList Content { get; set; }

    public Int32 CurrentPage { get; set; }
    public Int32 PageSize { get; set; }
    public int TotalRecords { get; set; }

    public int TotalPages
    {
        get { return (int)Math.Ceiling((decimal)TotalRecords / PageSize); }
    }
}

```

In the second part we have not used order by because we have to count only the total number of records on the basis of current search so no need to use the order by, unnecessarily it will slow down our process.

Service (customer.js): first we will try to understand our service code step by step, so create a property data in service

```
data: {
        currentcustomer: {},
        customers: [],
        selected:[],
        totalPages: 0,

        filterOptions: {
            filterText: '',
            externalFilter: 'searchText',
            useExternalFilter: true
        },
        sortOptions: {
            fields: ["CustomerID"],
            directions: ["desc"]
        },
        pagingOptions: {
            pageSizes: [10, 20, 50, 100],
            pageSize: 10,
            currentPage: 1
        }
    }

```

Here we defined different property in data to keep our data

- currentcustomer: an array to keep current customer
- customers: an array to keep the list of customer to bind in grid
- selected: an array of selected customers, we can select multiple records
- totalPages: total number of records for our search
- filterOptions: it the section for search feature, external search is true and defined the field name which will be used to search
- sortOptions: we provided default search column and direction
- pagingOptions:
  - pageSizes: [10, 20, 50, 100] to show in drop down to change the number of records to show in grid
  - pageSize: default page size, in our example it is 10
  - currentPage: default is 1

Let's explore the find method in our service and try to understand:

```
find: function () {

    var params = {
        searchtext: service.data.filterOptions.filterText,
        page: service.data.pagingOptions.currentPage,
        pageSize: service.data.pagingOptions.pageSize,
        sortBy: service.data.sortOptions.fields[0],
        sortDirection: service.data.sortOptions.directions[0]
    };

    var deferred = $q.defer();
    $http.get(baseUrl, { params: params })
    .success(function (data) {
        service.data.customers = data;
        deferred.resolve(data);
    }).error(function () {
        deferred.reject();
    });
    return deferred.promise;
}

```

Whatever we defined in our paging, sorting and filtering option used them here as parameters say params and finally used into get function as $http.get(baseUrl, { params: params }) and on success we assigned the return data from WebAPI we assigned to customers property `service.data.customers = data;`

Here is the complete service file customer.js

```
app.factory('CustomerService', function ($http, $q, $log, $rootScope) {

    var baseUrl = '/api/customer';

    var service = {
        data: {
                currentcustomer: {},
                customers: [],
                selected:[],
                totalPages: 0,

                filterOptions: {
                    filterText: '',
                    externalFilter: 'searchText',
                    useExternalFilter: true
                },
                sortOptions: {
                    fields: ["CustomerID"],
                    directions: ["desc"]
                },
                pagingOptions: {
                    pageSizes: [10, 20, 50, 100],
                    pageSize: 10,
                    currentPage: 1
                }
            },

            find: function () {
                var params = {
                    searchtext: service.data.filterOptions.filterText,
                    page: service.data.pagingOptions.currentPage,
                    pageSize: service.data.pagingOptions.pageSize,
                    sortBy: service.data.sortOptions.fields[0],
                    sortDirection: service.data.sortOptions.directions[0]
                };

                var deferred = $q.defer();
                $http.get(baseUrl, { params: params })
                .success(function (data) {
                    service.data.customers = data;
                    deferred.resolve(data);
                }).error(function () {
                    deferred.reject();
                });
                return deferred.promise;
            }
    }

    service.find();
    return service;
});

```

We completed our service to get data from database not time to bind the data into grid, here is our complete code in customer.js controller file

```
app.controller('CustomerController', function ($scope, CustomerService, $routeParams, $log) {
    $scope.data = CustomerService.data;

    $scope.$watch('data.sortOptions', function (newVal, oldVal) {
        $log.log("sortOptions changed: " + newVal);
        if (newVal !== oldVal) {
            $scope.data.pagingOptions.currentPage = 1;
            CustomerService.find();
        }
    }, true);

    $scope.$watch('data.filterOptions', function (newVal, oldVal) {
        $log.log("filters changed: " + newVal);
        if (newVal !== oldVal) {
            $scope.data.pagingOptions.currentPage = 1;
            CustomerService.find();
        }
    }, true);

    $scope.$watch('data.pagingOptions', function (newVal, oldVal) {
        $log.log("page changed: " + newVal);
        if (newVal !== oldVal) {
            CustomerService.find();
        }
    }, true);

    $scope.gridOptions = {
        data: 'data.customers.content',
        showFilter: false,
        multiSelect: false,
        selectedItems: $scope.data.selected,
        enablePaging: true,
        showFooter: true,
        totalServerItems: 'data.customers.totalRecords',
        pagingOptions: $scope.data.pagingOptions,
        filterOptions: $scope.data.filterOptions,
        useExternalSorting: true,
        sortInfo: $scope.data.sortOptions,
        plugins: [new ngGridFlexibleHeightPlugin()],
        columnDefs: [
                    { field: 'customerID', displayName: 'ID' },
                    { field: 'contactName', displayName: 'Contact Name' },
                    { field: 'contactTitle', displayName: 'Contact Title' },
                    { field: 'address', displayName: 'Address' },
                    { field: 'city', displayName: 'City' },
                    { field: 'postalCode', displayName: 'Postal Code' },
                    { field: 'country', displayName: 'Country' }
        ],
        afterSelectionChange: function (selection, event) {
            $log.log("selection: " + selection.entity.CustomerID);
            // $location.path("comments/" + selection.entity.commentId);
        }
    };

});

```

Let's divide it and try to understand

```
 // whatever the data will return from service we will assign to data in scope
 $scope.data = CustomerService.data;

$scope.$watch('data.sortOptions', function (newVal, oldVal) {
    $log.log("sortOptions changed: " + newVal);
    if (newVal !== oldVal) {
        $scope.data.pagingOptions.currentPage = 1;
        CustomerService.find();
    }
}, true);

```

First watch is to keep watching if there is any sort action then this will be call similarly we used for filter and paging. In every section we called `CustomerService.find()`, rest of the code it to bind the grid, our final step is to change our HTML in customerList.html file and here is it

```
<div>
    <h1>Customer List</h1>

    <div class="well" style="background-image: none">
        <div class="input-group">
            <input type="text"
                name="searchText"
                ng-model="data.filterOptions.filterText"
                class="form-control"
                placeholder="Search by ID, Contact Name, Contact Title, City and Country" />
            <span class="input-group-addon glyphicon glyphicon-search"></span>
        </div>
        <br />
        <div class="gridStyle" ng-grid="gridOptions"></div>
    </div>
</div>

```

There is nothing new which you cannot understand, except the very last div with ng-grid="gridOptions" which will show the data from the service.

Whatever we show in our first image, everything is working now, you can search by typing anything into the text box, you change the page size, page number and sort on any column of the grid, so we completed our article.

DOWNLOAD

See previous article ASP.Net MVC Web API AngularJS and Bootstrap application structure

Having 10+ years of experience in Microsoft Technologies (C#, ASP.Net, MVC and SQL Server). Worked with Metaoption LLC, for more than 8 years and still with the same company. Always ready to learn new technologies and tricks.
ng-grid angularjs webapi mvc
By Ali Adravi On 30 Nov, 14 Viewed: 15,521

Other blogs you may like

Angularjs CRUD with Web API, Entity Framework & Bootstrap modal popup part 2

Add/Edid Customer in Modal Popup --- In previous article we discussed how to search, delete and show detail of customer and contact list. In this article we will try to open the modal popup to add new customer and edit an existing customer record. I divided article in two parts because of the...
— By Ali Adravi On 29 Jul 2015 Viewed: 5,913

Custom Directives in AngularJS with example and demo

Angularjs custom directives are the common to avoid the duplicate code, it is easy powerful and really nice. There are already too many directive have been created by experts but as a developer we need to write our own to achieve our specific goal. ng-model, ng-if, ng-show, ng-repeat all these are...
— By Ali Adravi On 29 Jul 2015 Viewed: 979

Angularjs CRUD with Web API, Entity Framework & Bootstrap modal popup part 1

Search sort Insert update and delete are the basic features we need to learn first to learn any language, In this article we will try to create these features with Web API, bootstrap modal popup and ui.router. We will use enetity framework code first to save and retrieve data from database. We...
— By Ali Adravi On 28 Jul 2015 Viewed: 11,335

Angularjs Progress bar directive for entire application

Progress bar for Angularjs application by using directives is simple and real easy. I was googling and could not found any directive way, if you will check most of the people says before call the http method open the progress bas and close once you get the result or get any error. Do you think it...
— By Ali Adravi On 24 Jul 2015 Viewed: 5,907

Apply angularjs animation in 2 minutes

Angularjs animation is more than easy, even you don't have write a single line of code any kind of animation. In this demo I am going to show how you can animate the list of item coming from left, rotating and flying list items by just adding ngAnimation dependency and some copy and past of css and...
— By Ali Adravi On 21 Jul 2015 Viewed: 1,153

Post A Blog
Write A Blog
ASP.Net 45
MVC 35
GridView 5
C# 36
DataTable 3
Ajax 9
LINQ To SQL 3
SQL Server 32
Angularjs 18
Web API 4
JavaScript 13
jQuery 8
HTML 8
CSS 12
XML 6
OOPs 13
Web.Config 2
Routing 2