**Talkday Web App (Walk through)**

The Talkday Web App runs on ExpressJS + Kendo UI, so you will need to ensure that your NodeJS environments have been set up according to the pre-requisites required.

Once your environment has been set up, you just need to follow the steps below to get your first Kendo UI Grid up and running.

1. Open up your file system and browse to a local directory where you want to download the workshop project.
2. Go to <https://github.com/sforsherman/talkday-web-workshop> to download the project files onto your local system.  
     
   Note: You can start running the application in the NodeJS environment by opening either your Terminal client (on mac OS), or Command Prompt (on Windows), go to the path where you have extracted your project, and at the root of the project (e.g. C:\workshop\talk-day-web-workshop\), you can enter the following command: npm start. This will cause the project to run and with the nodemon package installed, you should be able to monitor changes as you are making it. Between the mac OS and Windows environment, it is helpful to note that in the mac environment, the project will always auto-recompile whenever changes in the project is detected. However on the windows environment, sometimes you might need to type a ‘rs’ in the command line interface to execute the re-compilation.  
     
   To view the web application in your browser, simply go to <http://localhost:2080/users/login>. The login details are as follows:  
     
   User Id: **sforsherman**  
   Password: **9JdjmbaO**
3. Launch your prefer IDE as mentioned in the pre-requisites, or any JS-capable IDE.

|  |
| --- |
| Exercise 1: Working with Grids |

Once the project is launched in your IDE, browse to views/event/dashboard.hbs.

1. We start by defining our <div /> area where the Grid will be displayed:  
     
   <div id="grid-events" style="margin-bottom: 15px;"></div>
2. Next, we will specify the Grid’s initialization within our script tag at Line 70:  
     
   $('#grid-events').kendoGrid({

dataSource: {

// We will specify our datasource properties here.

},

height: 550,

groupable: true,

sortable: true,

pageable: {

refresh: true,

pageSizes: true,

buttonCount: 5

},

columns: [

// We will specify our columns here.

]

});

1. To initialize the Grid control, we start by specifying which HTML element should be used for the Grid. This is done using the $([element\_id]).kendoGrid() method.
2. height – this property will determine the height of the entire Grid control.
3. groupable – accepts true / false. When set to true, a grouping header section will appear above the Grid view.
4. sortable – accepts true / false. When set to true, all columns in the Grid will be able to sort by ascending / descending sequence.
5. pageable – you can specify a number of different properties to this object to specify how the pagination feature will work:
   1. refresh – accepts true / false. This property will specify if the Grid will be able to refresh itself every time a page is triggered.
   2. pageSizes – accepts true / false. This property will determine if specifying pageSizes will work
   3. buttonCount – accepts an integer value. This property will determine the number of page numbers to appear in the pagination section of the Grid view.
6. You can refer to the Grid’s API for more details of the different properties and configurations available: <http://docs.telerik.com/kendo-ui/api/javascript/ui/grid>
7. Next, we will specify our dataSource attributes:  
     
   transport: {

read: "{{baseUri}}/api/events/get\_all\_events"

},

pageSize: 20

1. Place these properties within the dataSource: { … } section.
2. When working with the dataSource object, there are a number of different properties that you can specify, for a full list of properties, you can check out our documentation here: <http://docs.telerik.com/kendo-ui/api/javascript/data/datasource>
3. Within the transport object, this is where we can specify the various operations that the Grid can perform, usually used with inline editing:
   1. read – reading data from the specified URL
   2. update – updating the data to your remote service
   3. remove – removing a single record from the Grid
   4. create – creating a new record
4. In the dataSource object, this is where we also specify the pageSize attribute.
5. Now we can start defining the columns that we want to have in our Grid control.  
     
   {

field: "eventName",

title: "Event Name"

},

{

field: "\_eventStatus.statusName",  
title: "Status",  
width: 120

}

* 1. When working with columns, there are a number of basic properties that you can work with most of the time, those properties include:
     1. field – refers to the name of the field in the actual schema
     2. title – a string value for displaying in the header of the column
     3. width – the width of the column. If not specified, the column will auto-resize itself according to its content.
     4. headerTemplate – a HTML-driven template for the column’s header
     5. template – a HTML-driven template for the column’s content
  2. To get a full list of the configurations, you can check out the documentation here: <http://docs.telerik.com/kendo-ui/api/javascript/ui/grid#configuration-columns>

1. Before the eventName column, we are going to add a custom column which will present us with a colorCode field, but with dynamic CSS styling:  
     
   {

template: "<div style='text-align: center; border-radius: 5px; margin: 8px; height: 50px; width: 50px; background-color: #: \_eventCategory.colorCode#'></div>",

field: "colorCode",

title: " ",

width: 80

},  
  
And we will add another 2 more columns for displaying the Start and End Dates for the events.  
  
{

field: "startDate",

title: "Start Date",

template: "<span>#: kendo.toString(new Date(kendo.parseDate(startDate)), 'dd-MM-yyyy')# #: startTime#</span>",

width: 170

},

{

field: "endDate",

title: "End Date",

template: "<span>#: kendo.toString(new Date(kendo.parseDate(endDate)), 'dd-MM-yyyy')# #: endTime#</span>",

width: 170

},

* 1. When working with templates, there are 2 ways to bind your data:
     1. #: fieldName#
     2. #= fieldName#
  2. When working with Kendo UI, we can make use of Kendo UI’s framework to manipulate how we want our data to be presented.
  3. The kendo namespace contains many different capabilities including the toString() method.
  4. More library methods can be found in our documentation here: <http://docs.telerik.com/kendo-ui/api/javascript/kendo>

1. Next, let’s explore logical expressions within Kendo UI’s templating capability:

{

field: "isAllDay",

title: "isAllDay?",

template: "<span># if (isAllDay) { # YES # } else { # NO # } #</span>",

headerTemplate: "<div style='text-align: center;'>isAllDay?</div>",

width: 100

},

1. Once we have done all that, next add a custom column that will present us with 2 buttons so we can perform Edit / Delete operations:  
     
   {

template: "<div style='text-align: center;'><a class='form-btn grid-buttons k-button' href='/events/edit/?id=#: \_id#'>Edit</a>&nbsp;" +

"<a class='form-btn grid-buttons k-button' href='/api/events/delete\_event/?id=#: \_id#'>Delete</a></div>",

field: "\_id",

title: " ",

width: 188

},

1. Displaying a single level Grid view is easy. But what happens when we have more information that we want to present but do not have the width to display them? We make use of the details view feature. We start by declaring the template using a custom <script /> tag:  
     
   <script type="text/x-kendo-template" id="event-details-template">

<div style="width: 100%; margin: 5px;">

<p style="font-size: 12px; width: 100%; padding: 5px; line-height: 1.3em;">

<label style="width: 120px; text-align: right; margin-right: 5px;">Venue:</label>#: \_eventVenue.venueName#<br />

<label style="width: 120px; text-align: right; margin-right: 5px;">Event Type:</label>#: \_eventType.typeName#<br />

<label style="width: 120px; text-align: right; margin-right: 5px;">Event Category:</label>#: \_eventCategory.categoryName#<br />

</p>

</div>

</script>

1. In our final step, we will now specify the details template property within our Grid control, you can add this line of code just above where you declared your columns:  
     
   detailTemplate: kendo.template($("#event-details-template").html()),

Here is a final look at what everything looks like when it’s completed:

<div id="grid-events" style="margin-bottom: 15px;"></div>

<script>

$('#grid-events').kendoGrid({

dataSource: {

//type: "json",

transport: {

read: "{{baseUri}}/api/events/get\_all\_events"

},

pageSize: 20

},

height: 550,

groupable: true,

sortable: true,

pageable: {

refresh: true,

pageSizes: true,

buttonCount: 5

},

detailTemplate: kendo.template($("#event-details-template").html()),

columns: [

{

template: "<div style='text-align: center;'><a class='form-btn grid-buttons k-button' href='/events/edit/?id=#: \_id#'>Edit</a>&nbsp;" +

"<a class='form-btn grid-buttons k-button' href='/api/events/delete\_event/?id=#: \_id#'>Delete</a></div>",

field: "\_id",

title: " ",

width: 188

},{

template: "<div style='text-align: center; border-radius: 5px; margin: 8px; height: 50px; width: 50px; background-color: #: \_eventCategory.colorCode#'></div>",

field: "colorCode",

title: " ",

width: 80

},

{

field: "eventName",

title: "Event Name"

},

{

field: "isAllDay",

title: "isAllDay?",

template: "<span># if (isAllDay) { # YES # } else { # NO # } #</span>",

headerTemplate: "<div style='text-align: center;'>isAllDay?</div>",

width: 100

},

{

field: "startDate",

title: "Start Date",

template: "<span>#: kendo.toString(new Date(kendo.parseDate(startDate)), 'dd-MM-yyyy')# #: startTime#</span>",

width: 170

},

{

field: "endDate",

title: "End Date",

template: "<span>#: kendo.toString(new Date(kendo.parseDate(endDate)), 'dd-MM-yyyy')# #: endTime#</span>",

width: 170

},

{

field: "\_eventStatus.statusName",

title: "Status",

width: 120

}

]

});

</script>

<script type="text/x-kendo-template" id="event-details-template">

<div style="width: 100%; margin: 5px;">

<p style="font-size: 12px; width: 100%; padding: 5px; line-height: 1.3em;">

<label style="width: 120px; text-align: right; margin-right: 5px;">Venue:</label>#: \_eventVenue.venueName#<br />

<label style="width: 120px; text-align: right; margin-right: 5px;">Event Type:</label>#: \_eventType.typeName#<br />

<label style="width: 120px; text-align: right; margin-right: 5px;">Event Category:</label>#: \_eventCategory.categoryName#<br />

</p>

</div>

</script>

|  |
| --- |
| Exercise Two: Working with Schedulers |

The Scheduler control within Kendo UI has many different capabilities, features as well as customization options. But before you dive into the APIs of the control, you will follow this exercise to creating your own Scheduler control. In this exercise, we will take you through step by step as we create a new Scheduler control, and then adding a drop down list control to help with the filtering of the Events in the Scheduler.

1. First, we will start by opening the dashboard.hbs file within the event’s view folder. Go to the code section of the Tab Control which contains the Scheduler comment, and we will add the following line of code:

<div id="scheduler-wrapper"></div>

1. Next, we will go to the JavaScript section of our file and we will start by adding a few lines of codings which will be necessary for the Scheduler to work:

var *categories* = [];  
  
var *defaultCategory* = '';  
  
var *today* = new Date();  
var *startTime* = new Date(*today*.getFullYear(), *today*.getMonth(), *today*.getDate(), 7,0,0);  
var *endTime* = new Date(*today*.getFullYear(), *today*.getMonth(), *today*.getDate(), 23,0,0);  
  
$('#scheduler-wrapper').html('<div id="scheduler"></div>');

* 1. It is important to understand a few things here, we have added a categories[] array because we will be using this later on for specified the categories that we want to add to the Scheduler as a resource.
  2. startTime – this value that we get will specify the standard start time of each day that you want to display in the Scheduler control.
  3. endTime – this is the value of the end time of each day that you want to display in the Scheduler control.
  4. We have added a link of jQuery code to add our scheduler control’s html code within the wrapper. This is necessary because when we re-instantiate the control as each Facility Category is being selected, we will need to remove the HTML tags and re-add them again.

1. Next, we will add a document.ready(callback) function.

$(document).ready(function() {  
 // We will add some initialization coding here.  
});

1. Next, we will add our DropDownList initializer.

$('#eventCategories').kendoDropDownList({  
 index: 0,  
 dataTextField: "categoryName",  
 dataValueField: "\_id",  
 dataSource: {  
 //type: "json", // specifies data protocol  
 transport: {  
 read: "{{baseUri}}/api/eventcategories/get\_list\_selection"  
 }  
 },  
 value: *defaultCategory*,  
 select: function (e) {  
  
 },  
 height: 250  
});

1. Once our eventCategories DropDownList has been created, next, we need to set up our scheduler control. We will start by defining a new method called loadScheduler():

function *loadScheduler*() {  
   
}

1. Next, we will add an AJAX GET method, which will be used to retrieving all the categories that are available. In this workshop sample, we will be listing all the events available within a single Scheduler. But it will be an exercise for you to figure out how you can listing your events separately in different scheduler views.

$.get('{{baseUri}}/api/eventcategories/get\_list\_selection', function(data) {

});

1. Let’s focus on building the resources dataSource object.

data.forEach(function (category) {  
 *categories*.push({  
 text: category.categoryName,  
 value: category.\_id,  
 color: category.colorCode  
 });  
});

When working with resources, it is helpful to understand that there are a few attributes that will be useful and needed.

* 1. text – this value is used in the displaying of the category’s name in the Scheduler.
  2. value – this will be the identifier for the resource.
  3. color – colour that is being used to differentiate the different categories.

1. When working with a scheduler control, bear in mind that there are many different attributes and configurations you can set and customize, and a detailed discussion of that is beyond the scope of this workshop exercise. But looking at the coding sample below, here’s a simply explanation of some of the key properties you will most likely be working with:  
   1. date – this property refers to the current date ie. today
   2. timezone – there is a list of accepted timezone values that can be found within our online documentation. You can make the timezone dynamic depending on the user’s current location.
   3. startTime – as explained in an earlier step
   4. endTime – as explained in an earlier step
   5. height – this is the height of the scheduler control when rendered.
   6. views – this property will allow you to specify an array of configurations, starting with the types of display modes you want the Scheduler control to have. A few of those options include:
      1. day – current day
      2. week – span of an entire week
      3. month – view of a span of month(s)
      4. timeline – think of this like a continuous timeline view (horizontal mode)
   7. editable – by setting the editable property, it will enable / disable the inline editing capability.
   8. eventTemplate – this is the template that we will use to display our event descriptions.
   9. currentTimeMarker – User Experience is a very large part of what the team behind Kendo UI is striving at. And when it comes to the Scheduler control, we have a real time time marker that moves as the time of the day progresses.
   10. dataSource – please note that this dataSource object is different from the standard dataSource object that comes with the other data driven controls in Kendo UI.
   11. resources – the resources collection property is a very useful property to have. It allows you to specify collections of data that can be used for the rendering of the events within the Scheduler, but it can also store collections that you want to use in a custom Event Editor window. Discussion into this is beyond the scope, but you can explore and learn more about it from our documentations.

$('#scheduler').kendoScheduler({  
 date: new Date(),  
 timezone: "GMT0",  
 startTime: *startTime*,  
 endTime: *endTime*,  
 height: 1000,  
 views: [  
 { type: "day", showWorkHours: false },  
 { type: "week", selected: true, showWorkHours: false },  
 "month",  
 "timeline"  
 ],  
 editable: false,  
 eventTemplate: $('#scheduler-event-template').html(),  
 currentTimeMarker: {  
 updateInterval: 100  
 },  
 dataSource: {  
 batch: false, // Enable batch updates  
 transport: {  
 read: {  
 url: "{{baseUri}}/api/events/get\_all\_events"  
 //dataType: "json"  
 }  
 },  
 schema: {  
   
 }  
 },  
 resources: [  
 {  
 field: "categoryId",  
 name: "Categories",  
 dataSource: *categories*,  
 title: "Category"  
 }  
 ],  
 edit: function(e) {  
 e.preventDefault();  
 }  
});

1. Next, we will start to add our schema and model definition for the schedulerDataSource:

schema: {  
 model: {  
 id: "eventId", // The "id" of the event is the "taskId" field  
 fields: {  
 // Describe the scheduler event fields and map them to the fields returned by the remote service  
 eventId : { from: "\_id", type: "string" },  
 categoryId : { from: "\_eventCategory.\_id" },  
 title : { from: "eventName", defaultValue: "No title", validation: { required: true } },  
 description : { from: "eventDesc" },  
 start : { type: "date", from: "startDate" },  
 end : { type: "date", from: "endDate" },  
 startTime : { type: "string", from: "startTime" },  
 endTime : { type: "date", from: "endDate" },  
 timeSlotEnd : { type: "string", from: "endTime" },  
 isAllDay : { type: "boolean", from: "isAllDay" },  
 eventType : { type: "string", from: "\_eventType.typeName" },  
 typeId : { from: "\_eventType.\_id" },  
 statusId : { from: "\_eventStatus.\_id" },  
 venueId : { from: "\_eventVenue.\_id" },  
 colorCode : { from: "\_eventCategory.colorCode", type: "string" }  
 }  
 }  
}

A few key points that you will need to take note here:

* 1. The property names on the left-hand side are non-negotiable. Some of the attributes are out of the box with Kendo UI, but you can also add your own attributes. The recommendation is to keep the names in a single word and avoid using ‘\_’ underscore characters.
  2. Notice that each field has a “from” property tied to it. This ‘from’ property states where it’s getting its value from. We recognize that some times you might be interfacing an application from a database not created by you and you might not have the say in what sort of values are available. Just make sure that the types are correct.

1. Remember our Event Details template? It is the important part of what makes your Scheduler’s events look great.

<script type="text/x-kendo-template" id="scheduler-event-template">  
 <div>  
 <p style="font-size: 14px; font-weight: bold; width: 100%; margin: 5px; text-wrap: normal;">**#: title #**</p>  
 <p style="**font-size: 12px; width: 100%; padding: 5px;**">**#: eventType #**</p>  
 <button onclick="*editEvent*('#: eventId#')" class="control-buttons edit-event-btn">**Edit**</button>  
 </div>  
</script>

1. In this last and final step, we will add the code for when the user selects each category from the DropDownList. You will need to add this into the select event handler for the DropDownList.

*categories* = [];  
  
var scheduler = $('#scheduler').data('kendoScheduler');  
  
scheduler.destroy();  
  
$('#scheduler-wrapper').html('<div id="scheduler"></div>');  
  
*loadScheduler*();

Challenge Yourself:

With the code that has already been implemented, how do you filter the Scheduler to only displaying Events relevant to what has been selected from the DropDownList?

|  |
| --- |
| Exercise Three: Chart Controls |

When it comes to chart controls, Kendo UI has the most number of chart controls to choose from, and most of it, it’s the easiest to customize and configure. In this exercise, we are only going to focus on a single type of chart, which is the Donut Chart. A Donut Chart is very useful when it comes to display percentages in a reporting manner. And in our sample application, we most definitely want to be able to display our charts to look at what sort of Events have been created based on the different categories and what are their percentages.

1. We start by opening the home/dashboard.hbs file within our views folder.
2. You can start by adding the following lines of HTML code to the <div /> tag that is already present in the code view:

<div style="margin: 5% **auto**;">  
 <div id="eventByTypeChart" style="width: 50%; float: **left**;"></div>  
 <div id="eventByCategoryChart" style="width: 50%; float: **left**;"></div>  
</div>

1. Next, we will start to define our chart controls from within the javascript coding. We will start by looking at one of the charts that we are trying to create:

$("#eventByTypeChart").kendoChart({  
 title: {  
 text: "No. of Events (by Event Type)"  
 },  
 legend: {  
 position: "bottom"  
 },  
 dataSource: {  
 transport: {  
 read: {  
 url: "http://localhost:2080/api/events/get\_events\_report/?filter=type"  
 }  
 }  
 },  
 series: [{…}, {…}, {…}],  
 seriesColors: ["#42a7ff", "#666666"],  
 tooltip: {  
 visible: true,  
 template: "${ category } - ${ value }%"  
 }  
});

Let’s go through what some of the properties of the chart does:

* 1. title – the main title for your chart
  2. legend – you have a choice of placing your legend in 4 different places:
     1. bottom
     2. top
     3. right
     4. left
  3. dataSource – this dataSource object use by all the charts are the same as your data driven controls.
  4. series – covers all the data series that you might have. But in this example, we will only keep it to one.
  5. seriesColors – multiple colour options for your series of data.
  6. tooltip – shows a helpful text information when you mouse over the chart.

1. In this Exercise, you will now start to create your own Donut chart like the one that you have seen above. The URL that you will be using for this part of the exercise is <http://localhost:2080/api/events/get_events_report/?filter=category>. This URL will retrieve all the reporting data filter by Event Category.

|  |
| --- |
| Exercise Four: Editor Control |

Our Editor control is so powerful, yet at the same time very easy to implement. There are many amazing options available to you when working with our Kendo UI Editor control, features like copying-and-pasting from word documents, or even exporting the content into a PDF document. In this Exercise, we are going to look at a simple implementation of the Editor control, and then you will have an exercise of implementing the Editor control with the PDF Exporting capabilitity.

1. First thing that we do is to open the newsarticle/create.hbs file within our views folder. Once that is done, add the following lines of code in the file if it’s not there yet.

<div class="form-group">  
 <label for="articleContent" class="label-full">**Article Content**</label>  
 <textarea id="articleContent" name="articleContent" class="k-textbox" required aria-required style="display: **inline-block**; width: 100%; height: 550**px**;" rows="15"></textarea>  
</div>

The line of code that you should be focusing your attending on is the <textarea /> tag.

1. Next, within our JavaScript code segment, we will add the following line of code:

$("#articleContent").kendoEditor({  
   
});

Before you get excited about it, this isn’t enough for the control to function, there are a few other properties that you will need to specify for the Editor to render beautifully and to work.

tools: [  
 "bold",  
 "italic",  
 "underline",  
 "strikethrough",  
 "justifyLeft",  
 "justifyCenter",  
 "justifyRight",  
 "justifyFull",  
 "createLink",  
 "unlink",  
 "insertImage",  
 "createTable",  
 "addColumnLeft",  
 "addColumnRight",  
 "addRowAbove",  
 "addRowBelow",  
 "deleteRow",  
 "deleteColumn",  
 "foreColor",  
 "backColor"  
]

The tools property is where we specify an array of string values to state what are some of the functionalities we want to provide to the user. For a full list of all the capabilities within the Editor control, you can follow our online documentation for that information. Next, we will need to specify if we want to allow the user to resize the Editor’s editable view. We do that using the resizable attribute:

resizable: {  
 content: true,  
 toolbar: true  
},

By stating that the content & toolbars are both resizable, users can control how their Editor’s view will be rendered.

Challenge Yourself:

1. Following our documentation online, try to implement the PDF export capability. (Tip: You can use the PDF web services from Telerik for the proxy url - //demos.telerik.com/kendo-ui/service/export)
2. Also, try implementing the Image Browser functionality. (Tip: The Web APIs are already available within the relevant NewsArticle.js controller. Use that for your implementation.

|  |
| --- |
| Exercise Five: Window control |

For our last exercise, we will look at implementing the Window UI control. The Window UI control is a very simple idea, but its implementations are dynamic. There is just so many things that you can do with the Window control, be it to display information, or to create custom forms. In this example, we will use the window widget to create a View Article window. And as part of the exercise, you will also have a chance to make certain customizations and implementations on your own.

1. To work with the Window control, we will start by add the relevant HTML code.

<div id="readArticleWindow">  
 <!-- Add other html codes here later -->  
</div>

1. Next, we will need to instantiate the window control. Depending on your project requirements, sometimes you might want the window to be open by default when the page is loaded, or closed, its entirely up to you. You can specify that when instantiating the control itself.

$("#readArticleWindow").kendoWindow({  
 width: "600px",  
 title: "Article Details",  
 visible: false,  
 actions: [  
 "Pin",  
 "Minimize",  
 "Maximize",  
 "Close"  
 ]  
});

There are a few simple basic options when working with the Window control:

* 1. width – the actual physical width of the window.
  2. title – the title that appears at the top of the Window.
  3. visible – either true / false
  4. actions – specifies a list of options that you want the window to have:
     1. Pin
     2. Minimize
     3. Maximise
     4. Close

1. Now, when the user view the list view of News Articles, they should be able to click on the button “Read Article” and it will execute a JavaScript function to show our window and also populate the window with the article’s contents. But before we go into that code, we need to add in the HTML contents for our window view first. Add the following codes within the <div> tag for the Window control.

<input type="hidden" id="hdnArticleId" name="hdnArticleId" value="" />  
<div id="articleTitle" style="width: 100%; float: **left**; display: **inline-block**; font-weight: **bold**; font-size: 15**px**;"></div>  
<div id="articleContent" style="margin-top: 10**px**; width: 100%; float: **left**; display: **inline-block**;"></div>  
<div style="margin-top: 10**px**; width: 100%; float: **left**; display: **inline-block**; text-align: **right**;">  
 <a href="javascript:void(0)" onclick="*editArticleClick*()" class="control-buttons form-btn k-primary">**Edit Article**</a>  
 <a href="javascript:void(0)" onclick="*closeWindow*()" class="control-buttons form-btn k-primary">**Close Window**</a>  
</div>

1. Next, we need to implement the method for handling the showing of the window:

function *viewArticle*(id) {  
 $.get('{{baseUri}}/api/newsarticles/get\_article\_details/?id=' + id, function(data) {  
  
 $('#hdnArticleId').val(id);  
  
 var readArticleWin = $("#readArticleWindow").data('kendoWindow');  
 readArticleWin.center().open();  
  
 $('#articleTitle').html(data.articleTitle);  
 $('#articleContent').html(data.articleContent);  
 });  
}

In the above method, you notice how we have retrieved the relevant article details from the web service. And then we need to get the Window control before we can even start to manipulate it.

var readArticleWin = $("#readArticleWindow").data('kendoWindow');  
readArticleWin.center().open();

The above lines of codes will cause the Window to appear in the center of the browser space.

1. Although the Window control itself already has it own way of handling the close event, sometimes you might want to have your own buttons within the Window, and in our case, we will handle the closing of the Window with a button click.

function *closeWindow*() {  
 var readArticleWin = $("#readArticleWindow").data('kendoWindow');  
 readArticleWin.close();  
}

Challenge Yourself:

Now that we have a working Window view, try to implement the editing capability within the Window control and making it look like an inline editing. Most users will not enjoy having to be redirected to a separate page to do the editing. Using the skills you have learnt in this workshop, set up the Window so that you can both edit and save the changes of the article.