Contents

[require.js 2](#_Toc455993911)

[Purpose: 2](#_Toc455993912)

[Configuration: 2](#_Toc455993913)

[Step 1: Define Shim (app.js) 2](#_Toc455993914)

[Step 2: Initialising (app.js) 2](#_Toc455993915)

[Step 3: Initialising Services (mainService.js) 2](#_Toc455993916)

[Step 4: Wiring to the View (\_Layout.cshtml) 2](#_Toc455993917)

[interactive.js 3](#_Toc455993918)

[Purpose: 3](#_Toc455993919)

[Wiring of dataTable: 3](#_Toc455993920)

[Rewiring of dataTable: 3](#_Toc455993921)

[Wiring of formValidator: 3](#_Toc455993922)

[Rewiring of formValidator: 3](#_Toc455993923)

[Wiring datePicker: 3](#_Toc455993924)

[Wiring select2: 4](#_Toc455993925)

[Wiring Bootstrap ToolTip: 4](#_Toc455993926)

[Wiring Rich Text Editor/tinyMCE: 4](#_Toc455993927)

[Wiring Bootstrap Modal (Popup): 4](#_Toc455993928)

[Wiring Validate Email: 5](#_Toc455993929)

[dataTables.js 7](#_Toc455993930)

[Purpose: 7](#_Toc455993931)

[Dependency Configuration (app.js): 7](#_Toc455993932)

[HTML Configuration: 7](#_Toc455993933)

[Settings: 7](#_Toc455993934)

[jquery-fullcalendar.js 9](#_Toc455993935)

[Purpose: 9](#_Toc455993936)

[Configuration and setting up 9](#_Toc455993937)

# [require.js](http://requirejs.org/)

## Purpose:

It is a JavaScript file and module loader that is optimized for in-browser use. Using a modular script loader like require.js will improve the speed and quality of the code.

## Configuration:

### Step 1: Define Shim ([app.js](http://10.9.230.200/tfs/DefaultCollection/Doe.Ls.ProjectTemplate/_versionControl?path=%24%2FDoe.Ls.ProjectTemplate%2FTrunk%2FDoe.Ls.SampleProject.Web%2FScripts%2Fapp.js&_a=contents))

* The scripts that are not require.js enabled defined here.
* Files should be referenced without the extension.
* Plugins should be added to the Shim by following the pattern of

require.config({

baseUrl: PATH\_TO\_SCRIPTS\_FOLDER e.g.: window.appUrl + 'Scripts/',

paths: {

jquery: PATH\_TO\_JQUERY\_FILE\_WITHOUT\_EXTENSION e.g.: Framework/jquery-1.12.0.min',

otherPlugins : PATH\_TO\_OTHER\_PLUGIN\_FILE\_WITHOUT\_EXTENSION

},

shim: {

jquery: {

deps: [modernizr]

},

bootstrap:{

deps: [jquery]

},

OTHER\_PLUGIN:{

deps: [DEPENDENCY]

}

}

});

### Step 2: Initialising ([app.js](http://10.9.230.200/tfs/DefaultCollection/Doe.Ls.ProjectTemplate/_versionControl?path=%24%2FDoe.Ls.ProjectTemplate%2FTrunk%2FDoe.Ls.SampleProject.Web%2FScripts%2Fapp.js&_a=contents))

Invoke the function for basic initialisation with the required plugins

require(['jquery', 'mainService', 'modernizr'], function ($, main) {

$(function () {

main.initialise();

});

});

### Step 3: Initialising Services ([mainService.js](http://10.9.230.200/tfs/DefaultCollection/Doe.Ls.ProjectTemplate/_versionControl?path=%24%2FDoe.Ls.ProjectTemplate%2FTrunk%2FDoe.Ls.SampleProject.Web%2FScripts%2FmainService.js&_a=contents))

Invoking the basic initialisation and wiring the custom JavaScript services depending upon the form name or the wrapper name

return {

initialise : function(){

basicInitialisation();

var wrapperId = getWrapperId();

var formId = getFormId();

if(wrapperId== ‘wrapper-A’ || formId== ‘form-A’){

require([‘AService’], function(service){

service.initialise();

});

}

}

}

### Step 4: Wiring to the View ([\_Layout.cshtml](http://10.9.230.200/tfs/DefaultCollection/Doe.Ls.ProjectTemplate/_versionControl?path=%24%2FDoe.Ls.ProjectTemplate%2FTrunk%2FDoe.Ls.SampleProject.Web%2FViews%2FShared%2F_Layout.cshtml&_a=contents))

Declare the require object that depends upon the app.js and finally add reference to require.js

<script type="text/javascript">

var require = {

baseUrl: "/scripts/",

waitSeconds: 15,

deps: ['@Url.Content("~/Scripts/app.js")']

};

</script>

<script src="@Url.Content("~/Scripts/Framework/require.js")"></script>

# [interactive.js](http://10.9.230.200/tfs/DefaultCollection/Doe.Ls.ProjectTemplate/_versionControl?path=%24%2FDoe.Ls.ProjectTemplate%2FTrunk%2FDoe.Ls.SampleProject.Web%2FScripts%2Finteractive.js&_a=contents)

## Purpose:

This custom JavaScript code is to wire-up all the events stating from setting up dataTable, modal popups and all the plugins

## Wiring of dataTable:

* Wiring of dataTable occurs at the initialisePlugins function.
* It wires up the tables that does not have class name ajaxDataTable or notDataTable
  + ajaxDataTable class is used to wire the dataTable with ajax
  + notDataTable class is used for a normal table that should not be represented as dataTable
* Any function that should be called as soon as the dataTable is loaded (such as hookPopupButton), should be called at the fnDrawCallback property of the dataTable
* For the dataTable page length dropdown list to have the select2 effect, select2 is initialised through hookDataTableSelect

## Rewiring of dataTable:

* applyNewDataTableSettings allows the developer to rewire the dataTable
* Parameters:
  + extendedSettings – The new settings that to be applied on the dataTable
  + $table – The table object where the dataTable to be applied
* The old dataTable has to be destroyed by using fnDestroy function from the initialised dataTable object

## Wiring of formValidator:

* Wiring of formValidator occurs at the initialisePlugins 🡪 hookGenericFormValidation function.
* Initial wiring of formValidator works with html5 input types (such as number, email, url) and properties (such as required)
* The validation messages can be set as html element attribute such as data-fv-notempty-message for required field and/or data-fv-integer-message for input type number.
* Initially the disabled, hidden and/or :not(:visible) fields are excluded from the formValidation but this can be overwritten by setting the exclude property when initialised.   
  e.g.: excluded: [':disabled', ':hidden', ':not(:visible)']

## Rewiring of formValidator:

* validateForm allows the developer to rewire the formValidator
* Parameters:
  + form – The form ID upon which the new validation will be applied
  + extendedSettings – The new settings that to be applied
* The old formValidator object to be destroyed by using form.data('formValidation').destroy()

## Wiring datePicker:

* Wiring of datePicker occurs at initialisePlugins 🡪 basicInitialisation 🡪 hookDatePicker function
* The wiring occurs for any input with type date and not readonly   
  e.g.: $('input[type=date]').not('[readonly]') and for any input with data-date attribute is date   
  e.g.: $('input[data-date=date]')

## Wiring select2:

* Wiring of select2 occurs at initialisePlugins 🡪 basicInitialisation 🡪 hookSelect2 function
* The wiring occurs for any input with class name select2picker with a container   
  e.g.: $('.select2picker', pContainer)
* Select2 can be reinitialised upon a dropdown list using hookSelect2 function

## Wiring Bootstrap ToolTip:

* Wiring of ToolTip occurs at initialisePlugins 🡪 basicInitialisation 🡪 applyToolTip function
* The wiring occurs for any html element with data-tag and/or data-content attribute

e.g.: $('\*[data-tag],\*[data-content]')

* Wired ToolTip will appear for hover only
* Developer can configure the ToolTip by the following element attributes:
  + data-content – Text to display
  + data-tag – Tag ID that will be fetched from database through ajax
  + data-delay – Delay value in millisecond upon with the ToolTip will disappear (default: 100)
  + data-placement – Placement of the ToolTip (default: left)
* For ajax ToolTip developer should update the cnt.MessageUrl with the MVC action that will send the ToolTip as json object

## Wiring Rich Text Editor/tinyMCE:

* Wiring of tinyMCE occurs at initialisePlugins 🡪 basicInitialisation 🡪 hookTinymce function
* The wiring occurs for any html element with any of the following class name
  + tinymce – Default settings (height: 150px)
  + tinymce-rich – Rich settings (height: 800px)
  + tinymce-rich-small – Rich but small settings (height: 100px)
  + tinymce-simple – Simple settings (height: 30px)
* For legacy applications with html element with data-role-rich="rich-text" attribute will also wire up the tinyMCE
* For formValidation to work with tinyMCE the following steps to be followed
  + Step 1: onKeyUp event for the editor the text should be saved in the targeted html element

e.g.: editor.on('keyup', function (e) {

editor.save();

…

* + Step 2: Initiate formValidation.revalidateField for the targeted html element

e.g.: var fieldName = $(targetElement).attr('name');

$(targetElement).closest('form').formValidation(cnt.fv\_RevalidateField, fieldName);

## Wiring Bootstrap Modal (Popup):

* Wiring of Popup occurs at hookPopupButton function
* Rewiring of Popup (e.g.: dataTable buttons) occurs using the same hookPopupButton function
* hookPopupButton function parameters
  + validatorSettings – Any explicit formValidation rules for the Popup (default: {})
  + modalLoadCallBack – Function to be executed as soon as Popup is loaded
  + useAjaxSubmitHandler – Boolean value to control ajax submit (default: true)
  + successCallback – Function to be executed for successful submit
  + errorCallBack – Function to be executed for submit with error
  + elementSelector – Html Class or ID to hook the Popup upon (default: .pop-up)
  + autoHide – Boolean value to automatically hide the Popup upon successful submission (default: true)
* Setting up steps
  + Step 1: Add the modal div to the bottom of the Html body
    - The structure of the modal div is

<div class="modal fade" tabindex="-1" role="dialog" id="tempModal">

<div class="modal-dialog">

<div class="modal-content"></div> <!-- /.modal-content -->

</div> <!-- /.modal-dialog -->

</div>

* + - As the modal div comes with an ID, make sure that previously added modal div is removed before adding the new modal div
  + Step 2: Invoke loaded.bs.modal from Bootstrap and inject the html elements of the Popup page into the Modal

$('#tempModal').on('loaded.bs.modal', function (e) {

var content = $(e.target.innerHTML);

var remoteFormId = '#' + $('form', content).attr('id');

* + Step 3: Execute the modalLoadCallBack if not null otherwise use basicInitialisation so that the elements have the plugin functionalities

if (modalLoadCallBack != null) {

modalLoadCallBack($(remoteFormId));

} else {

var popupContainer = $('#tempModal');

basicInitialisation(popupContainer);

}

* + Step 4: Wire up the formValidator settings

var settings = $.extend(

{},

validatorSettings,

{

framework: 'bootstrap'

});

var formValidation = validateForm($(remoteFormId), settings);

* + Step 5: Wire up the submit handler upon validated form using success.form.fv from formValidator

formValidation.on('success.form.fv', function (evt) {

evt.preventDefault();

var form = $(evt.target);

var validator = $(evt.target).data('formValidation');

if (useAjaxSubmitHandler == false) {

validator.defaultSubmit();

} else {

$.ajax({

});

* + Step 6: Add successCallback & errorCallback in the ajax callback functions
  + Step 7: Remove the tempModal form the DOM along with the backdrop (some browser needs that) using hidden.bs.modal

$('#tempModal').on('hidden.bs.modal', function () {

$('#tempModal').remove();

$('.modal-backdrop').remove();

});

## Wiring Validate Email:

* Wiring of Validate Email can be done using any of the below functions:
  + hookValidateEmail
    - Parameters:
      1. verifyButton – Verify button object
      2. fieldList – Array of fieldnames to be updated
      3. successCallback – Success Call back function
      4. errorCallback – Error Call back function
  + hookValidateEmailV2
    - Parameters:
      1. verifyButton – Verify button object
      2. successCallback – Success Call back function
      3. errorCallback – Error Call back function
* hookValidateEmail can be called as per the below example

interactive.hookValidateEmail(

$('.verifyEmail'), // Verify button

['UserName', 'DisplayName', 'Email'], // Fields to Update

function (userInfo) { // Success callback

},

function (jqXHR, textStatus, errorThrown) { // Error callback

});

* On click of the verifyButton, the function will use a ajax call to retrieve the user information from the UserInfoWebservice and populate the fields and if there is an error it will clear the fields

$(verifyButton).on('click', function () {

var email = $('input[name\*="Email"]').val();

var form = $(verifyButton).closest('form');

api.ajaxGet(

cnt.GetUserInfoFromAdByEmailUrl,

'email=' + email,

function (userInfo) {

$('input[name\*="Email"]').val(email.toLowerCase());

$.each(fieldList, function (index, fieldName) {

$('input[name\*="' + fieldName + '"]').val(userInfo[fieldName]);

});

if (helper.hasValue(successCallback)) {

successCallback(userInfo);

}

},

function () {

$('input[name\*="Email"]').val('\_' + email.toLowerCase());

$.each(fieldList, function (index, fieldName) {

$('input[name="' + fieldName + '"]').val('');

});

if (helper.hasValue(errorCallback)) {

errorCallback();

}

});

});

}

# dataTables.js

## Purpose:

It is a highly flexible, bootstrap enabled and responsive jQuery plugin that adds advanced interaction control to any html table.

## Dependency Configuration (app.js):

* Paths (including extensions):

'datatables': 'Plugins/bootstrapDatatable/datatables.min',

'datatables.net': 'Plugins/bootstrapDatatable/DataTables-1.10.11/js/jquery.dataTables.min',

'datatables.net-bs': 'Plugins/bootstrapDatatable/DataTables-1.10.11/js/dataTables.bootstrap.min', 'datatables.net-responsive': 'Plugins/bootstrapDatatable/Responsive-2.0.2/js/dataTables.responsive.min',

'datatables.net-responsive-bs': 'Plugins/bootstrapDatatable/Responsive-2.0.2/js/responsive.bootstrap.min'

* Shim:

'datatables': {

deps: [ 'jquery',

'bootstrap',

'datatables.net',

'datatables.net-bs',

'datatables.net-responsive',

'datatables.net-responsive-bs']

},

## HTML Configuration:

* Add the below css files
  + ~/Scripts/plugins/bootstrapDatatable/DataTables-1.10.11/css/jquery.dataTables.min.css
  + ~/Scripts/plugins/bootstrapDatatable/Responsive-2.0.2/css/responsive.dataTables.min.css
  + ~/Scripts/plugins/bootstrapDatatable/Responsive-2.0.2/css/responsive.bootstrap.min.css
* Well formatted HTML table is required (that includes thead, tbody and optional tfoot)

e.g.: <table id="table\_id" class="table">

<thead>

<tr>

<th>Column 1</th>

<th>Column 2</th>

</tr>

</thead>

<tbody>

<tr>

<td>Row 1 Data 1</td>

<td>Row 1 Data 2</td>

</tr>

<tr>

<td>Row 2 Data 1</td>

<td>Row 2 Data 2</td>

</tr>

</tbody>

</table>

* For ajax sourced data, the tbody should be emptied

## Settings:

* iDisplayLength: Number of rows to be displayed initially e.g.: 25
* aLengthMenu: This parameter allows to specify the entries in the length drop down menu. It can be either a 1D array of options which will be used for both the displayed option and the value, or a 2D array which will use the array in the first position as the value, and the array in the second position as the displayed options (useful for language strings such as 'All'). [Legacy]  
  e.g.: [[5, 10, 25, 50, -1], [5, 10, 25, 50, "All"]]
* pagingType: The pagination option that will display a pagination control below the table   
  e.g.: full\_numbers   
  Full list of options are:
  + simple - 'Previous' and 'Next' buttons only
  + simple\_numbers - 'Previous' and 'Next' buttons, plus page numbers (Default)
  + full - 'First', 'Previous', 'Next' and 'Last' buttons
  + full\_numbers - 'First', 'Previous', 'Next' and 'Last' buttons, plus page numbers
* aoColumnDefs: An array that allows to target a specific column, multiple columns, or all columns, using the aTargets property of each object in the array. This allows great flexibility when creating tables, as the aoColumnDefs arrays can be of any length, targeting the specific columns. The aTargets property is an array to target one of many columns and each element in it can be:
  + a string - class name will be matched on the TH for the column
  + 0 or a positive integer - column index counting from the left
  + a negative integer - column index counting from the right
  + the string "\_all" - all columns (i.e. assign a default)

e.g.: [{ 'bSortable': false, 'aTargets': [-1] }]

* aoColumns: If specified, then the length of this array must be equal to the number of columns in the original HTML table. Use 'null' where you wish to use only the default values and automatically detected options. Both aoColumnDefs parameter and aoColumns can be used together, although aoColumnDefs is preferred due to its flexibility. If both are used, aoColumns definitions will take the highest priority. Likewise, if the same column is targeted multiple times in aoColumnDefs, the first elements in the array will take the highest priority, and the last the lowest.  
  Full list of options are:
  + aDataSort – Allows a column's sorting to take multiple columns into account when doing a sort. For example first name / last name columns make sense to do a multi-column sort over the two columns.

Type: array

Default: null  
e.g.: "aoColumnDefs": [  
 { "aDataSort": [ 0, 1 ], "aTargets": [ 0 ] },  
 { "aDataSort": [ 1, 0 ], "aTargets": [ 1 ] },  
 { "aDataSort": [ 2, 3, 4 ], "aTargets": [ 2 ] }  
 ]

* + asSorting – Allows to override the default sorting direction and alter the behaviour of the sort handler where needed

Type: array

Default: [‘asc’, ‘desc’]

e.g.: "aoColumnDefs": [  
 { "asSorting": ["asc" ], "aTargets": [ 1 ] },  
 { "asSorting ": ["desc" ], "aTargets": [ 3 ] },  
 ]

* + bSearchable – Allows user to enable or disable filtering on the data in the column

Type: boolean

Default: true

e.g.: "aoColumnDefs": [

{ "bSearchable": false, "aTargets": [ 0 ] }

]

* + bSortable – Allows user to enable or disable sorting on the data in the column

Type: boolean

Default: true

e.g.: "aoColumnDefs": [

{ "bSortable": false, "aTargets": [ 0 ] }

]

* + bVisible – Allows user to enable or disable display on the data in the column

Type: boolean

Default: true

e.g.: "aoColumnDefs": [

{ "bVisible": false, "aTargets": [ 0 ] }

]

* + mData – Allows user to read data from a JSON data source property, including deeply nested objects/propertiesenable or disable display on the data in the column

Type: string

Default: null

e.g.: "aoColumnDefs": [

{ "mData": "engine" },

{ "mData": "browser" }

{ "mData": "platform.detail" }

{ "mData": "platform.version.major" }

]

* + mRender – Allows user to customise the display data upon the mData value or any other value within the row.

Type: string

Default: null

e.g.: "aoColumnDefs": [

{ "mData": "engine" },

{ "mData": "browser" }

{

"mData": "platform",

"mRender": "[, ].name"

},

{

"mData": "downloadLink",

"mRender": function(data, type, full){

return "<a href=" + data + ">" + full[2] + "</a>";

}

}

]

* + sClass – Allows user to add a class name to each cell of the column.

Type: string

Default: empty string

e.g.: "aoColumnDefs": [

{ "sClass": "myClass", "aTargets": [ 0 ] }

]

Or

"aoColumns": [

{ "sClass": "myClass" },

null,

null,

null

]

* + sName – Only used for server-side processing as this maps the client side columns to the backend columns. Also allows the reorder of data from the server.

Type: string

Default: empty string

e.g.: "aoColumnDefs": [

{ "sName": "engine", "aTargets": [ 0 ] },

{ "sName": "browser", "aTargets": [ 1 ] },

{ "sName": "platform", "aTargets": [ 2 ] }

]

Or

"aoColumns": [

{ "sName": "engine" },

{ "sName": "browser" },

{ "sName": "platform" }

]

* aaSorting: If sorting is enabled, then DataTables will perform a first pass sort on initialisation. Column(s) can be define on which the sort is performed upon, and the sorting direction, with this variable. The aaSorting array should contain an array for each column to be sorted initially containing the column's index and a direction string ('asc' or 'desc').

e.g.: "aaSorting": [[2,'asc'], [3,'desc']]

* fnDrawCallback: This function is called on every 'draw' event, and allows user to dynamically modify any aspect about the created DOM. Generally, it is used to attach any event handler to the table that requires re-attaching as a result of Ajax based filtering and/or pagination.

e.g.: "fnDrawCallback": function( oSettings ) {

alert( 'DataTables has redrawn the table' );

}

* initComplete: This function fires as soon as the datatable is fully initialised

e.g.: "initComplete": function( settings, json ) {

$('div.loading').remove();

}

* fnServerParams: Function to pass additional data to server-side when making an Ajax request

e.g.: "fnServerParams": function ( aoData ) {

aoData.push( { "name": "more\_data", "value": "my\_value" } );

}

zxcz

# jquery-fullcalendar.js

## Purpose:

This plugin is used for managing calendar events in the application. You can have JSON feed as the event source.

## Configuration and setting up

Step1: Include style sheets and script files

* Include fullcalendar.css and fullcalendar.print.css in <head> of the page.
* Make sure added the following entries in app.js ‘moment: 'plugins/jQueryCalendar/moment.min'’ and ‘fullCalendar: 'plugins/jQueryCalendar/jquery-fullcalendar.min'’

Step2: Setting up

* Add div with id in html file

<div id='calendar'></div>

* Initialise

$('#calendar').fullCalendar({

header: {

left: 'prevYear,nextYear,prev,next today',

center: 'title',

right: 'month,agendaWeek,agendaDay'

},

defaultDate:startdate //if not mentioned current date will be default date,

selectable: true,

firstDay: 1,

slotEventOverlap: false,

timeFormat: 'HH:mm',

events: **JsonEvents** ,// Json event source

eventRender:function(event, element, view){

//customise how the events to be shown

}

});

Events can be loaded as below

var **JsonEvents** = {

url: cnt.GetFacilityEventsForSchoolApplicationCalendar + "?id=" + Id,

type: 'GET',

dataType: "json",

error: function (err) {

alert('there was an error while fetching events for calendar!' + err);

}

}