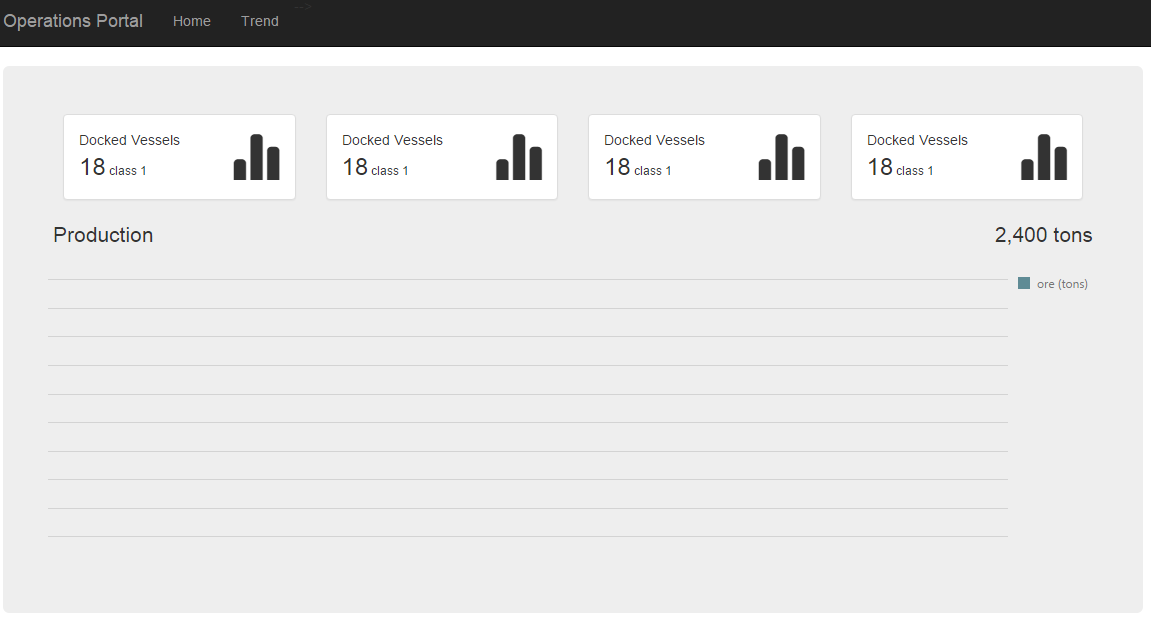
# Build Web-Based Real-Time Dashboards

## 1:20pm - 2:35pm

Building great dashboards that will bring joy to those that use them is something every UI developer strives to do. This lab will help you do just that. You will work through designing and building a simple dashboard using the PI Web API, AngularJS and other web technologies. We will talk about how to deploy real-life dashboard solutions including security considerations. This lab requires basic PI System knowledge, some experience working in Visual Studio, and basic C# programming skills. JavaScript and HTML5 knowledge is a plus, but not required.

1. Discuss some basics about design a dashboard (10 minutes)
   1. Review the project – show completed sample



* 1. Show some slides on basics for dashboard design

1. Start with a skeleton application that can be deployed to Azure as a web app, explain how it was created
   1. New Project – ASP.NET Web Application
   2. Project Name – dashboard
   3. Template – Empty Project
   4. Check Web API
   5. Create an index.html page in the base level of the project
   6. Add some “hello world” text to the body and run

<body>

<div>Hello UC2016!</div>

</body>

1. Bring a charting library and other Dependencies into AngularJS as module (5 minutes)
   1. Bootstrap 3.3.6 (will install jQuery)
   2. AngularJS.Core 1.5.0
   3. AngularJS.Route 1.5.0
   4. AngularJS.Sanitize 1.5.0
   5. jquery-globalize **0.1.1 This version is needed for the DevExtreme chart control**
   6. DevExtreme.Web 15.2.5 (may update jQuery)
   7. Add stylesheets and references to index.html
      1. Stylesheets – in the <head> section

<link href="Content/bootstrap.css" rel="stylesheet" />

<link href="Content/dx.common.css" rel="stylesheet" />

<link href="Content/dx.light.css" rel="stylesheet" />

* + 1. Scripts – at the end of the body section (below your hello world text)

<script src="Scripts/jquery-1.11.3.min.js"></script>

<script src="Scripts/jquery.globalize/globalize.js"></script>

<script src="Scripts/angular.min.js"></script>

<script src="Scripts/angular-route.min.js"></script>

<script src="Scripts/angular-sanitize.min.js"></script>

<script src="Scripts/bootstrap.js"></script>

<script src="Scripts/dx.webappjs.js"></script>

<script src="Scripts/dx.chartjs.js"></script>

* 1. Run application, using F12 in the browser, insure there are no errors

1. Create a dashboard layout with bootstrap styling (10 minutes)
   1. Add html for base framework to index.html, replace “hello world” text

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<a class="navbar-brand" href="#/Summary">Operations Portal</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="#/Summary">Summary</a></li>

<li><a href="#/Production">Production</a></li>

</ul>

</div>

</div>

</div>

<div class="container">

<div class="jumbotron">

</div>

</div>

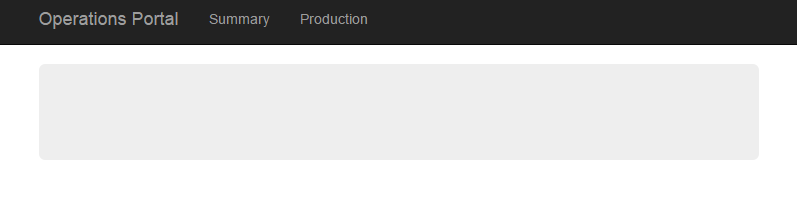
* 1. Create a folder called “App” in the base project
  2. Add a stylesheet in the “App” folder called app.css stylesheet and add the following styles:

body {

padding-top: 70px;

}

1. Add the stylesheet to index.html head section
2. Run and check for proper layout and insure there are no errors



1. Add in angular view and routing
   1. Explain angular views, controllers, and routes
   2. Create a folder in the “App” folder call “Views”
   3. Add a script file to the “Views” folder and call summaryController.js

'use strict';

angular.module('portalApp')

.controller('summaryController', ['$scope', function ($scope) {

$scope.summaryCount = '2018';

}]);

* 1. Add a html file to the “Views” folder and call summaryView.html

<!DOCTYPE html>

<html>

<head>

<title></title>

<meta charset="utf-8" />

</head>

<body>

<div>Summary View</div>

<div>Summary Count: **{{**summaryCount**}}**</div>

</body>

</html>

1. Copy the summary files with content for the Production view and controller, rename all occurrences of the word summary with production (watch spelling and case). Change the productionCount value to a new number.
2. Add a script file to the “App” folder called app.js and add the following:

'use strict';

angular.module('portalApp', ['ngRoute'])

.config(['$routeProvider', function ($routeProvider) {

$routeProvider.when('/Summary', {

controller: 'summaryController',

templateUrl: '/App/Views/summaryView.html',

}).when('/Production', {

controller: 'productionController',

templateUrl: '/App/Views/productionView.html',

}).otherwise({ redirectTo: '/Summary' });

}]);

1. Modify the index.html to connect the angular routing and views

<body ng-app="portalApp" ng-controller="summaryController">

1. Update nav-bar

<ul class="nav navbar-nav">

<li ng-class="{ active: isActive('/Summary') }">

<a href="#/Summary">Summary</a>

</li>

<li ng-class="{ active: isActive('/Production') }">

<a href="#/Production">Production</a>

</li>

</ul>

1. Add ng-view to main <div>

<div ng-view class="jumbotron">

1. Add script references

<!--application scripts-->

<script src="App/app.js"></script>

<script src="App/Views/productionController.js"></script>

<script src="App/Views/summaryController.js"></script>

1. Test to see that views are loaded and binding is working, use F12 to find errors
2. Create summary dashboard with test data
   1. Add ‘dx’ reference for chart control to the angular app module in app.js

Lk

angular.module('portalApp', ['ngRoute', 'dx'])

lk

* 1. Replace summaryView html with the dashboard layout – copy text from “summaryView-html.txt” file.
  2. Replace symmaryController with test data

'use strict';

angular.module('portalApp')

.controller('summaryController', ['$scope', function ($scope) {

$scope.summaryCount = '2018';

$scope.todayProduction = 8;

$scope.yesterdayProduction = 48;

$scope.monthlyProduction = 475;

$scope.dockedVessels = 16;

$scope.productionDataSource = [

{ 'Value': 12, 'Timestamp': 'March' },

{ 'Value': 14, 'Timestamp': 'April' },

{ 'Value': 10, 'Timestamp': 'May' }

];

$scope.production = {

bindingOptions: {

dataSource: 'productionDataSource'

},

commonSeriesSettings: {

argumentField: 'Timestamp',

type: 'bar'

},

series: [

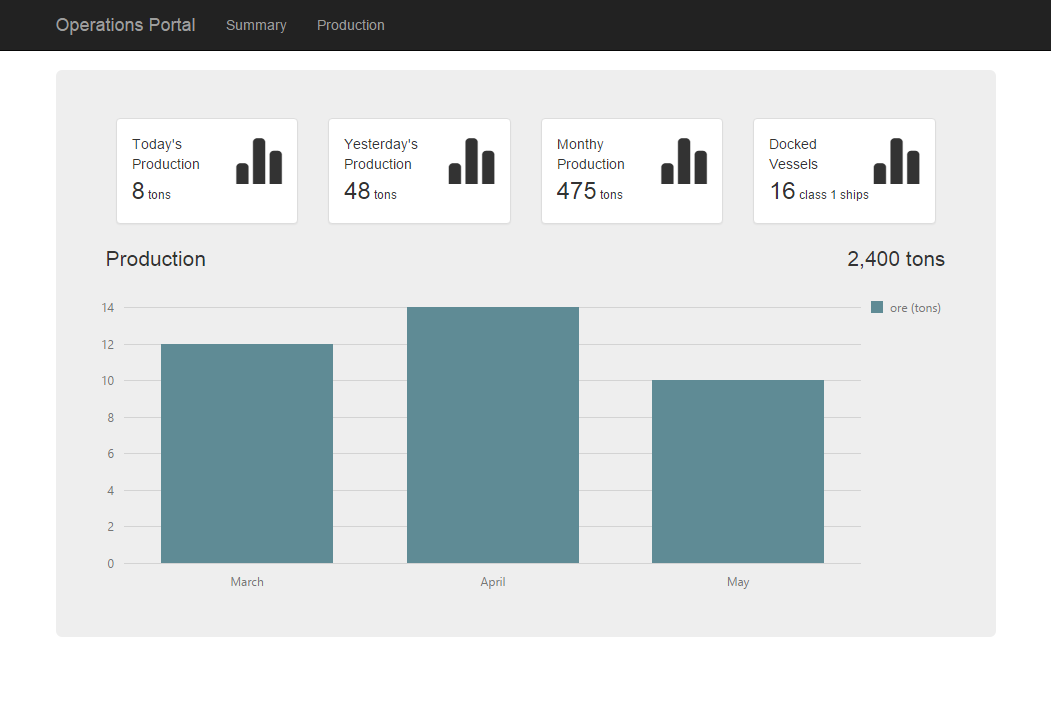
{ valueField: 'Value', name: 'ore (tons)' }

],

};

}]);

* 1. Test



1. Create a ASP.Net Web API Controller (5 minutes)
   1. Create an empty Web API 2 controller called “KPIController” in the controllers folder.
   2. Replace all code in the “KPIController.cs” from with code from the KPIController.txt file.
   3. Change the config.Routes.MapHttpRoute in the “WebAPIConfig.cs” file (found in the App\_Start folder) to:

config.Routes.MapHttpRoute(

name: "ActionApi",

routeTemplate: "{controller}/{action}/{id}",

defaults: new { id = RouteParameter.Optional }

);

* 1. Test with the following url: <http://localhost:60115/kpi/getkpis>, should see data from PI.

1. Explain code to Create a calls using the PI Web API to retrieve data from an AF PI Point Attribute (10 minutes)
2. Create a service in AngularJS that consumes the API call and Populate the chart with the data (5 minutes)
   1. Added a folder to “App” call “Services”
   2. Added a script file to the “Services” folder call “dataService.js” and add:

'use strict';

angular.module('portalApp')

.factory('dataService', ['$http', function ($http) {

return {

getKPIs: function () {

return $http.get('/kpi/getKPIs');

}

};

}]);

* 1. Add the “dataSevices.js” script reference to index.html
  2. Add the dataServices module to the summaryController:

.controller('summaryController', ['$scope','dataService', function($scope, dataService) {

* 1. Add the function to load the data when the view loads (init function), and remove the test data:

$scope.init = function () {

dataService.getKPIs().success(function (data) {

$scope.productionDataSource = data;

}).error(function (err) {

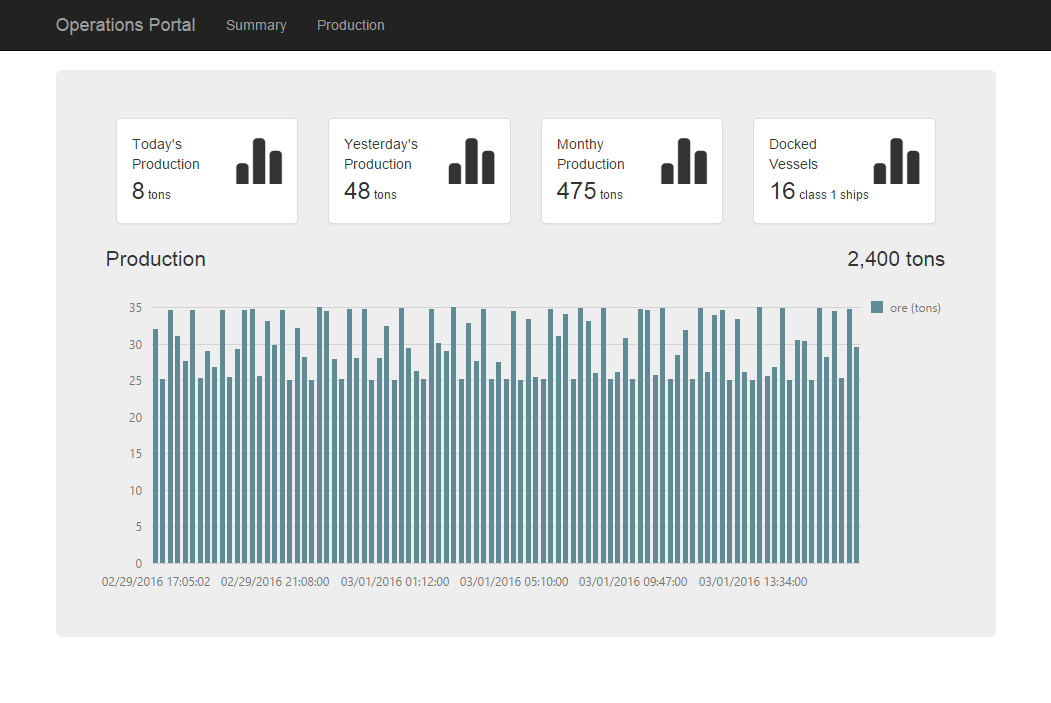
$scope.productionDataSource = [];

});

};

$scope.productionDataSource = [];

* 1. Test, use F12 to see errors



1. Congratulations!!!!!