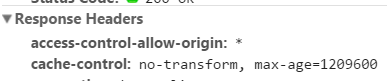
HD3C09 – Using a Cross-Domain Service

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/SAPUI5 | **MOTIVATION**  This case illustrates how to enable CORS and use a publically available weather service.  **PREREQUISITES**  HD1sCO3 – The Base Application |
| **Target Audience**  Undergraduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |

# CORS

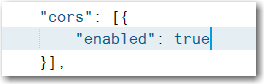
Normally, a browser cannot access resources from a domain other than the domain from which a web page was initially loaded. The domain is defined by the protocol, host name and port of the page’s URL. This is a mechanism to prevent security threats such as cross-site scripting. Cross-domain resource sharing (CORS) allows a browser and server to interact in a way to determine whether cross-domain requests can be handled safely. To enable CORS, a server must add the **access-control-allow-origin** property to the response header of HTTP requests. The image below shows a portion of the response header in Chrome’s Developer Tools. The \* indicates that all domains are allowed. Fortunately, there is a way to enable CORS for HANA applications so that HANA adds the appropriate headers. You can read about CORS in the HANA Developer Guide [here](http://help.sap.com/saphelp_hanaplatform/helpdata/en/a9/fc5c220d744180850996e2f5d34d6c/content.htm).



In this case, our application accesses a publically available service from **Open Weather Map** (<http://openweathermap.org/>). Since the host name of that service is different than the host name of the HANA system, this represents a cross-domain request.

## .xsaccess

To enable CORS open the .xsaccess file and set the value of the cors property to true.



# The Weather Service

The application uses two services: one for the current weather and one for a forecast for a city.

|  |
| --- |
| Open Weather Map have recently changed their API policies and now require that you include an API key in your requests. It’s free and easy to register. Navigate to <http://openweathermap.org/api> and scroll down until you see this:    Click the link How to get API Key and follow the instructions. In this document replace YOUR APIID with your API key. |

## Current Weather

The JSON code below shows the output from the following URL:

http://api.openweathermap.org/data/2.5/weather?q=Sydney&APPID=<YOUR APIID>

This shows the current weather for Sydney. You can find an explanation of the data elements [here](http://bugs.openweathermap.org/projects/api/wiki/Weather_Data).

|  |
| --- |
| {  coord: {  lon: -60.18,  lat: 46.14  },  sys: {  message: 0.0143,  country: "CA",  sunrise: 1429434375,  sunset: 1429483998  },  weather: [  {  id: 801,  main: "Clouds",  description: "few clouds",  icon: "02n"  }  ],  base: "stations",  main: {  temp: 270.386,  temp\_min: 270.386,  temp\_max: 270.386,  pressure: 1034.33,  sea\_level: 1037.24,  grnd\_level: 1034.33,  humidity: 96  },  wind: {  speed: 3.39,  deg: 12.0002  },  clouds: {  all: 12  },  dt: 1429485573,  id: 6354908,  name: "Sydney",  cod: 200  } |

## Forecast

The JSON output below shows a portion of the forecast generated from this URL:

http://api.openweathermap.org/data/2.5/forecast/daily?q=Sydney&units=metric&cnt=5&APPID=<YOUR APIID>

It generates a 5 day forecast in metric units for Sydney.

|  |
| --- |
| {  cod: "200",  message: 0.0298,  city: {  id: 6354908,  name: "Sydney",  coord: {  lon: -60.183102,  lat: 46.135101  },  country: "CA",  population: 0,  sys: {  population: 0  }  },  cnt: 5,  list: [  {  dt: 1429455600,  temp: {  day: -0.77,  min: -4.85,  max: -0.77,  night: -4.85,  eve: -1.03,  morn: -0.77  },  pressure: 1031.76,  humidity: 97,  weather: [  {  id: 803,  main: "Clouds",  description: "broken clouds",  icon: "04d"  }  ],  speed: 4.4,  deg: 12,  clouds: 76  }, |

# Create a Weather View

### Add the View to the App Navigation

Add a new object for the Weather view to the views.json file. You can find and icon [here](https://openui5.hana.ondemand.com/iconExplorer.html). Also, add a route to the Component.js file.

Create a view called Weather in the view package.

### Weather.view.xml

Add the code shown below to the Shell.view.xml file:

|  |
| --- |
| <core:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" controllerName="ui5.controller.Weather" xmlns:commons="sap.suite.ui.commons">  <Page title="Start Coding Cross-Origin Resource Sharing!" id="idPage"  showNavButton="true" navButtonPress="handleNavButtonPress">  <content>  <VBox class="marginBoxContent">  <items>    <ObjectHeader id="details" title="{weather>/name}"  number="{weather>/main/temp}"  numberUnit="Celsius"  icon="{weather>/weather/0/icon}"  backgroundDesign="Transparent" intro="Weather Forecast"  iconActive="true">  <attributes>  <ObjectAttribute text="{weather>/weather/0/description}" />  <ObjectAttribute  text="{weather>/wind/speed},{weather>/wind/deg}" />  </attributes>  </ObjectHeader>  <Table id="idForecastTable" inset="false"  items="{forecast>/list}">  <headerToolbar>  <Toolbar>  <Label text="Forecast"></Label>  <ToolbarSpacer></ToolbarSpacer>  <Input id="inpSearch" editable="true" value="" maxLength="80" width="200px"/>  <Button id="bntSearch" text="Search" tap="actSearch" />  <ToolbarSpacer></ToolbarSpacer>  <Button icon="sap-icon://refresh"  press="handleRefreshPressed" />  </Toolbar>  </headerToolbar>  <columns>  <Column>  <header>  <Text text="Date" />  </header>  </Column>  <Column minScreenWidth="Tablet" demandPopin='true'>  <header>  <Text text="Wind" />  </header>  </Column>  <Column minScreenWidth="Tablet" demandPopin='true'>  <header>  <Text text="Hi" />  </header>  </Column>  <Column minScreenWidth="Tablet" demandPopin='true'>  <header>  <Text text="Low" />  </header>  </Column>  <Column>  <header>  <Text text="Forecast" />  </header>  </Column>  <Column>  <header>  <Text text="Temperature" />  </header>  </Column>  </columns>  <items>  <ColumnListItem>  <cells>  <ObjectIdentifier  title="{forecast>dt}"  text="{forecast>weather/0/description}" />  <ObjectIdentifier  title="{forecast>speed},  {forecast>deg}" />  <ObjectNumber  number="{forecast>temp/max}"  unit="Celsius" />  <ObjectNumber  number="{forecast>temp/min}"  unit="Celsius" />  <Image src="{forecast>weather/0/icon}" />  <ObjectNumber  number="{forecast>temp/day}"  unit="Celsius" />  </cells>  </ColumnListItem>  </items>  </Table>  </items>  </VBox>  </content>  </Page>  </core:View> |

Listing 1

This code organizes the three main elements of the page using a [VBox](https://openui5.hana.ondemand.com/docs/api/symbols/sap.m.VBox.html) control, which is used to arrange elements vertically. The first element on the page is an input control and button which are arranged horizontally using an [HBox](https://openui5.hana.ondemand.com/docs/api/symbols/sap.m.HBox.html) control. This Input control allows a user to enter a city. Next, an [ObjectHeader](https://sapui5.hana.ondemand.com/sdk/explored.html#/entity/sap.m.ObjectHeader/samples) control is created to show the current weather. The attributes of the ObjectHeader control are bound to properties in the weather model which returns the current weather. You can use the JSON example above to examine how the various fields are accessed. For example, this binding number="{weather>/main/temp}" refers to the portion of the output shown below. The name weather> refers to the SAPUI5 model defined in the controller. The path /main/temp indicates the value to bind to the number attribute.

|  |
| --- |
| …  base: "stations",  main: {  temp: 270.386,  … |

A more interesting example is text="{weather>/weather/0/description}". This refers to the weather property in the output which is an array with only one element. The 0 is the index of the array element and description is a property of the object within that array element.

|  |
| --- |
| weather: [  {  id: 801,  main: "Clouds",  description: "few clouds",  icon: "02n"  }  ], |

There a few things to note on the [Table](https://sapui5.hana.ondemand.com/sdk/explored.html#/entity/sap.m.Column/samples) control which follows the ObjectHeader. The first is that the items attribute is assigned to the list array in the forecast model. This will cause it to create a table row for each object in that array. The URL used to retrieve the forecast specifies a 5 day forecast so five rows will be created. The Table also has a toolbar that allows the user to refresh the models. This is necessary because, since this is a free service, it sometimes does not respond if the servers are busy.



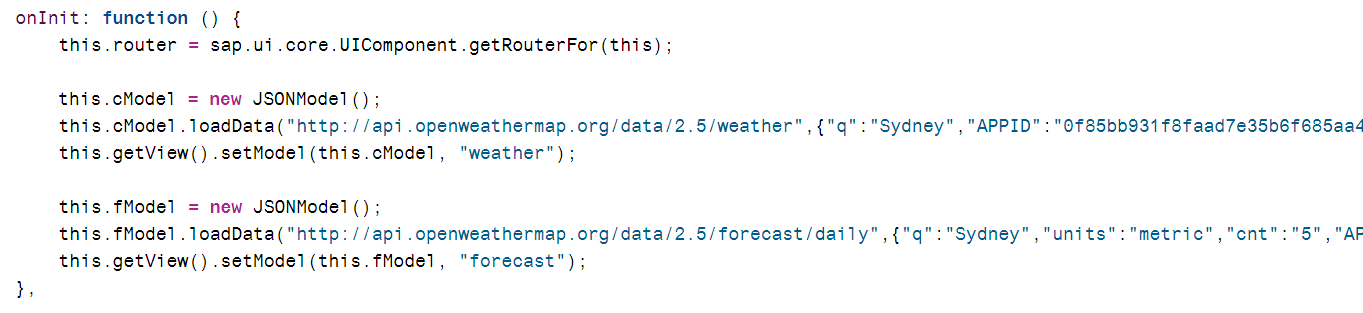
### Weather.controller.js

Add the code shown below to the Weather.controller.js file:

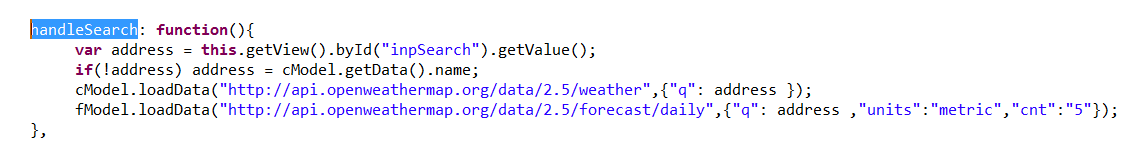
|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/ui/model/json/JSONModel"  ], function(Controller,JSONModel) {  "use strict";  return Controller.extend("ui5.controller.Weather", {  onInit: function () {  this.router = sap.ui.core.UIComponent.getRouterFor(this);    this.cModel = new JSONModel();  this.cModel.loadData("http://api.openweathermap.org/data/2.5/weather",{"q":"Sydney","APPID":" YOUR APPID "})  this.getView().setModel(this.cModel, "weather");    this.fModel = new JSONModel();  this.fModel.loadData("http://api.openweathermap.org/data/2.5/forecast/daily",{"q":"Sydney","units":"metric","cnt":"5","APPID":" YOUR APPID "});  this.getView().setModel(this.fModel, "forecast");  },    handleNavButtonPress: function(){  this.router.navTo("Master", {  from: "IconTab"  });  },    actSearch: function(){  var address = this.getView().byId("inpSearch").getValue();  this.cModel.loadData("http://api.openweathermap.org/data/2.5/weather",{"q": address,"APPID":" YOUR APPID "});  this.fModel.loadData("http://api.openweathermap.org/data/2.5/forecast/daily",{"q": address ,"units":"metric","cnt":"5","APPID":" YOUR APPID "});  },    handleRefreshPressed: function(){  var address = this.getView().byId("inpSearch").getValue();  if(!address) address = this.cModel.getData().name;  this.cModel.loadData("http://api.openweathermap.org/data/2.5/weather",{"q": address,"APPID":" YOUR APPID "});  this.fModel.loadData("http://api.openweathermap.org/data/2.5/forecast/daily",{"q": address ,"units":"metric","cnt":"5","APPID":"YOUR APPID"});  }  });  }); |

Listing 2

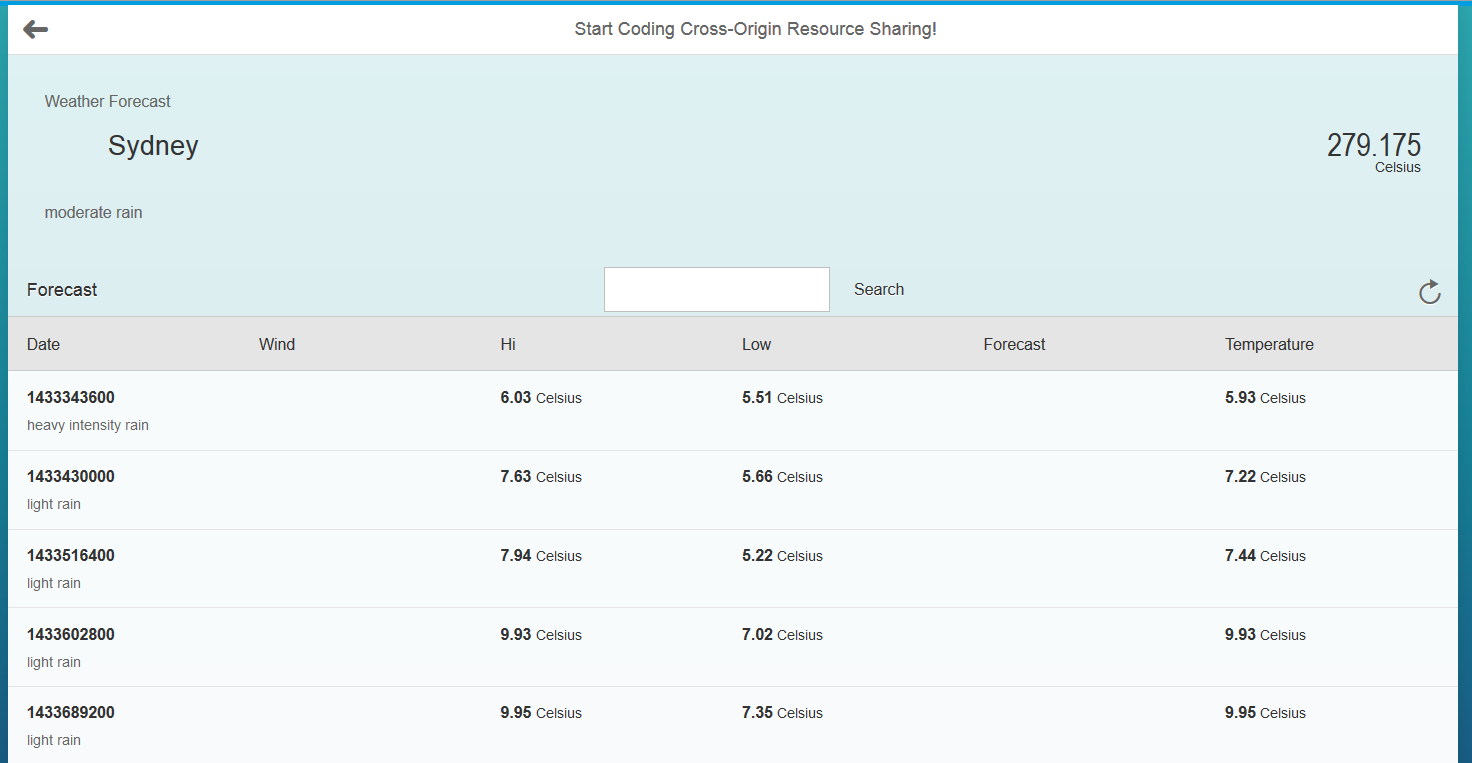
The onInit function is executed when the view is first initialized. It creates the two models and assigns the models to the View. The second argument of the loadData() method, specifies query parameters that will be attached to the URL in a format like the URLs shown earlier in this case. **NOTE you must replace the ‘YOUR APPID’ with the APPID from Open Weather Map.**



The handleSearch function is invoked when the user enters a new city and presses the search button or if the user clicks the refresh button. The code retrieves the value from the input box. It then checks to see if the box was empty. If it is, it retrieves the last city retrieved. Finally, it uses the loadData() function to retrieve the data inserting the address variable into the query parameters.



Run the application.



It’s a good start but there are a number of improvements to make. We’ll add some formatter functions to format various values and add a graph to show the temperatures in the forecast.

## Add Formatter Functions

You may notice that there are a number of anomalies in the application as it stands now. For example, some of the temperatures are actually in Kelvin rather than Celsius. Also, the dates are not in a good format. These values must be formatted before being displayed. In XML views this is handled using formatter functions. We’ll add several formatter functions.

### Format the Date

Add the formatter function to **Weather.controller.js.**

|  |
| --- |
| date: function(date){  return new Date(date\*1000).toLocaleDateString();  } |

Listing 3

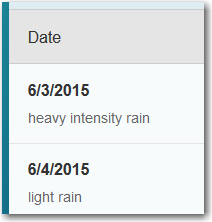
Now, update the highlighted portion in the code below of the ObjectIdentifer for the first table column.

|  |
| --- |
| <ObjectIdentifier  title="{ path: 'forecast>dt', formatter: '.date' }"  text="{forecast>weather/0/description}" /> |

Listing 4

The value from the service is passed to the formatter function as the date parameter. The function then returns the transformed value before it is displayed. You can also include multiple fields in the formatter function by using [extended syntax](http://help.sap.com/saphelp_uiaddon10/helpdata/en/0c/803921b1bf4b3a97a038fbd51ef542/content.htm?frameset=/en/a2/fe8e763014477e87990ff50657a0d0/frameset.htm&current_toc=/en/e4/843b8c3d05411c83f58033bac7f072/plain.htm&node_id=324&show_children=false).

The dates look much better.



## Format the Temperatures

Add the formatter function to **Weather.controller.js.**

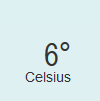
|  |
| --- |
| celsius: function(temp){  temp = parseFloat(temp);  var degreesCelsius = Math.round(temp - 273.15);  return degreesCelsius + "\u00b0";  } |

Listing 5

Now update the number attribute of the current weather ObjectHeader

|  |
| --- |
| number="{ path: 'weather>/main/temp', formatter:'.celsius' }" |

Listing 6



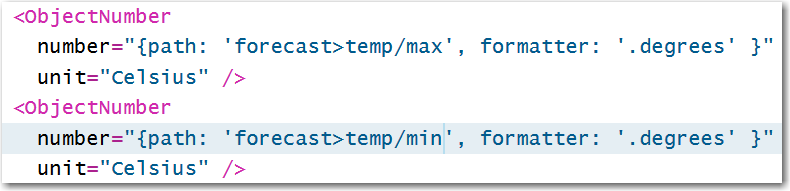
### Add a Degrees Symbol to the Forecast Temperatures

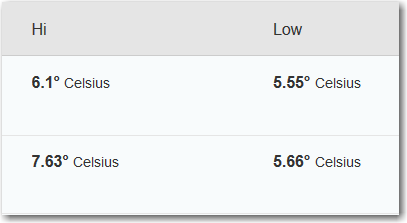
Add the formatter function to **Weather.controller.js.**

|  |
| --- |
| degrees: function(temp){  return temp + "\u00b0";  } |

Listing 7

Update the min and max temperatures in the Table control.





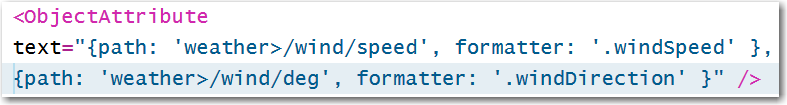
### Format the Wind Speed and Direction

Add the formatter function to **Weather.controller.js.**

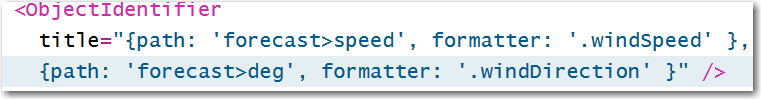
|  |
| --- |
| windDirection: function(direction){  return Math.round(direction) + "\u00b0";  },    windSpeed: function(speed){  return speed + " kph";  } |

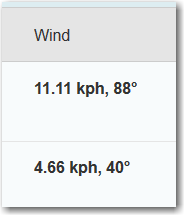
Listing 8

Update the title with the wind speed and direction in the ObjectHeader:



And in the second column of the Table.





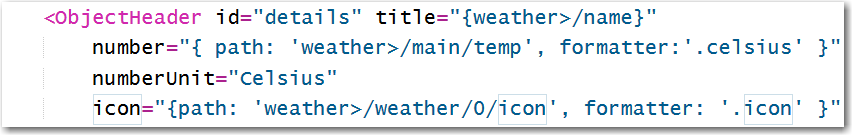
### Format the Icon

The weather service provides the name of an image file that reflects the weather condition but in order to use it, we have to attach the URL and file extension. Add this formatter function to Weather.controller.js.

|  |
| --- |
| icon: function(icon){  return "http://openweathermap.org/img/w/" + icon + ".png";  } |

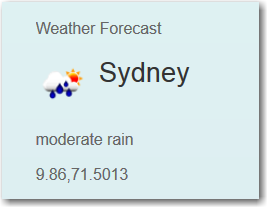
Listing 9

The icons appear in two places. One in the current weather ObjectHeader:



And one in the table:





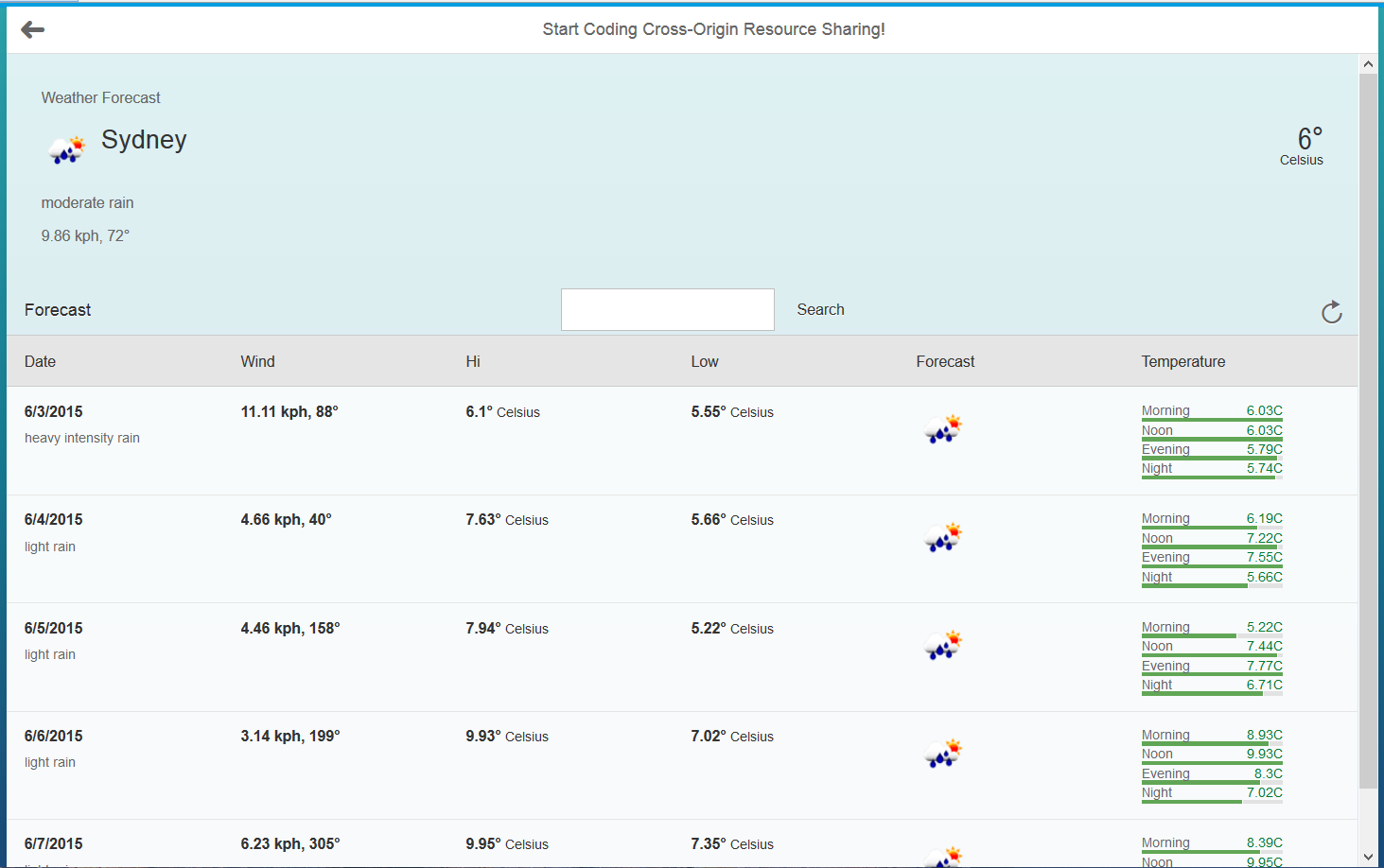
## Add a Chart

Finally, to show the forecasted temperatures, we’ll add a [ComparisonChart](https://sapui5.hana.ondemand.com/sdk/explored.html#/entity/sap.suite.ui.commons.ComparisonChart/samples) to the last column of the table. Replace the ObjectNumber in the last column of the table with this code:

|  |
| --- |
| <commons:ComparisonChart size="S" scale="C"  class="marginTopLeft">  <commons:data>  <commons:ComparisonData title="Morning"  value="{forecast>temp/morn}" color="Good" />  <commons:ComparisonData title="Noon"  value="{forecast>temp/day}" color="Good" />  <commons:ComparisonData title="Evening"  value="{forecast>temp/eve}" color="Good" />  <commons:ComparisonData title="Night"  value="{forecast>temp/night}" color="Good" />  </commons:data>  </commons:ComparisonChart> |

Listing 10





# Exercise

Use the [ObjectStatus](https://sapui5.netweaver.ondemand.com/sdk/explored.html#/entity/sap.m.ObjectStatus/samples) control to add the sunrise and sunset times to the ObjectHeader. The formatter for the time is similar to the date formatter except it uses the **.toLocaleTimeString()** function to format the result.

