## References

* .NET MVC
  + <https://www.asp.net/mvc>
* Bootstrap
  + <https://getbootstrap.com/>
* Chart.js
  + <http://www.chartjs.org/>
* SignalR
  + <https://www.asp.net/signalr>
* Tutorial: Getting Started with SignalR 2 and MVC 5
  + <https://www.asp.net/signalr/overview/getting-started/tutorial-getting-started-with-signalr-and-mvc>
* Using SignalR with Web Apps in Azure App Service
  + <https://www.asp.net/signalr/overview/deployment/using-signalr-with-azure-web-sites>

## Requirements

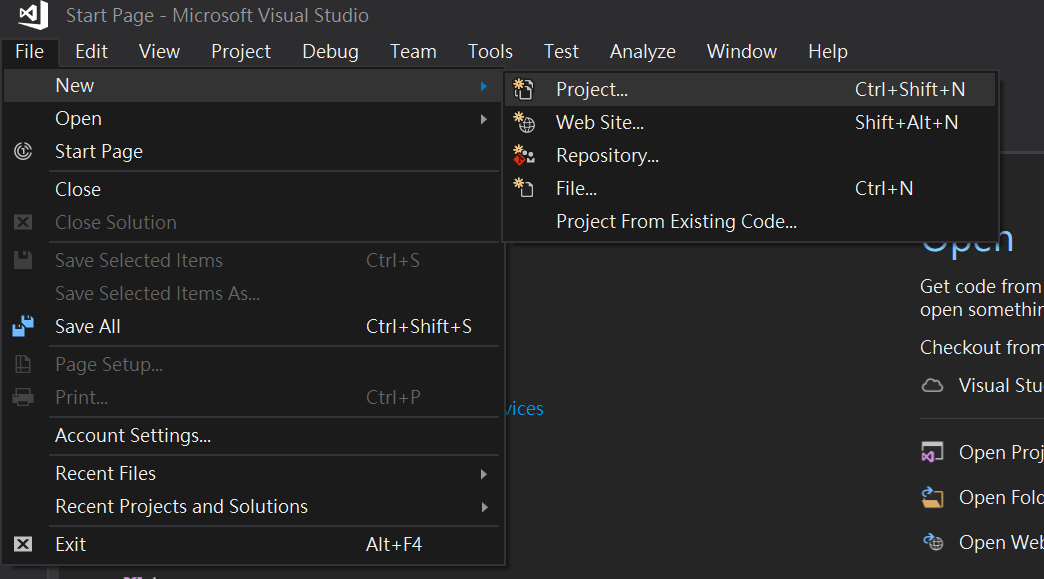
* Azure Subscription
* Visual Studio
* Finished the part 4 of HOL

## Goals

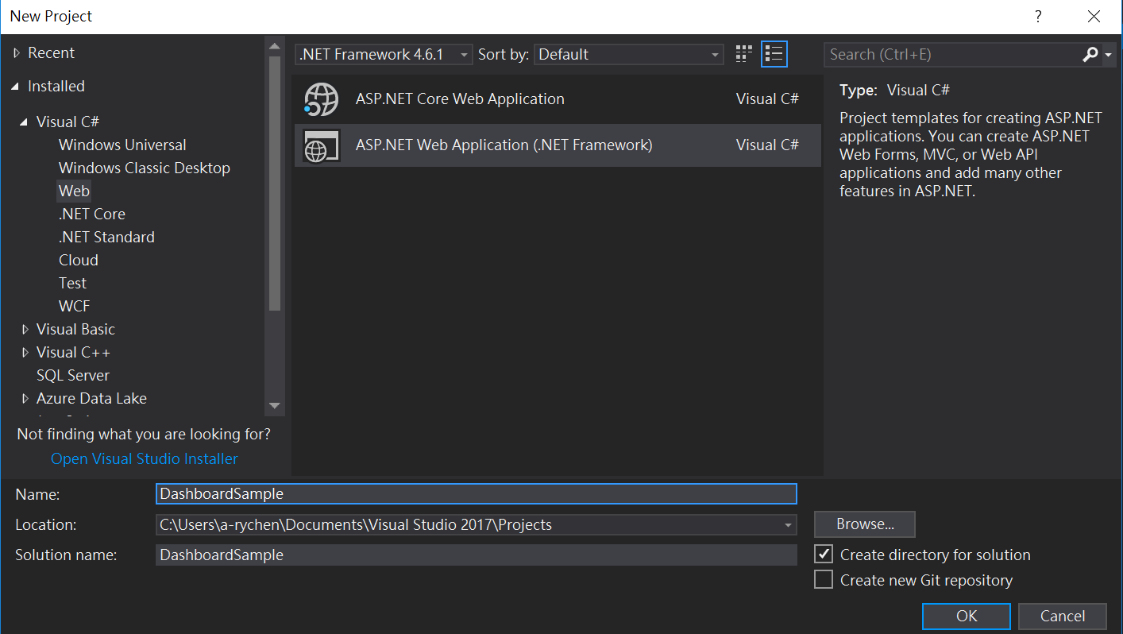
* Customize a dashboard UI.
* Use the **Signal R** to communication between Web Server and Browser Client.
* Use the **Event Processor Host** to push the telemetry data to Web Server.
* Use the **Service Bus** to push the alert data to Web Server.
* Show the real time dashboard in the web app.

## Step 1: Create a .NET MVC project

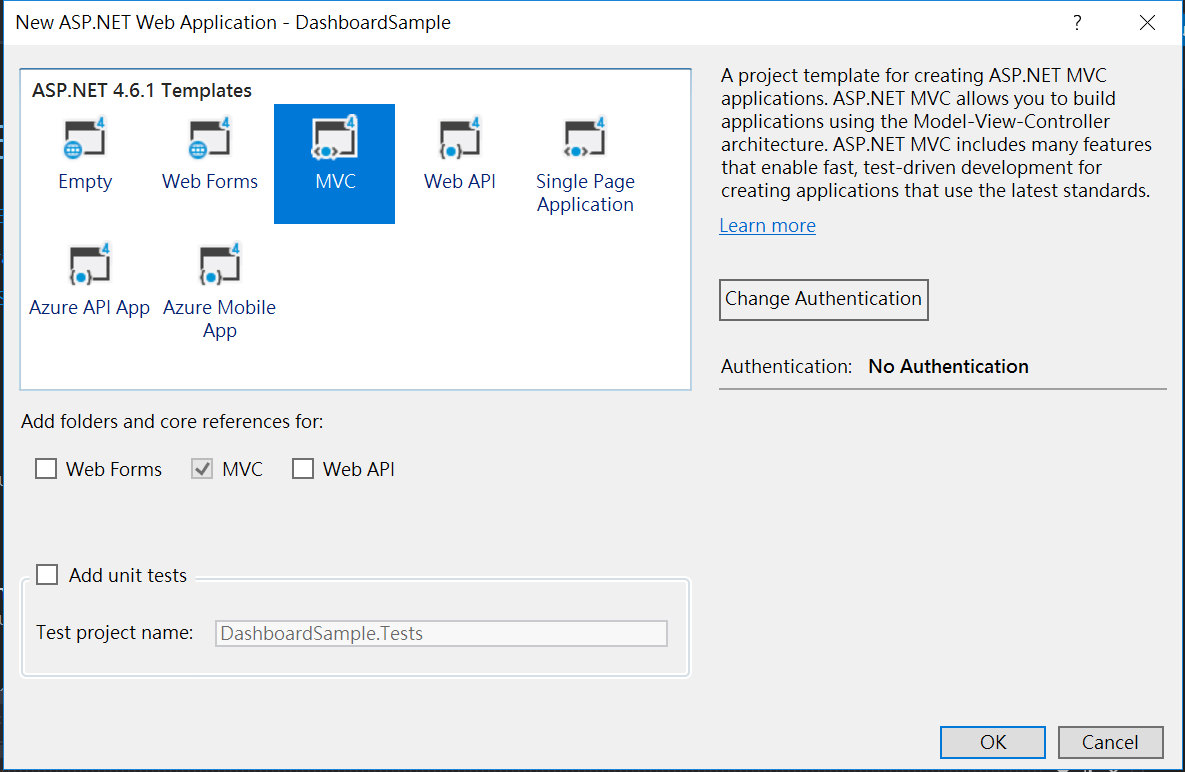
* Click **File** >> **New** >> **Project** on Visual Studio



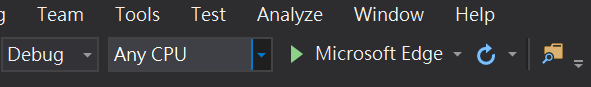
* Select **Web** under **Visual C#**.
* Select **ASP.NET Web Application(.NET Framework),** and naming project name with **DashboardSample**, then click **OK**. (The project name is fixed in this workshop)

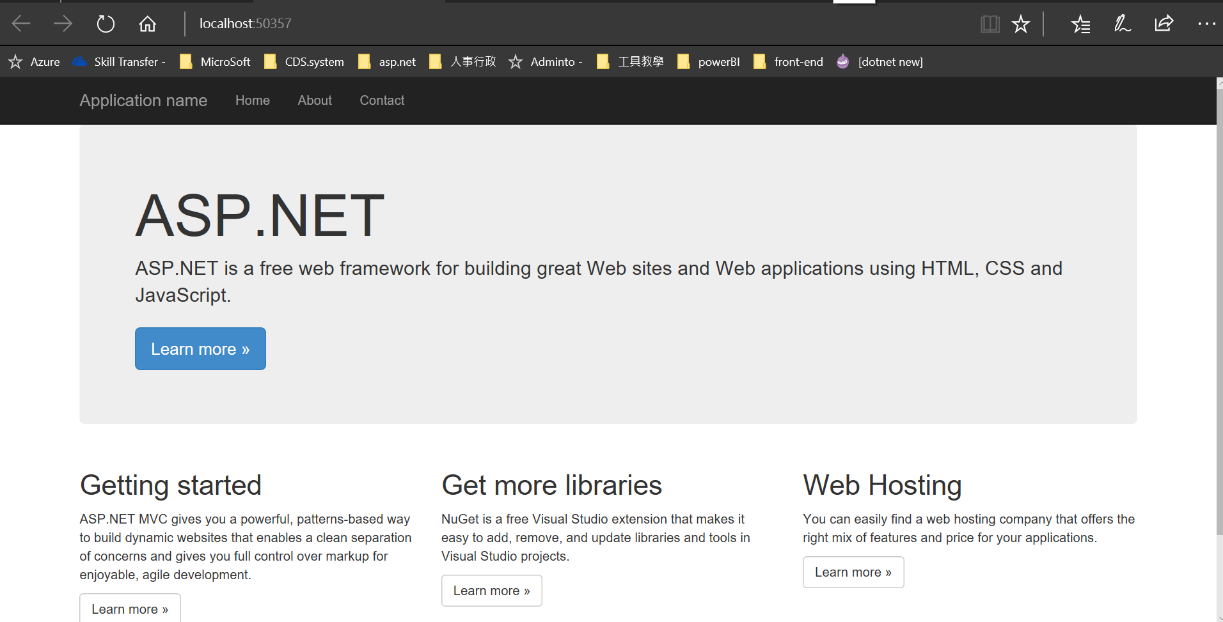


* Select **MVC** template, and click OK.



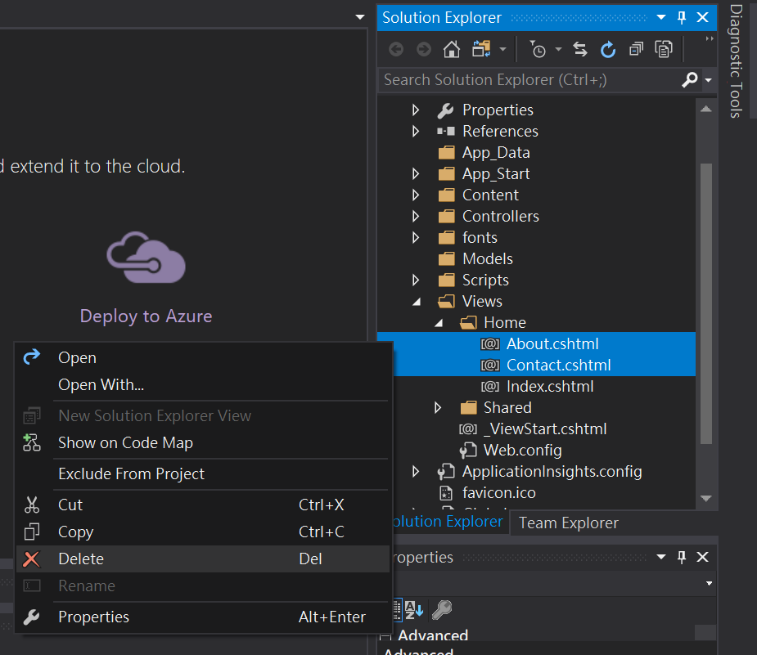
* Build the project, we will see a default web page at localhost.





## Step 2: Customize Dashboard User Interface

* Delete unnecessary files and code.
  + Delete **About.cshtml** and **Contact.cshtml** in **Views/Home**.



* + Delete below code in **Controllers/HomeController.cs**

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

public ActionResult Contact()

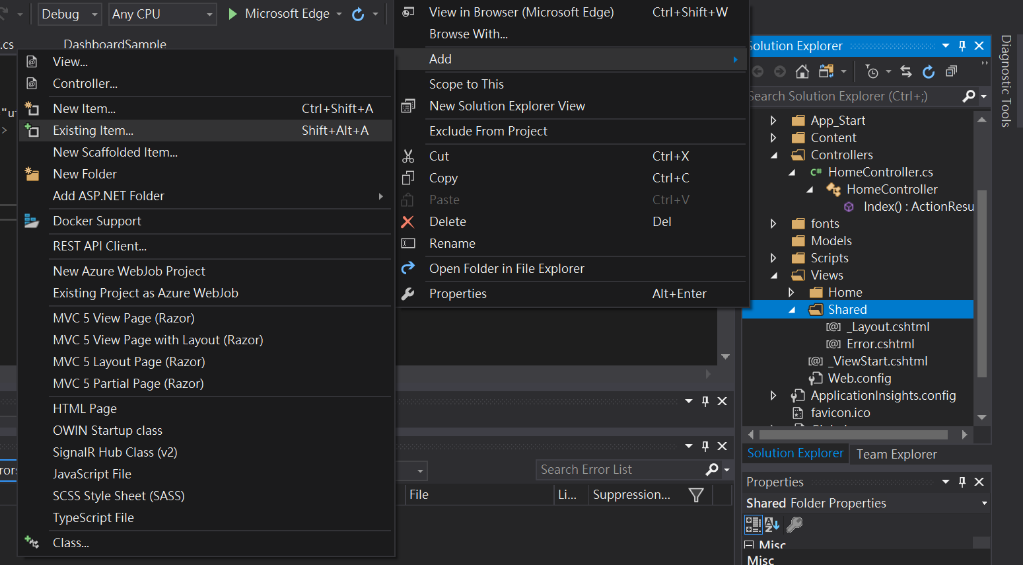
{

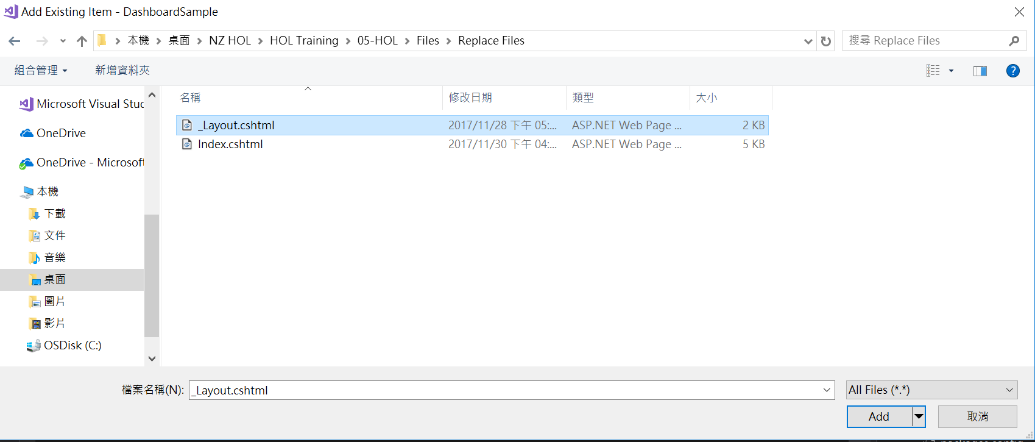
ViewBag.Message = "Your contact page.";

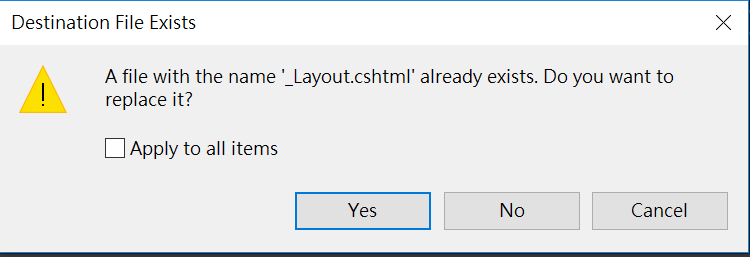
return View();

}

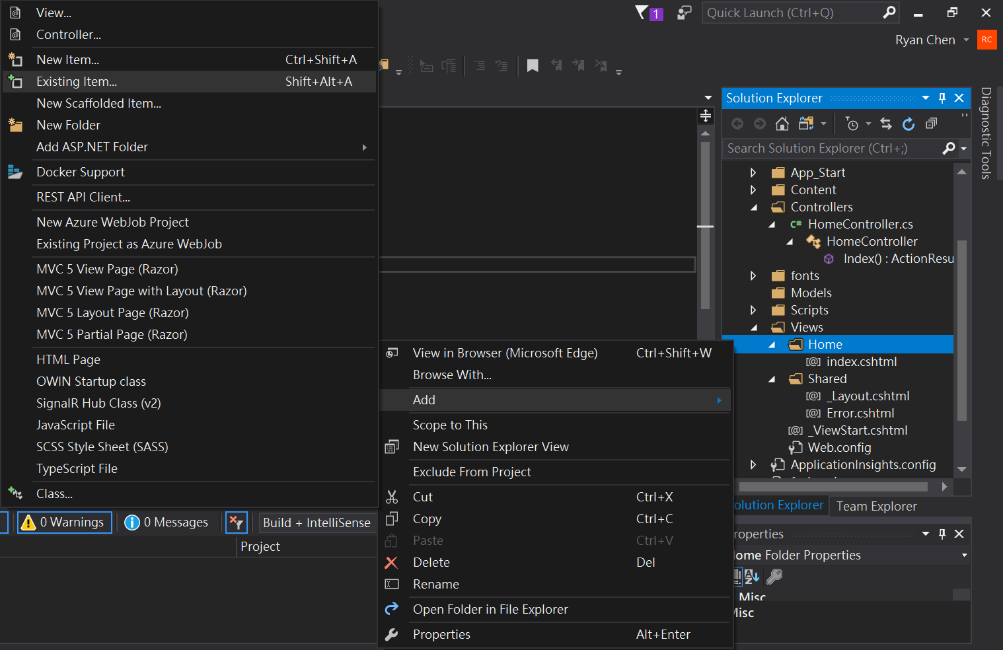
* Replace HTML files for customized dashboard web app
  + Right-click **DasboardSample/Views/Shared**
  + Select **Add >> Existing Item**
  + Select **05-HOL**>> **Files** >> **Replace Files** >> **\_Layout.cshtml**
  + Click **Add**
  + Click **Yes** to apply the file replacement

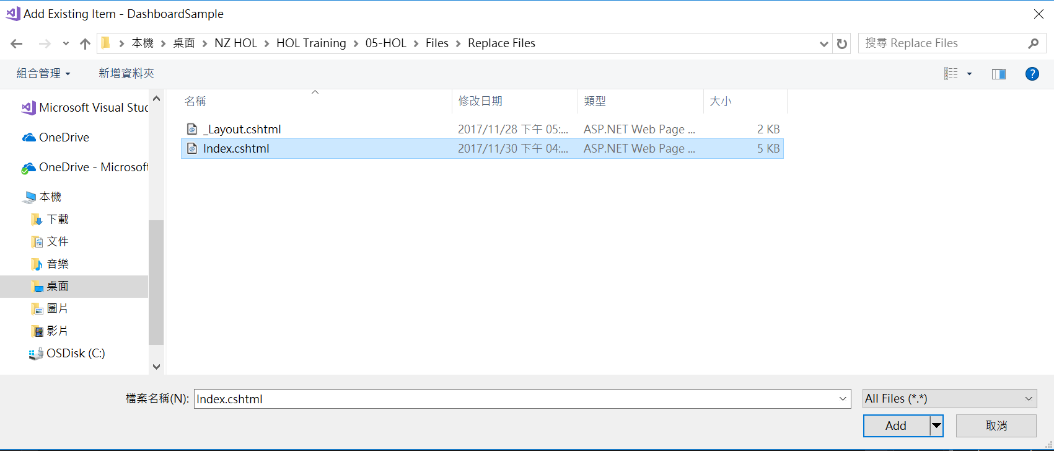


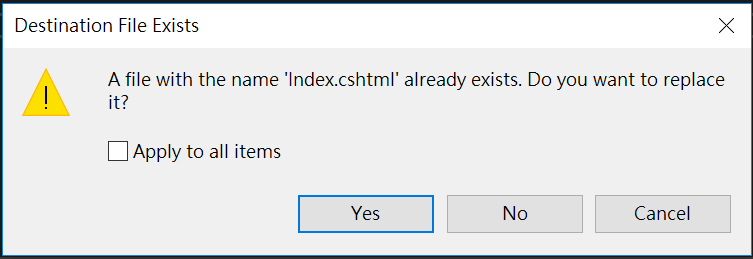




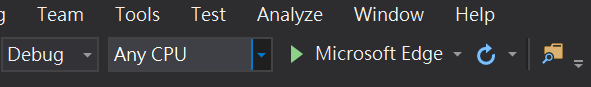
* + Right-click **DasboardSample/Views/Home**
  + Select **Add >> Existing Item**
  + Select **05-HOL**>> **Files** >> **Replace Files** >> **Index.cshtml**
  + Click **Add**
  + Click **Yes** to apply the file replacement

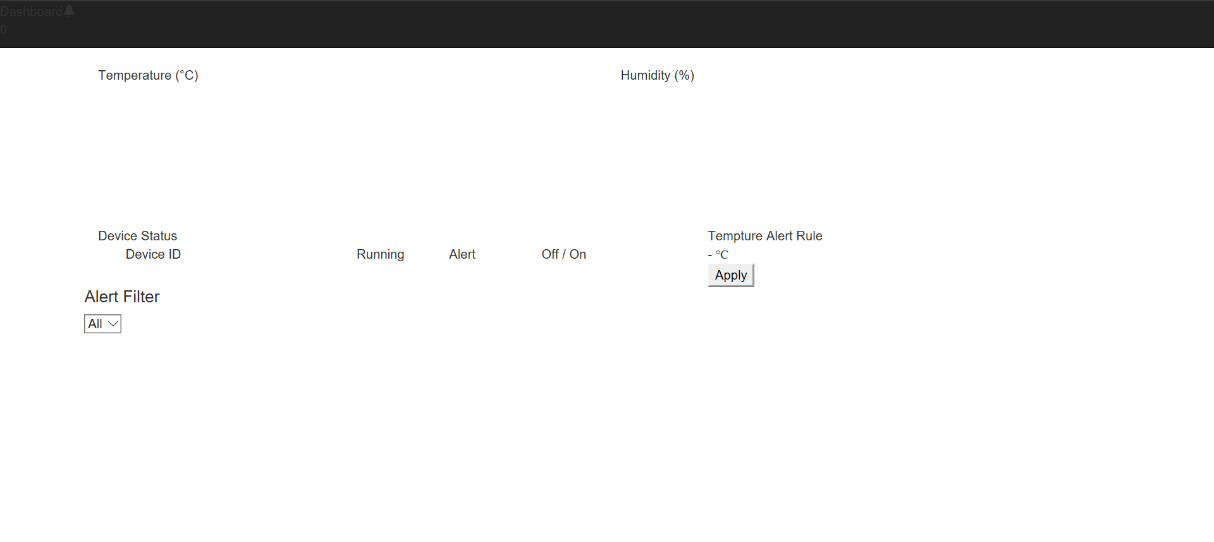




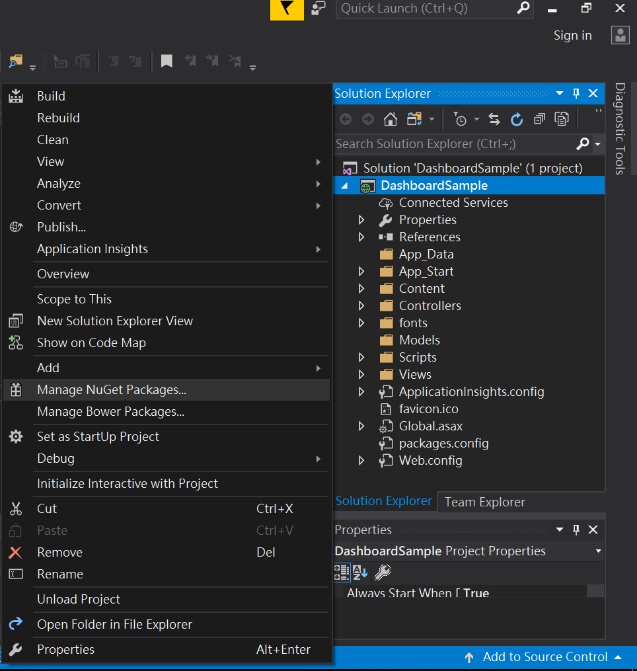


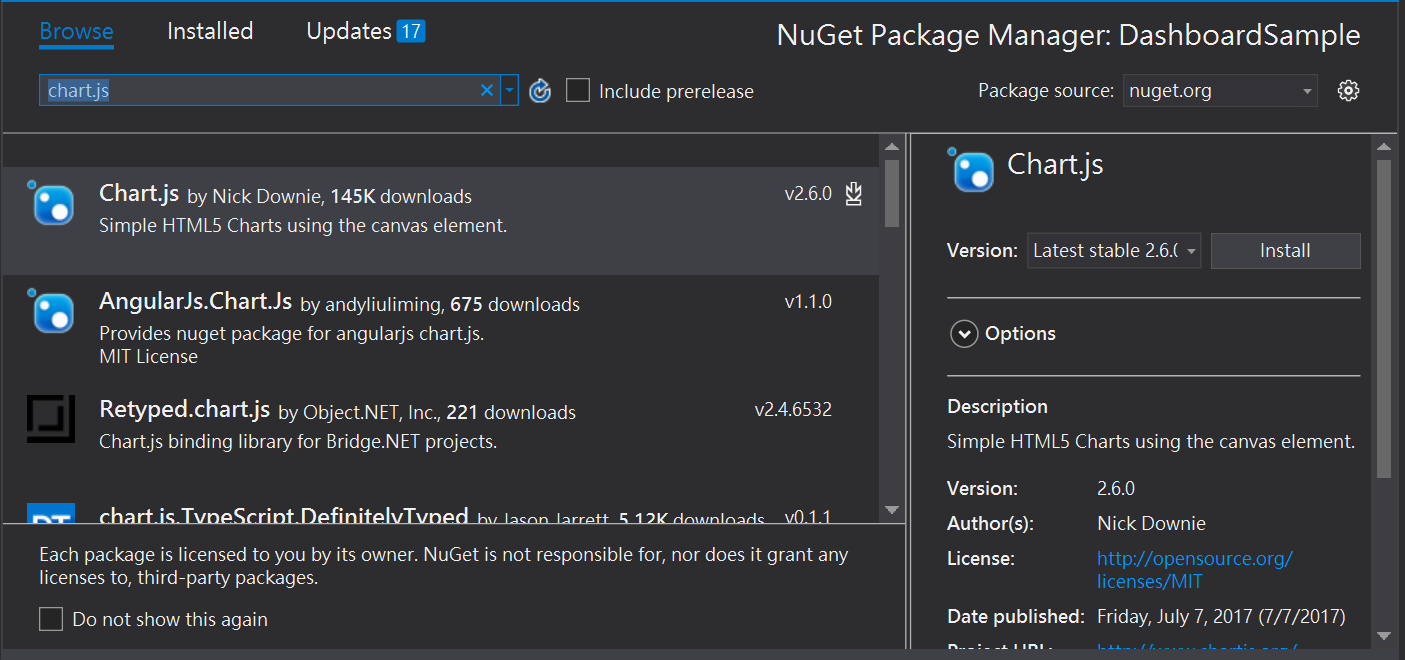
* + Build and run the project. For now, the web page is changed, but it looks weird. That’s because we don’t have any css files to style our web app. And We also need some javascript files to make the UI can interact with user.



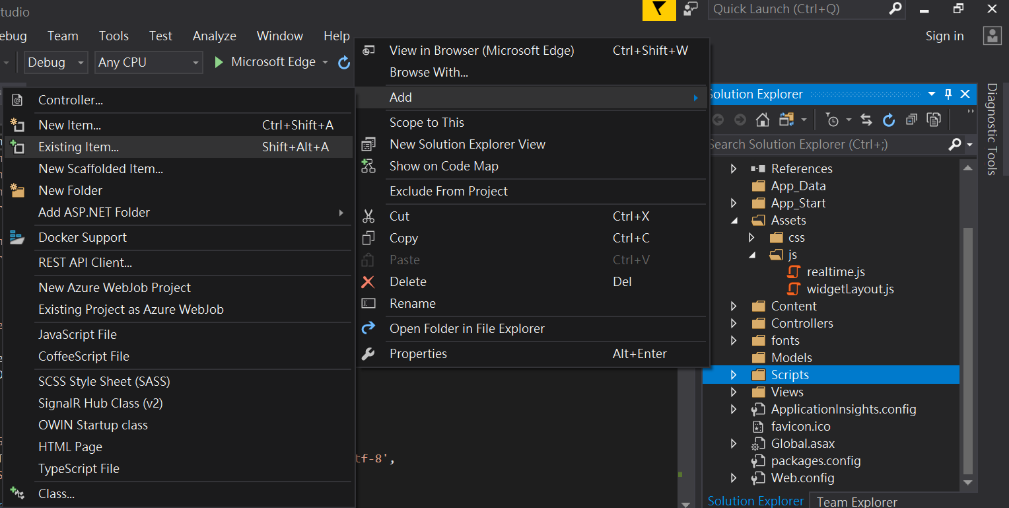


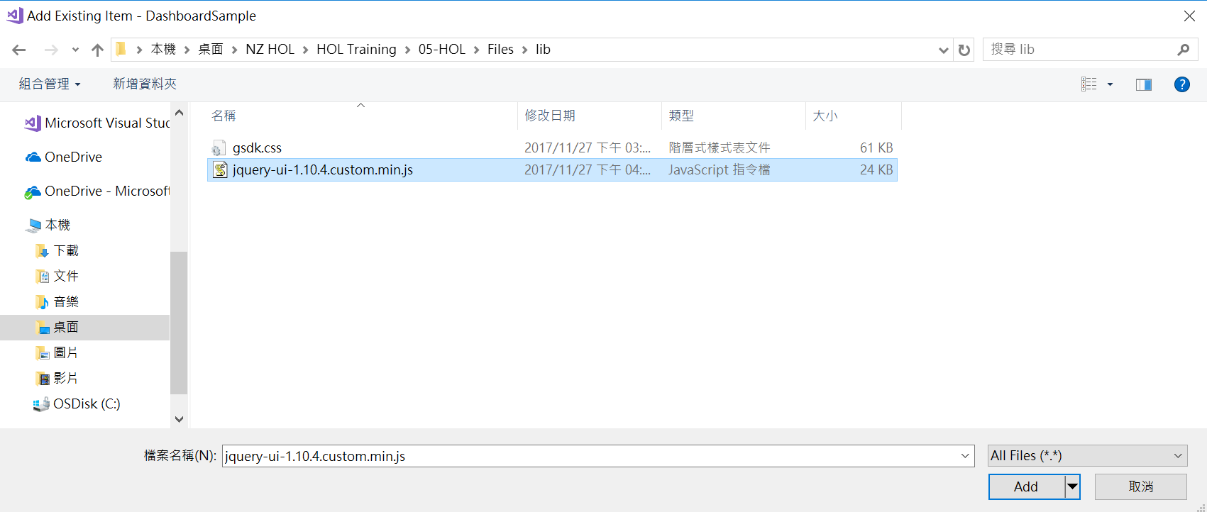
* Add the css and js reference to render the real time line chart.
  + Right click **DashboardSample** project.
  + Click **Manage NuGet packages**
  + **Browse** to find **Chart.js**, then install this package.



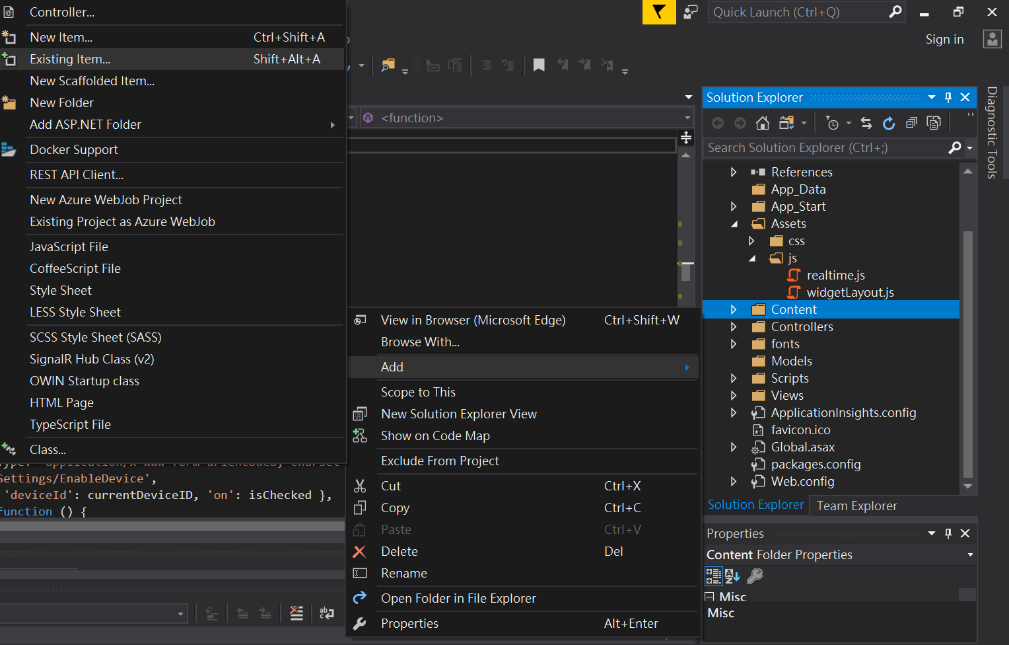


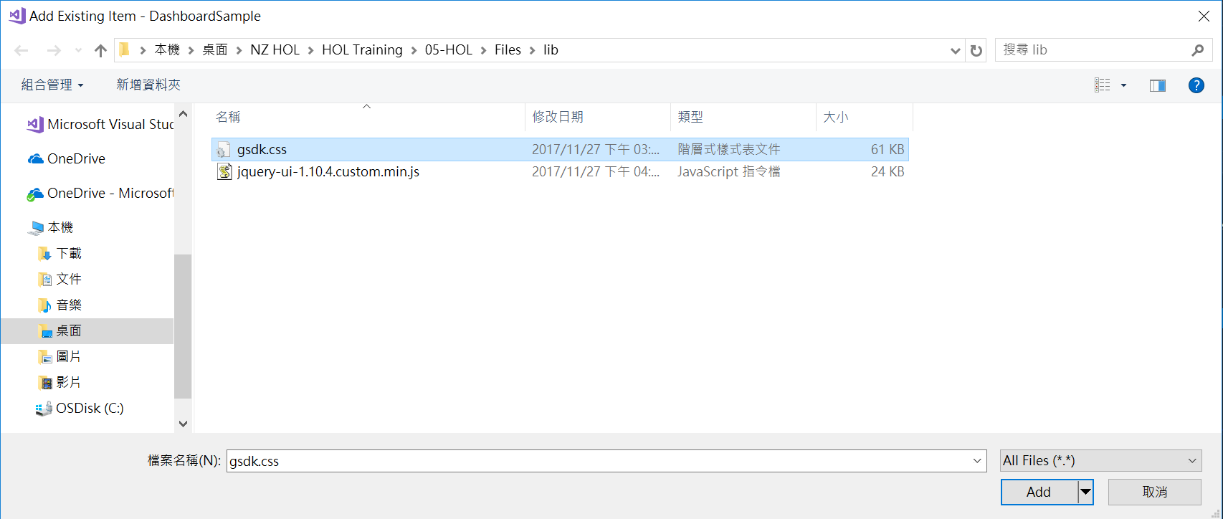
* + Right-click **DasboardSample/Scripts**
  + Select **Add** >> **Existing Item**
  + Select **05-HOL** >> **Files** >> **lib** >> **jquery-ui-1.10.4.custom.min.js**
  + Click **Add**



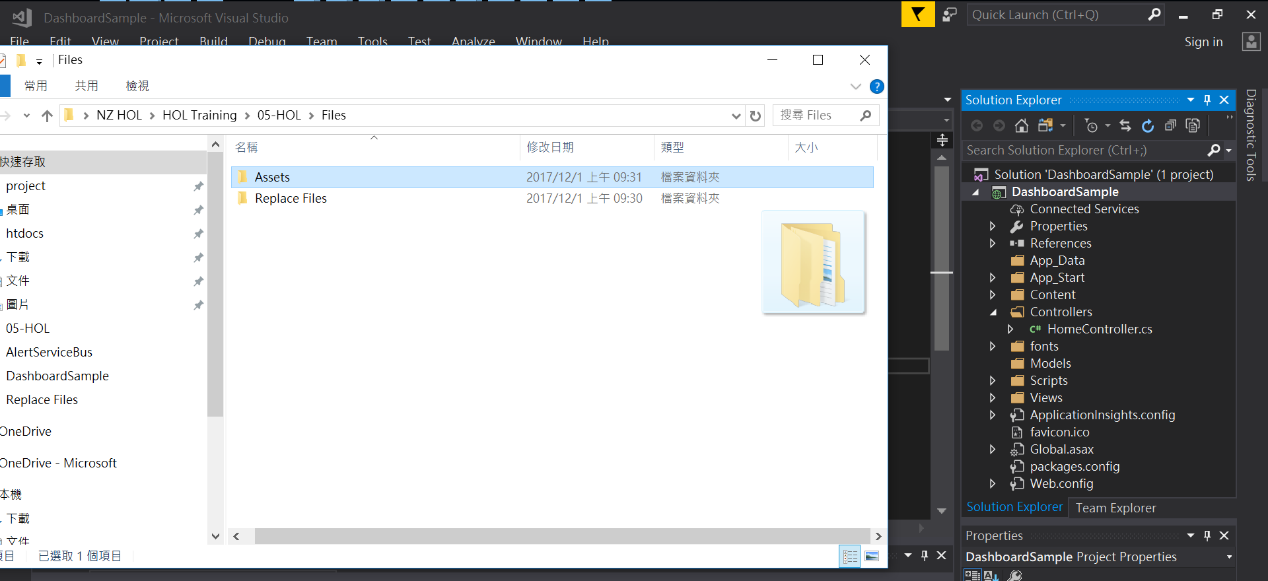


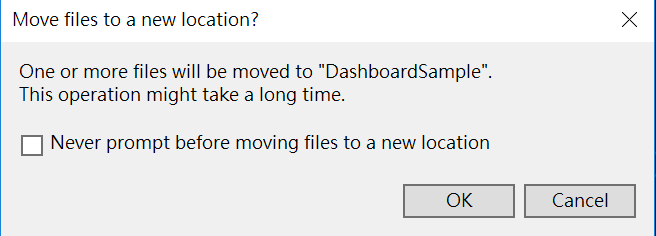
* + Right-click **DasboardSample/Content**
  + Select **Add >> Existing Item**
  + Select **05-HOL**>> **Files** >> **lib** >> **gsdk.css**
  + Click **Add**

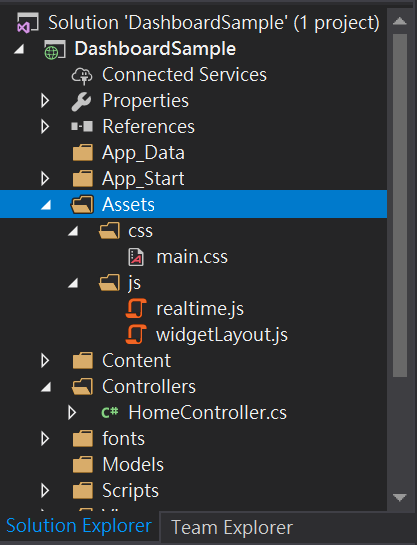




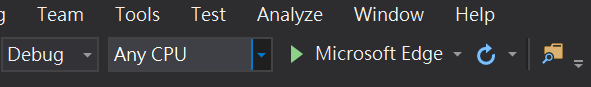
* + Drag **05-HOL** >> **Files** >> **Assets** into **DashboardSample** project.
  + Click OK in **alert** window.
  + The Assets folder will show in the project.

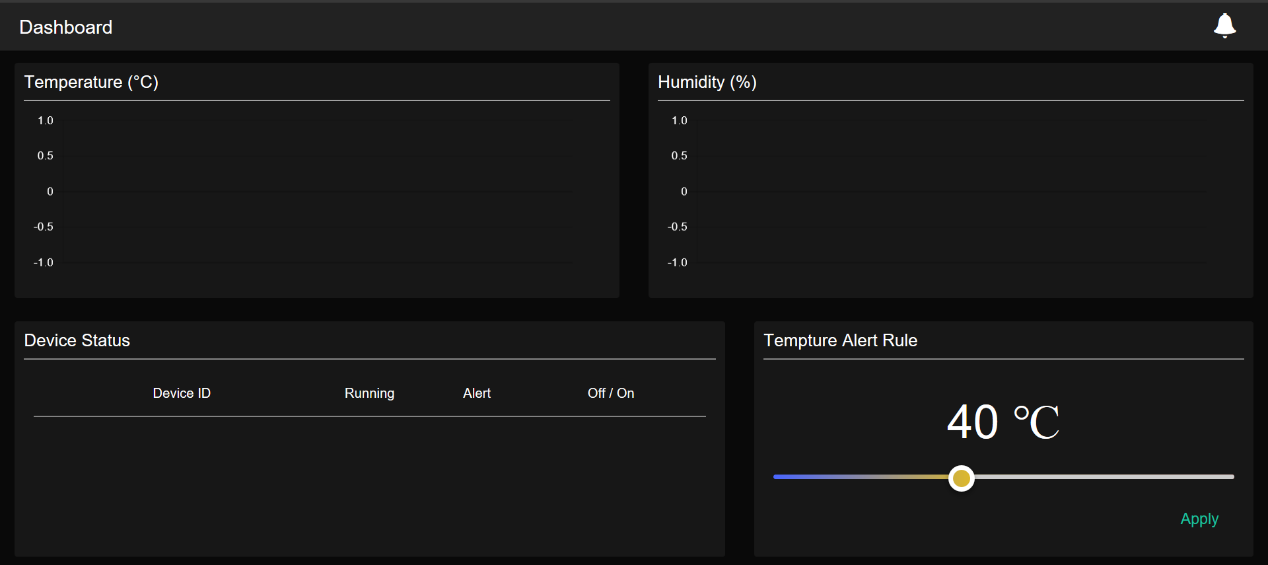






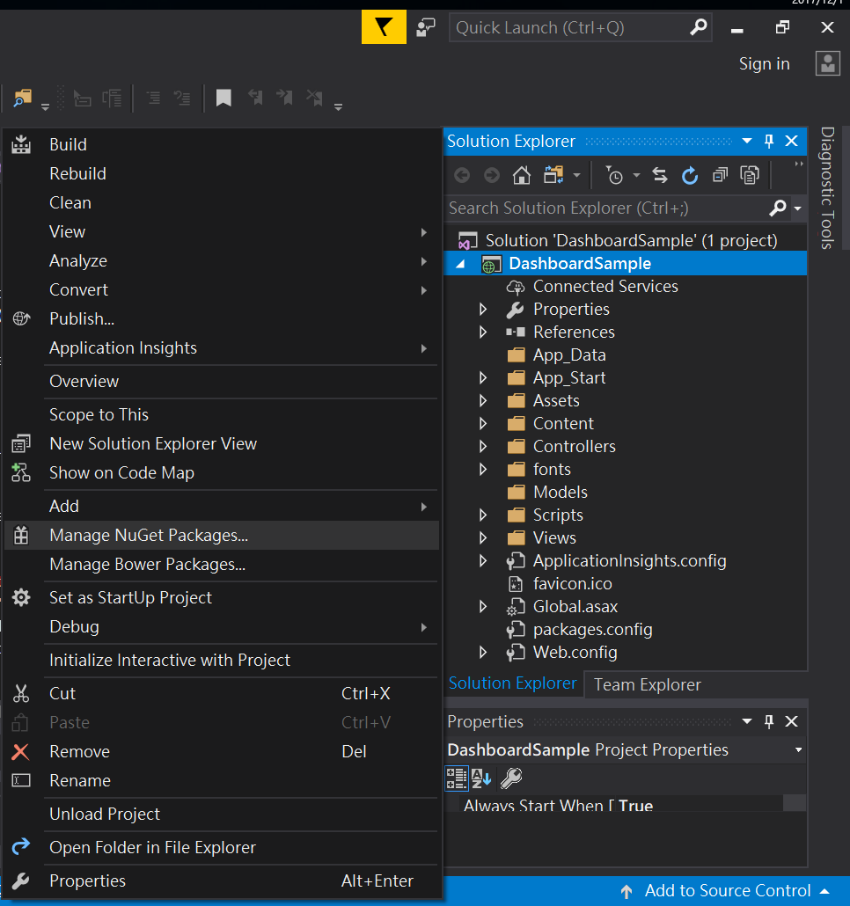
* + Update these packages to the correct version through **Manage NuGet Packages**
    - Bootstrap 🡪 v3.0.0
    - jQuery 🡪 1.10.2
    - jQuery.Validation 🡪 1.11.1
    - Microsoft.jQuery.Unobtrusive.Validation 🡪 3.2.3
  + Build and run the project, now it looks better.



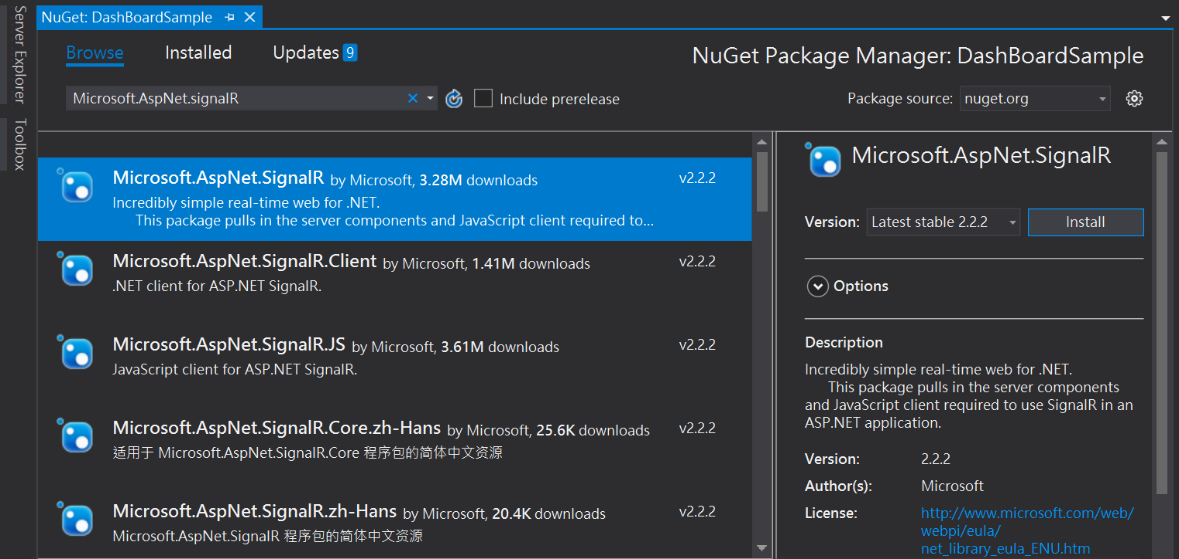


## Step 3: Add the SignalR Library

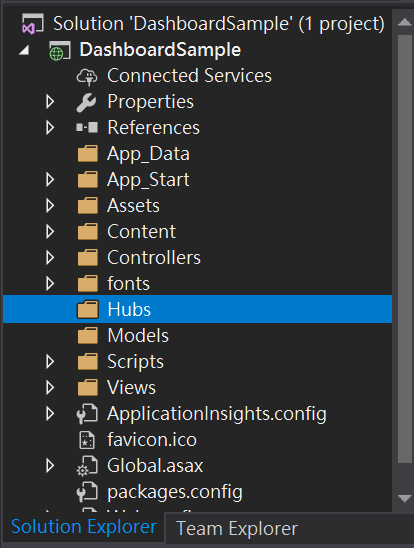
* Open the **Solution Explorer** >> **DashboardSample** >> **Manage NuGet Packages** and get the SignalR library.



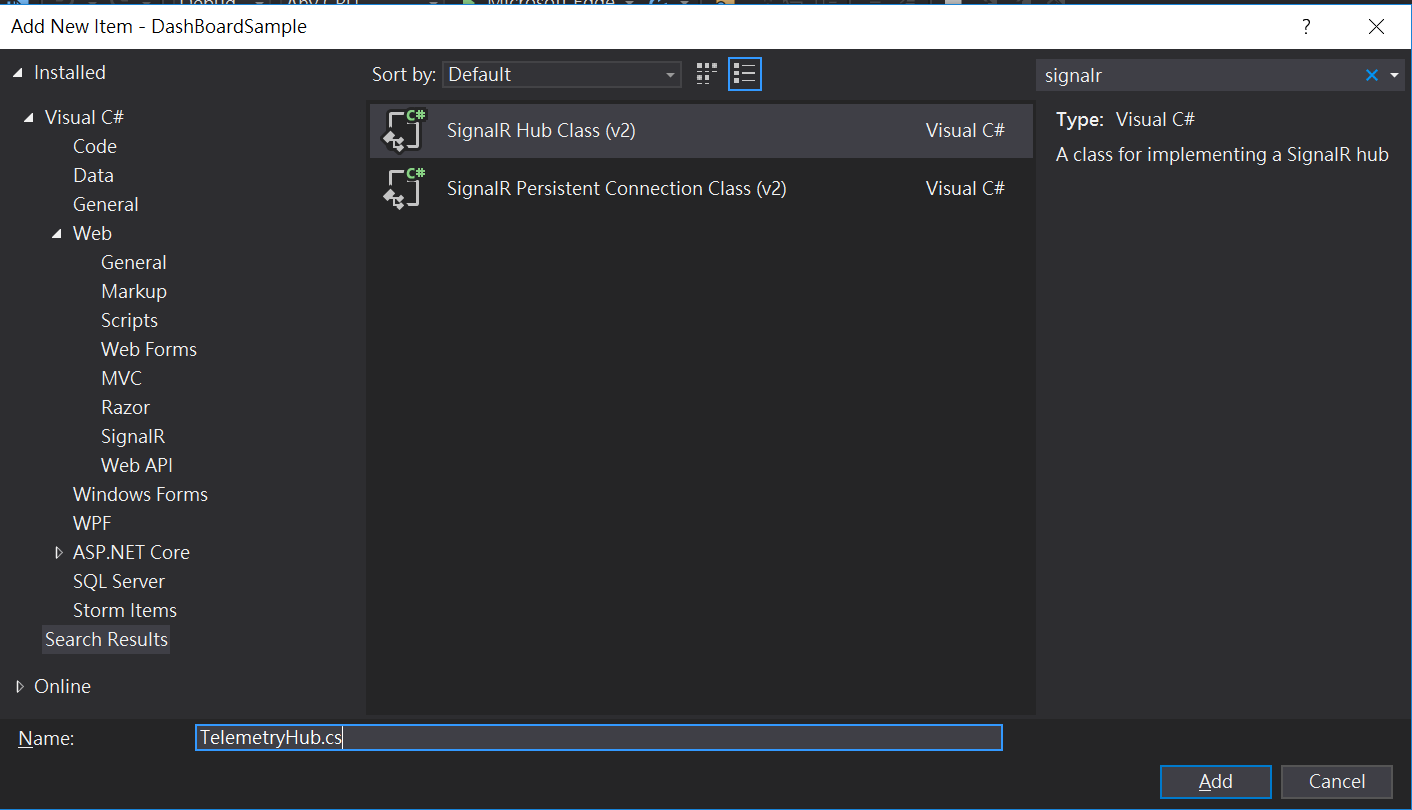
* **Browse** to find **Microsoft.AspNet.signalR**, then install this package with the version 2.2.2.

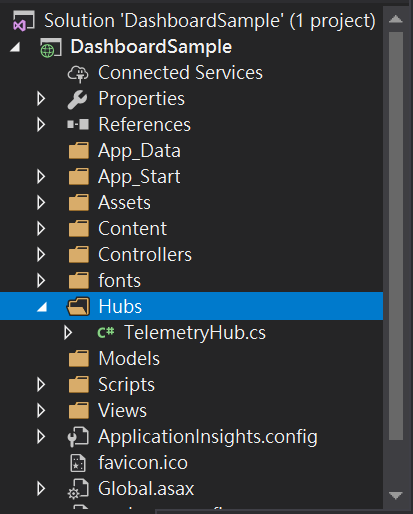


* In **Solution Explorer**, right-click the project **DashboardSample**, select **Add** >> **New Folder**, and add a new folder named **Hubs**.



* Right-click the **Hubs** folder, click **Add** >> **New Item**, select the **Visual C#** >> **Web** >> **SignalR** node in the **Installed** pane, select **SignalR Hub Class (v2)** from the center pane, and create a new hub named **TelemetryHub.cs**. You will use this class as a SignalR server hub that sends messages to all clients.





* Replace the code in the **TelemetryHub** class with the following code.

using Microsoft.AspNet.SignalR;

namespace DashboardSample.Hubs

{

public class TelemetryHub : Hub

{

public void Hello()

{

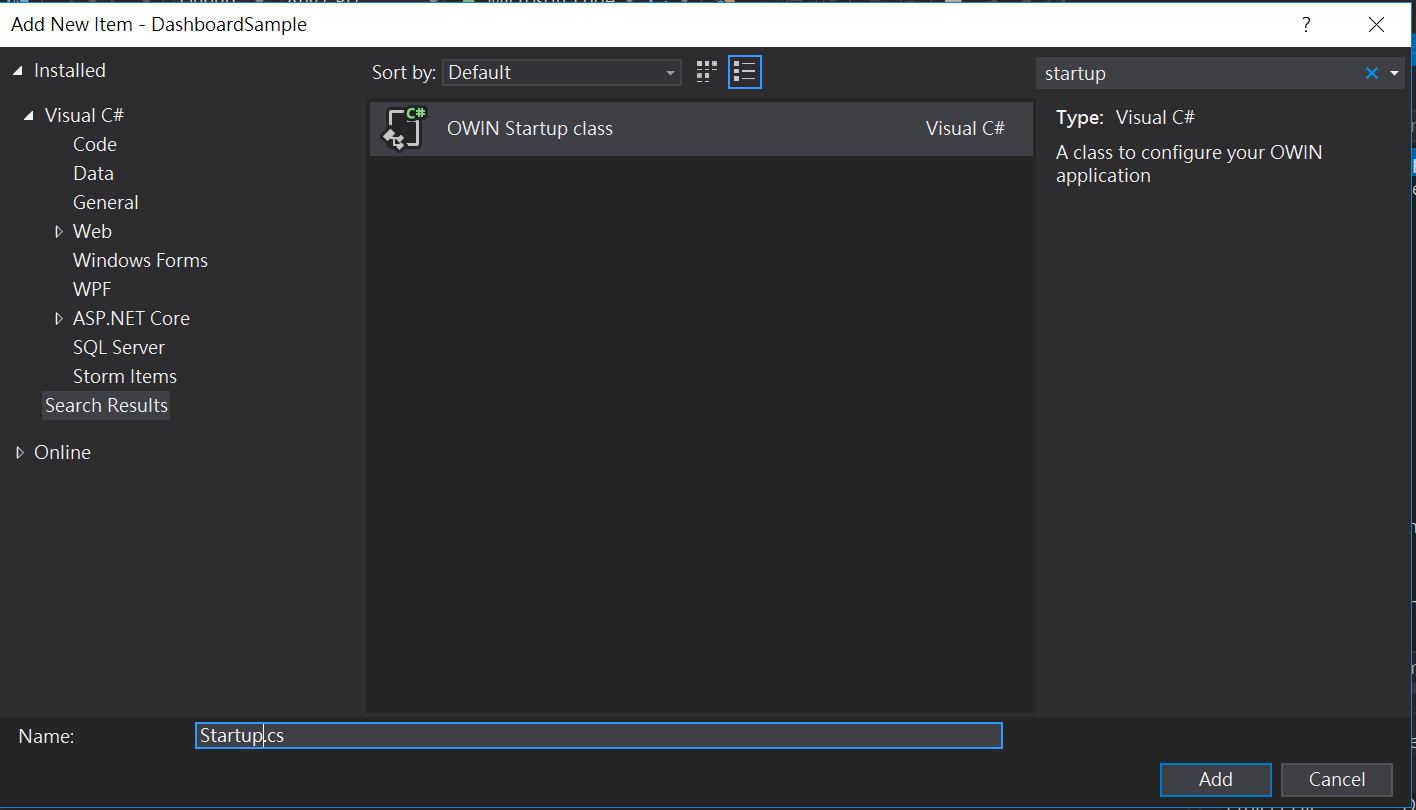
System.Diagnostics.Debug.WriteLine("Hello!");

}

}

}

* Right-click the project **DashboardSample**, click **Add** >> **Class**, search startup, and select **OWIN Startup class** from the center panel, and naming the Startup.cs , click **Add.**



* Update the **Startup.cs** of Root directory. Change the contents of the file to the following.

using System;

using System.Threading.Tasks;

using Microsoft.Owin;

using Owin;

[assembly: OwinStartup(typeof(DashboardSample.Startup))]

namespace DashboardSample

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

// Any connection or hub wire up and configuration should go here

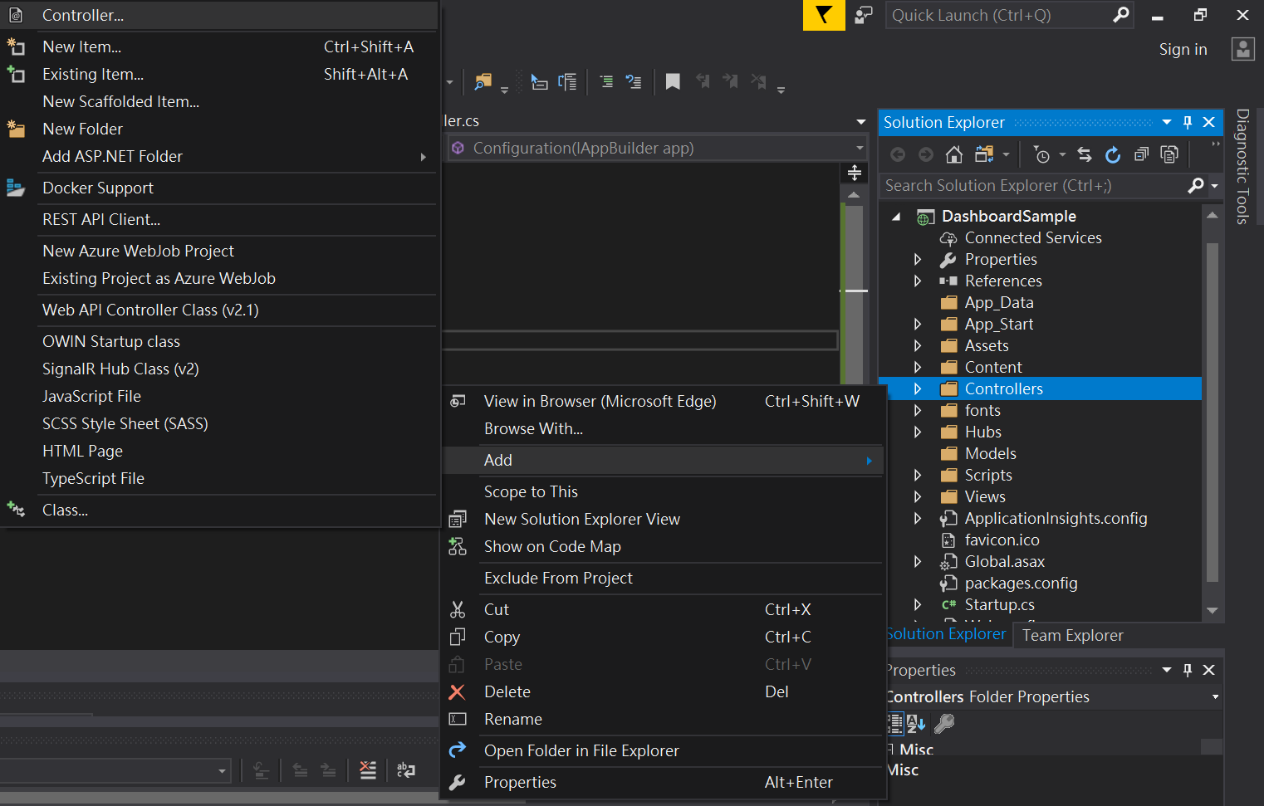
app.MapSignalR();

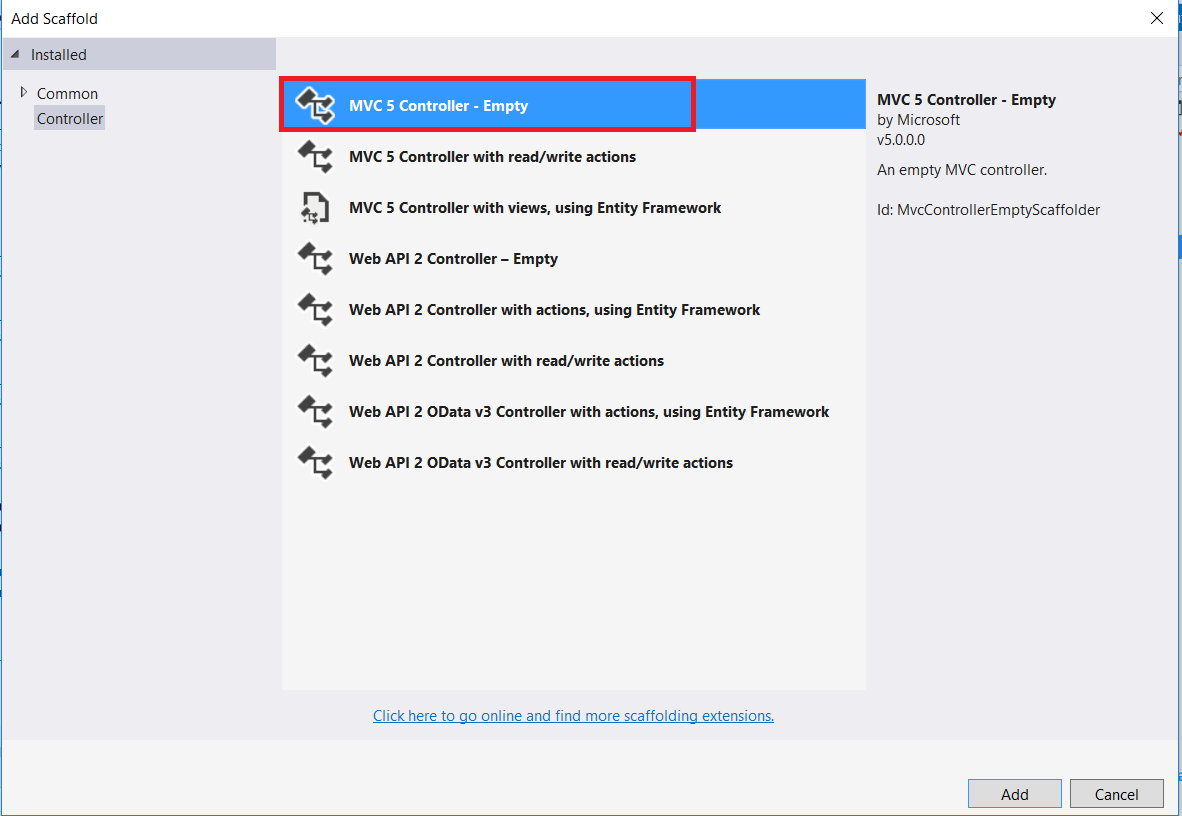
}

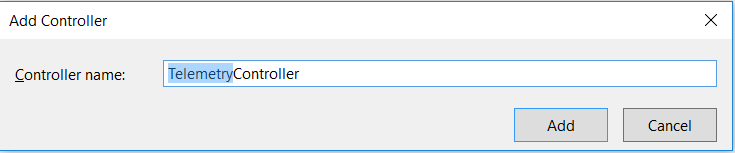
}

}

* Add the **TelemetryController.cs** class in Controllers.







* Update the **TelemetryController.cs**. Change the contents of the file to the following. This method returns the Empty view that you will create in a later step.

using Microsoft.AspNet.SignalR;

using Newtonsoft.Json.Linq;

using DashboardSample.Hubs;

using System.Web.Mvc;

namespace DashboardSample.Controllers

{

public class TelemetryController : Controller

{

// POST Request from Event Processor Host

[HttpPost]

public ActionResult PutTelemetry(string deviceId, string msgId, double temperature, double humidity, string time)

{

System.Diagnostics.Debug.WriteLine("deviceId = {0}, msgId = {1}, temperature = {2}, humidity = {3}, time = {4}",

deviceId,

msgId,

temperature,

humidity,

time);

var context = GlobalHost.ConnectionManager.GetHubContext<TelemetryHub>();

dynamic telemetryObject = new JObject();

telemetryObject.deviceId = deviceId;

telemetryObject.msgId = msgId;

telemetryObject.temperature = temperature;

telemetryObject.humidity = humidity;

telemetryObject.time = time;

context.Clients.All.sendTelemetry(telemetryObject);

return this.Content("");

}

// POST Request from Event Processor Host

[HttpPost]

public ActionResult AlarmTelemetry(string ioTHubDeviceID, string messageID, string alarmType, string reading, double threshold, string localTime, string createdAt)

{

var context = GlobalHost.ConnectionManager.GetHubContext<TelemetryHub>();

dynamic alarmObject = new JObject();

alarmObject.ioTHubDeviceID = ioTHubDeviceID;

alarmObject.alarmType = alarmType;

alarmObject.reading = reading;

alarmObject.threshold = threshold;

alarmObject.createdAt = createdAt;

context.Clients.All.sendAlarmTelemetry(alarmObject);

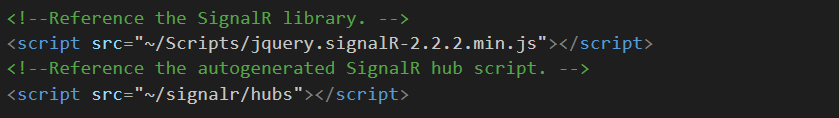
return this.Content("");

}

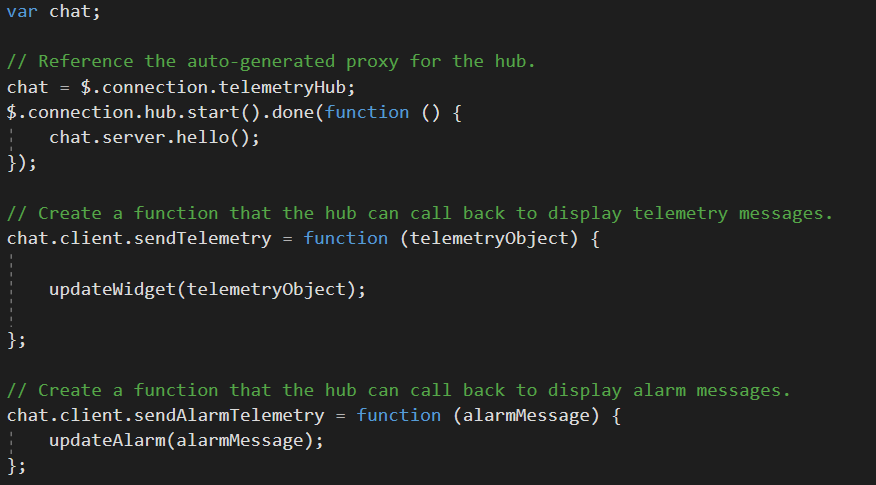
}

}

* See the bottom of View/Shared/**\_Layout.cshtml**, which included the signalR library.

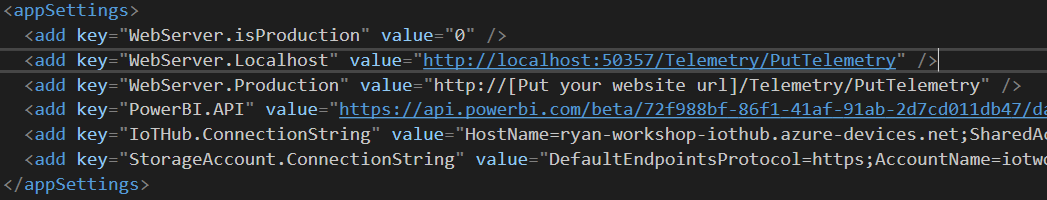


* See the code below in Assets/js/**realtime.js**, define actions after receiving real time data and alert.



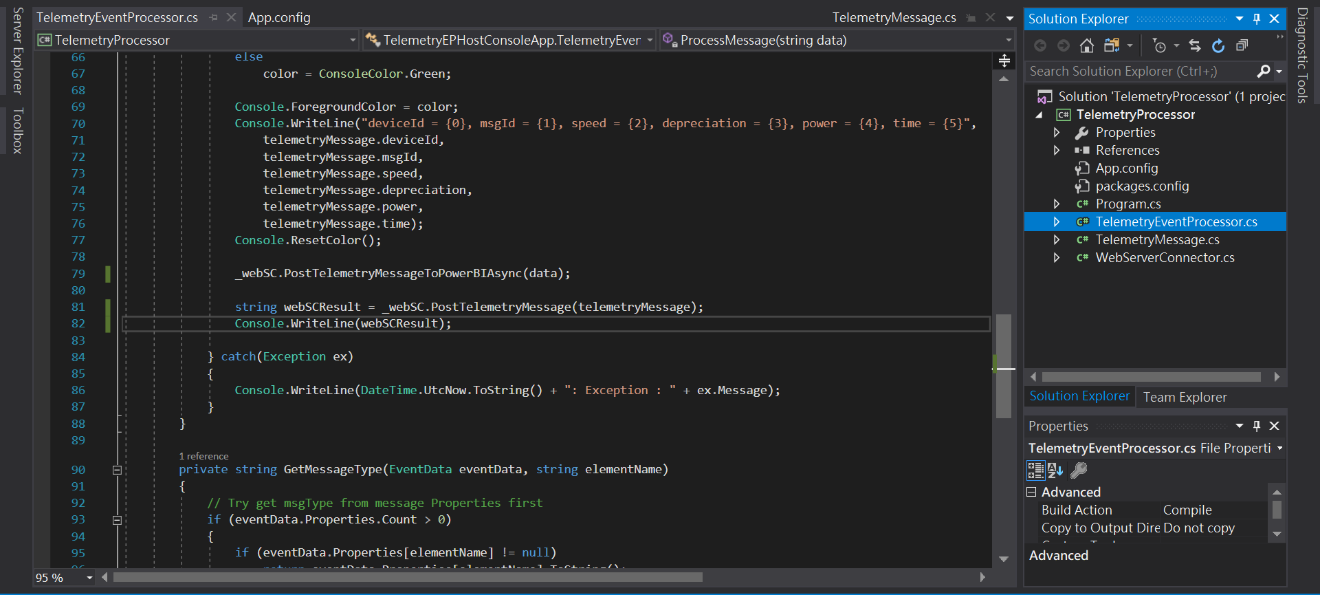
* Build and run the project. For now, it is ready for the real time data. Next, let’s update the Event Processor Host.

## Step 4: Update Event Processor Host to push the telemetry data

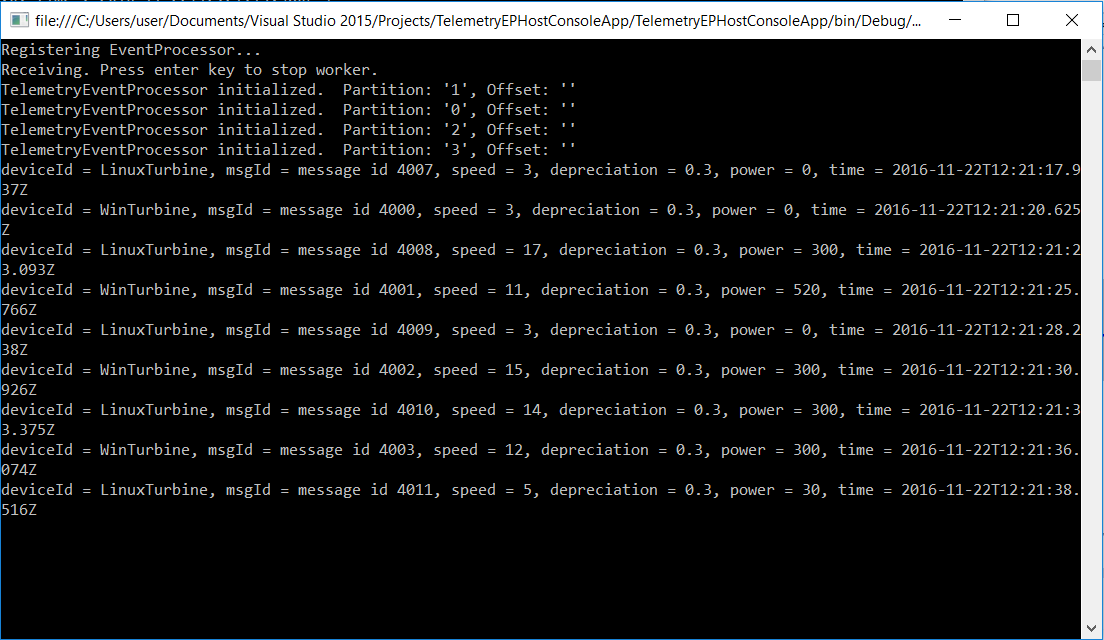
* Add DashboardSample local url with port number to TelemetryProcessor / **App.config  
  (bear in mind the difference of http and https)** 
* Open TelemetryProcessor / **TelemetryEventProcessor.cs**
* Remove comments on

//string webSCResult = \_webSC.PostTelemetryMessage(telemetryMessage);

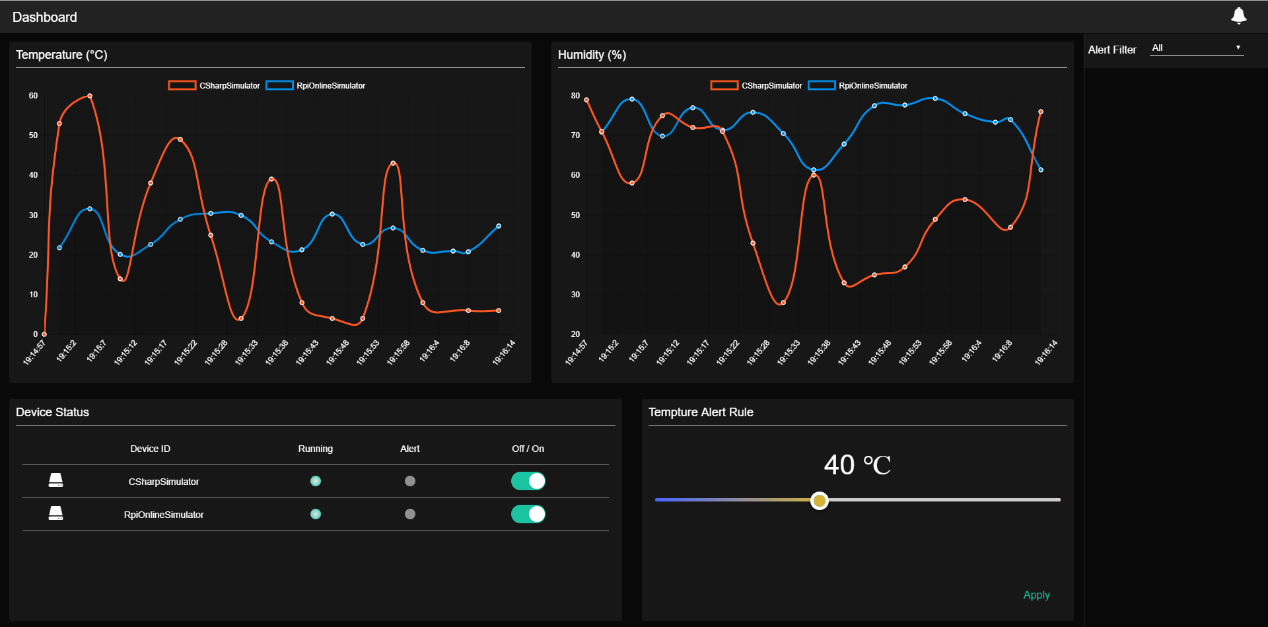
//Console.WriteLine(webSCResult);



* Run the Event Processor Host and check the output of console App

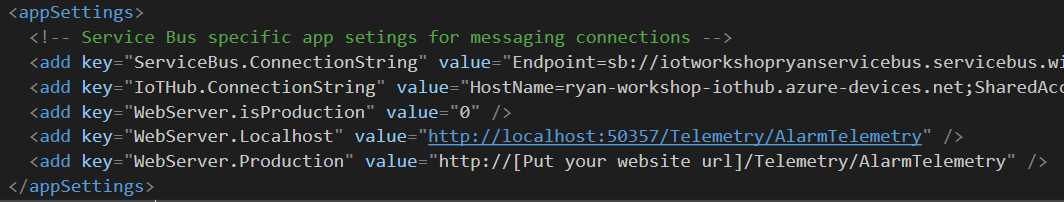


* Watch the real time data on Dashboard Web App.



## Step 5: Update Alert Service Bus to push the alert message

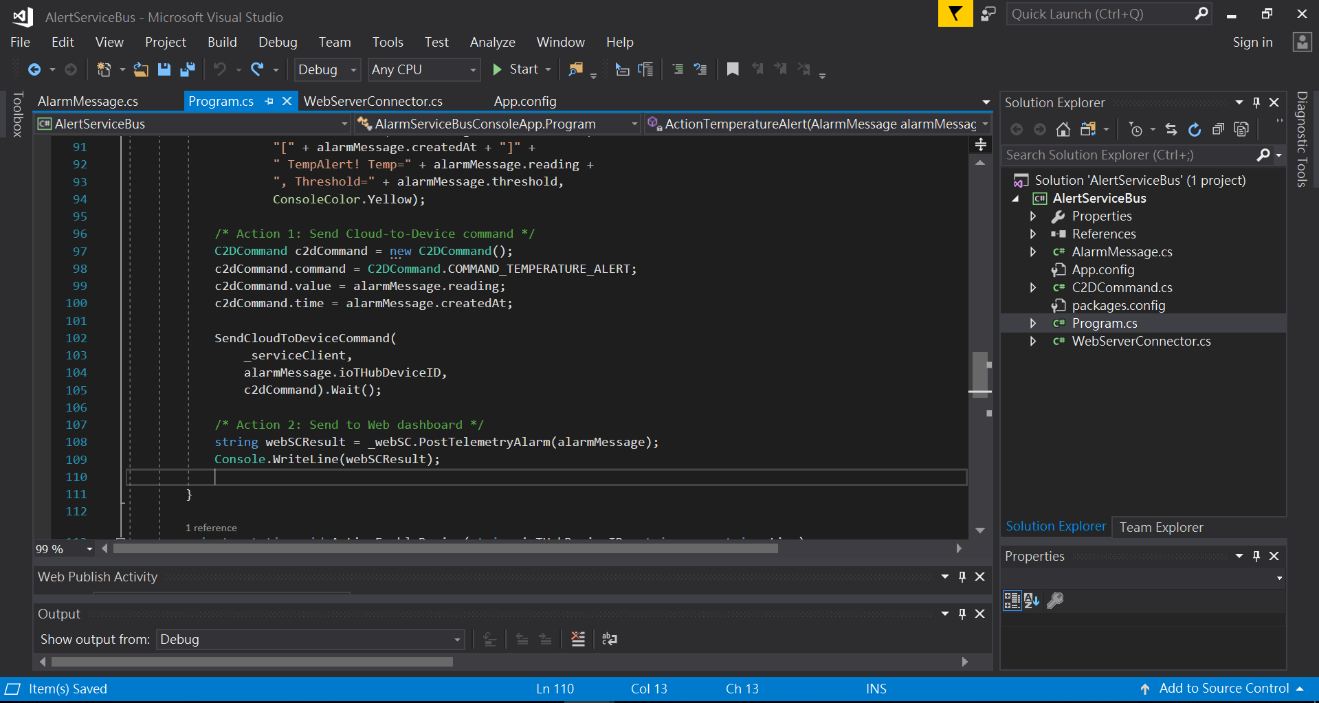
* Add DashboardSample local url with port number to AlertServiceBus / **App.config**



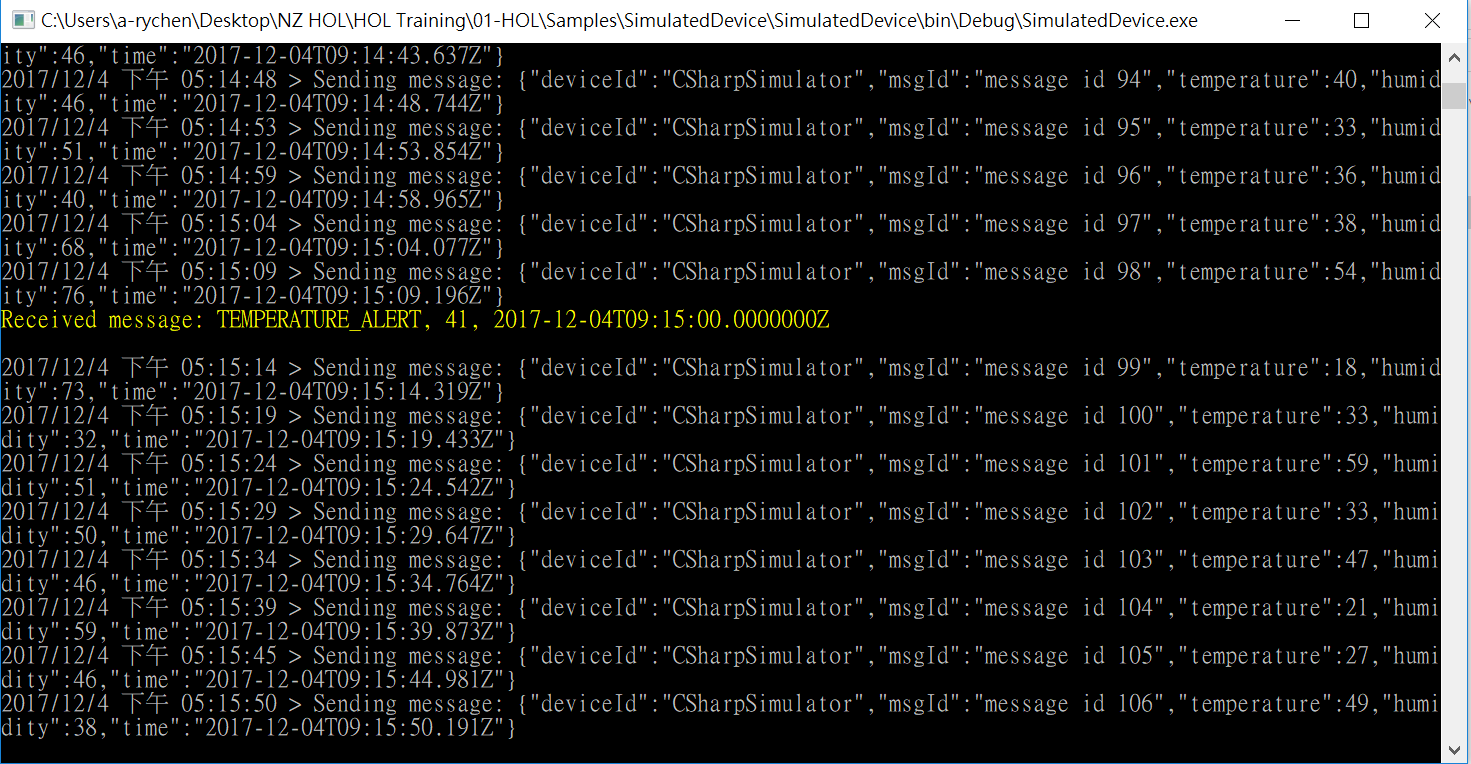
* Open AlertServiceBus >> **Program.cs**
* Remove these 2 comments on ActionTemperatureAlert method.

//string webSCResult = \_webSC.PostTelemetryAlarm(alarmMessage);

//Console.WriteLine(webSCResult);

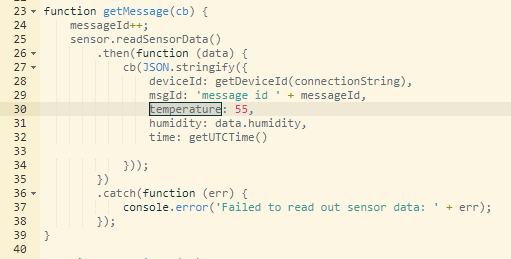


* Run the Alert Service Bus and check the output of console App

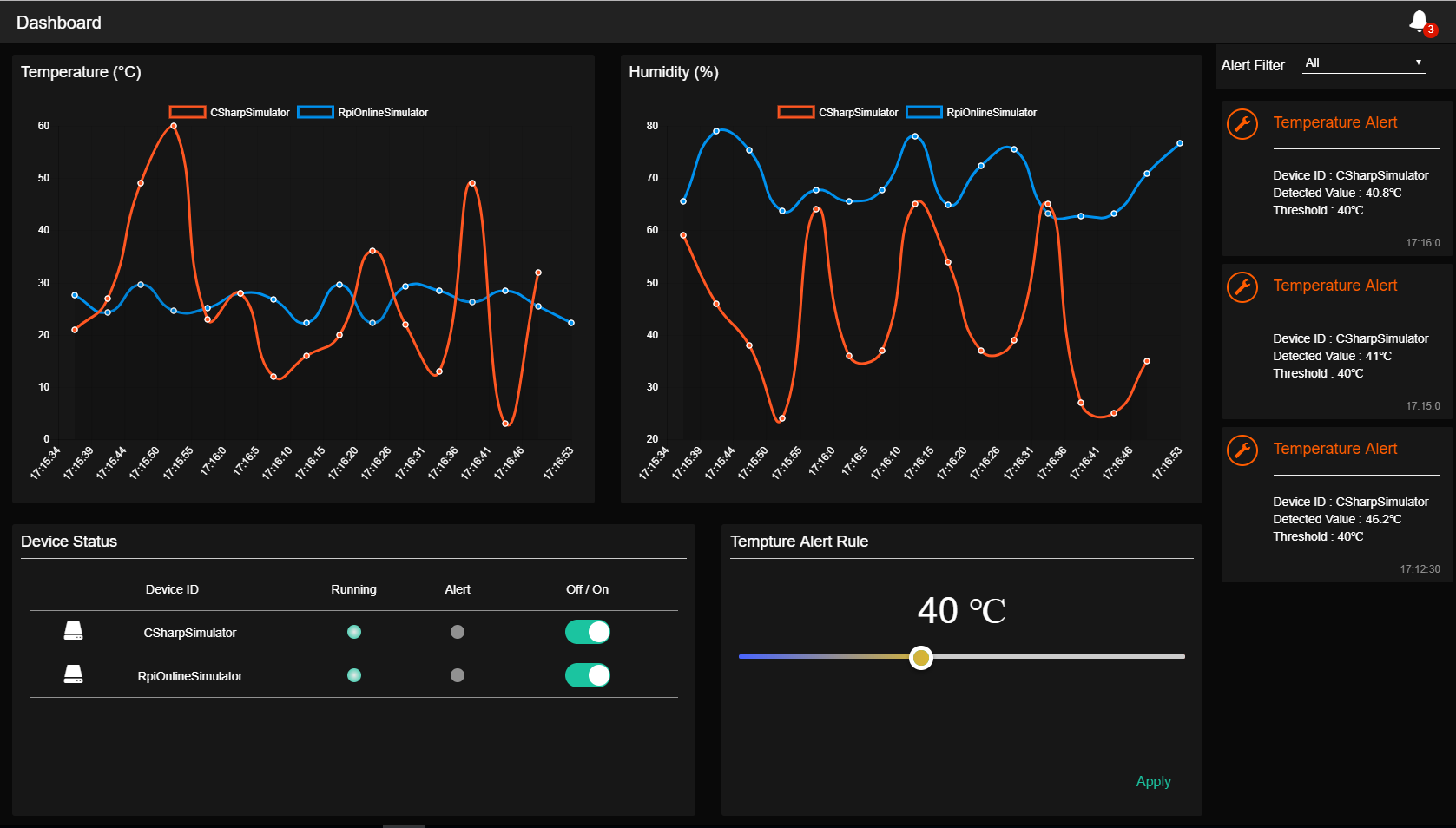
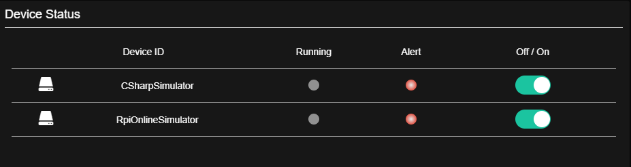


## Step 6: Show the real time data and alert in dashboard

* Before running these devices, you can modify the temperature so that it is greater than the threshold (The default value is 40).
  + Raspberry Pi Azure IoT Online Simulator



* Run the Device simulator.
* Watch the alert message in the dashboard web app.



* *The HOL 5 has been completed. Now we create a customized web app, and it can read the telemetry data from IoT Hub through the Event Processor Host, and feed-in the data to the backend web server by Http API. Finally, we use SignalR technology to communicate between web server and browser to get the real-time dashboard.*