knowledge16
May 15–20, 2016 • Mandalay Bay • Las Vegas, NV

Lab Guide

Creating Custom Applications from Scratch 301

Cory Seering & Chris Maloy

Default Login / Password:

admin / Knowledge16

itil / Knowledge16

employee / Knowledge16

This
Page
Intentionally
Left
Blank

Lab Goal

Lab 1.1
Setup

On your marks, get set up, GO! The goal of this lab is to make sure you are set up properly to complete the subsequent labs.

Required: You must have access to **GitHub** and can **fork a public project** from your logged in account**.**

We want to make sure that you:

- Have proper access to your assigned instance.
- Can find and open Studio.
- Can import an application already built in the Git repo.

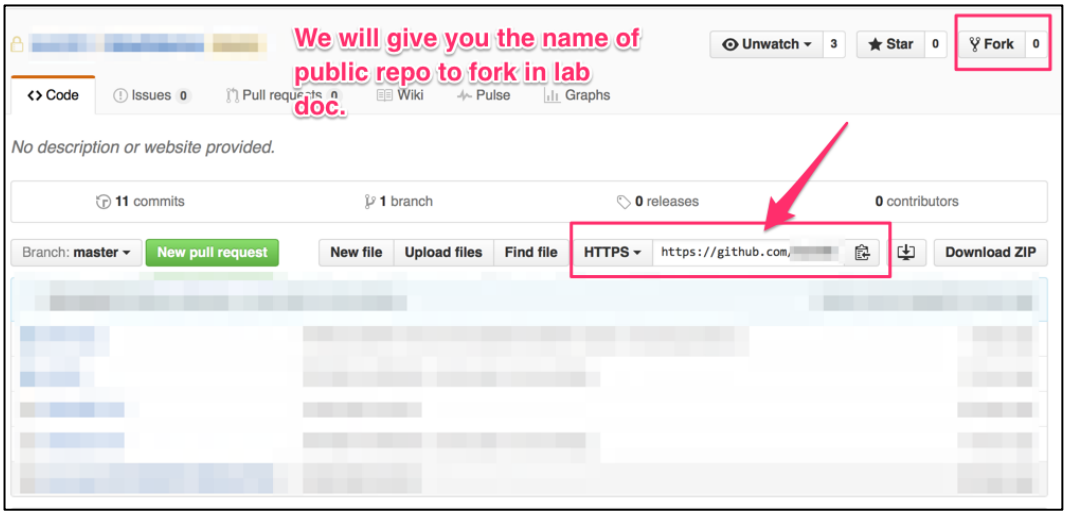
Log in to Your Provided Instance

1. Navigate to the unique instance URL provided to you.

2. Log on with the provided credentials.

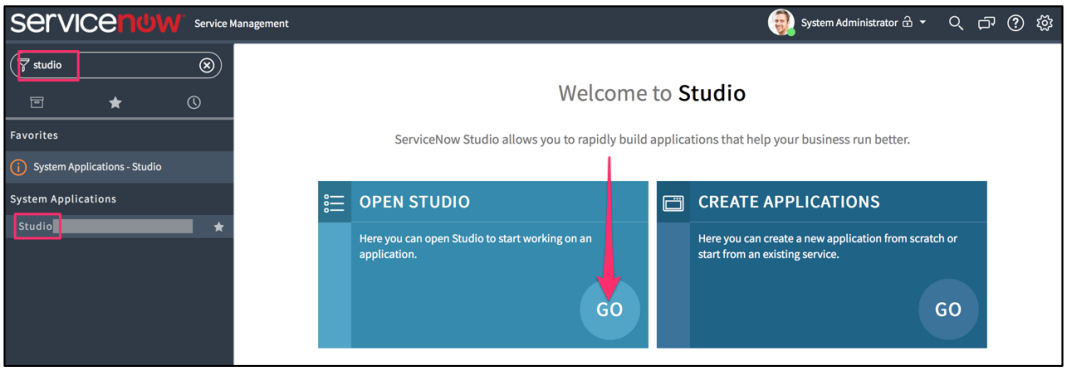
Fork the Git Repo, Open Studio, Link to Git Repo App

1. In this lab, you log into GitHub and find the K16-301-GameTimeNow repo and fork it into your own account. You need the project URL to configure ServiceNow.

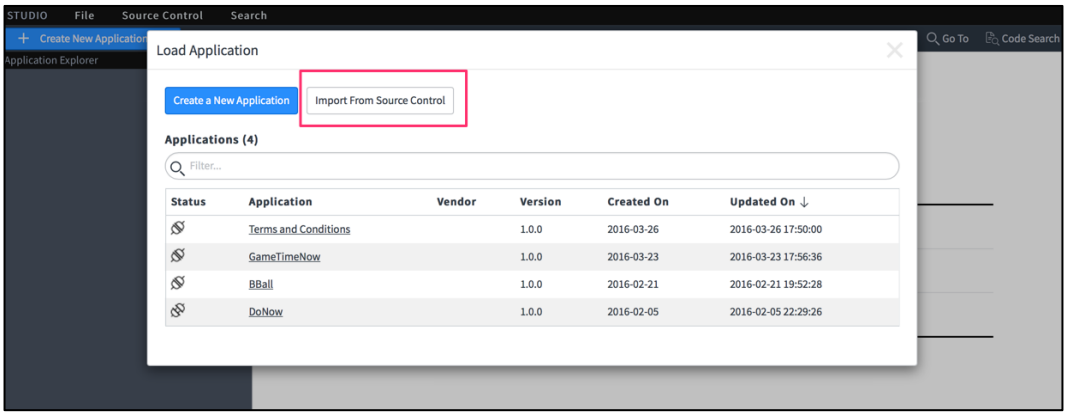


©2016 ServiceNow, Inc. All rights reserved.

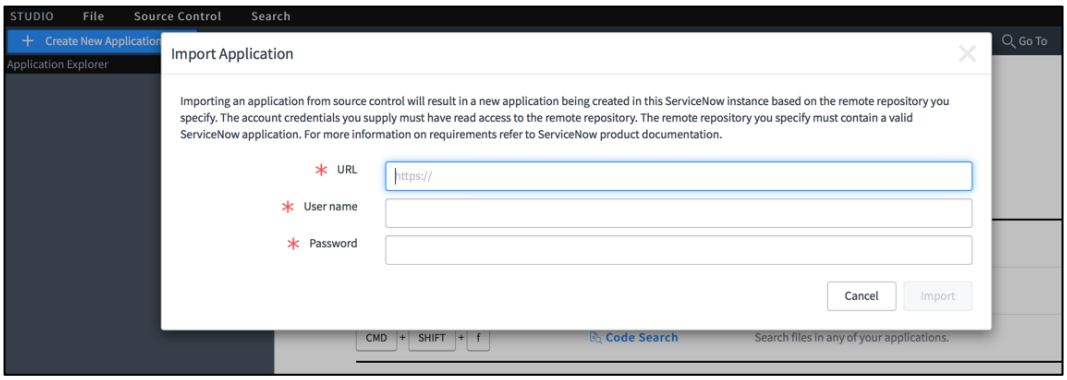
2. Navigate to and open **Studio**.



3. You are prompted to link to source control or open an existing application. Select **Import from source control**.



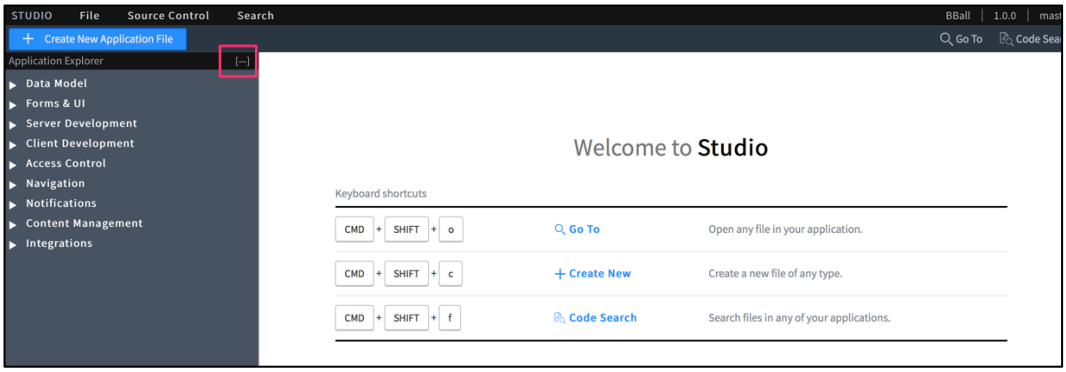
4. Fill in the https project URL you copied from GitHub from the earlier step. Use your username and password.



©2016 ServiceNow, Inc. All rights reserved.

Lab Success Verification

1. In the application explorer you should see the lab as it has been done up to the termination of the 201 course, with some additional content we added for this lab. Starting from here we customize the UI.



®2016 ServiceNow, Inc. All rights reserved.

Lab Goal

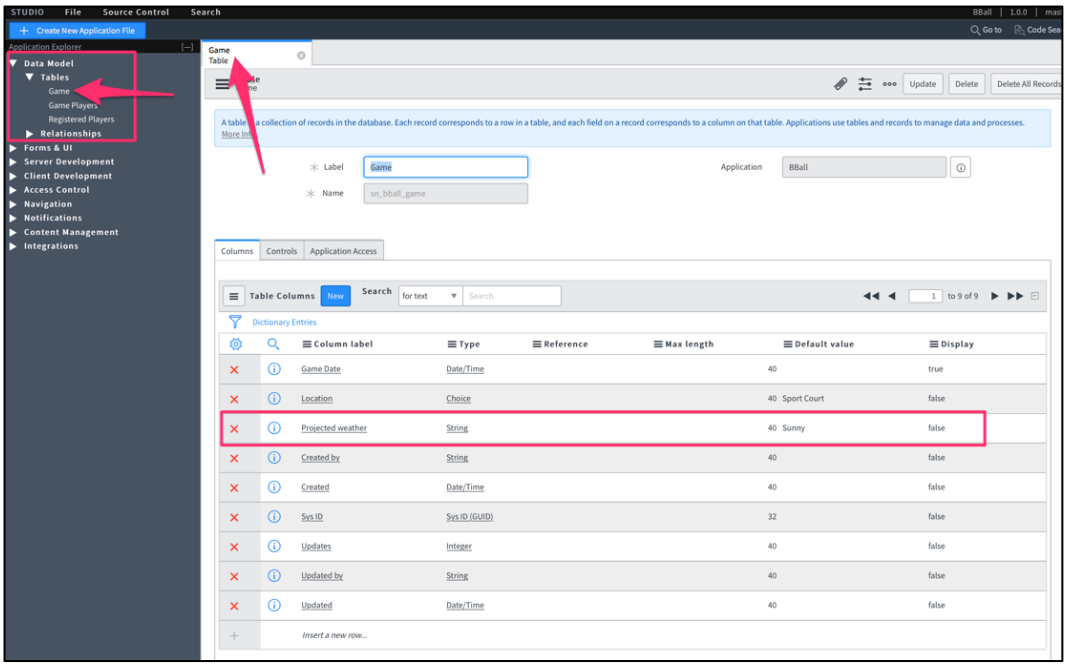
Lab 1.2
Change the Data Model

This lab explains how to make a small update to the data model to include the weather. We try to expose more advanced topics in application development during these labs to give you an understanding of the building tools you can use.

We get the weather for a particular day in the next lab. It is a simple integration and this new field on our game table is where we store that data.

Update the Game Table

1. With Studio open, go into the **Data Model** section of the application explorer and open the **Game** table.



2. In the column tab insert a new row called **Column label – Projected Weather**, **Type – String**, **Default value – Sunny**.

3. Click **Update** on the table.

®2016 ServiceNow, Inc. All rights reserved.

Lab Goal

Lab 2.1 REST Message

REST is an ideal way to communicate with the server or other web service type systems. ServiceNow can act both as the client or server depending on the initiator of a particular web service call.

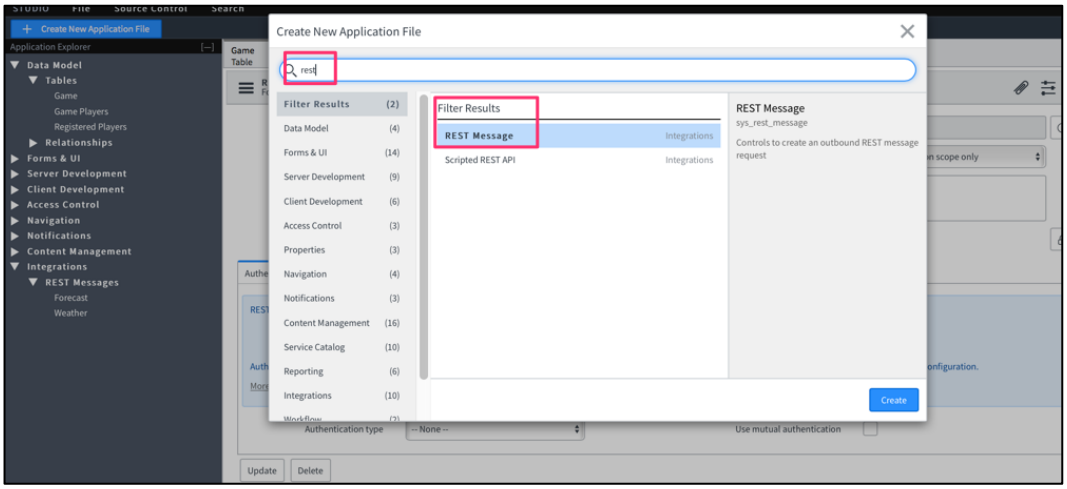
All of the tables in ServiceNow come ready with basic table APIs. We will use these later when building Angular code.

This lab defines a REST Message that communicates to an open source weather API. It uses this data to update the projected weather field, just created, on the Game table.

There are two REST messages used in the final application. You build one of them; the other is already built in this branch of starting code.

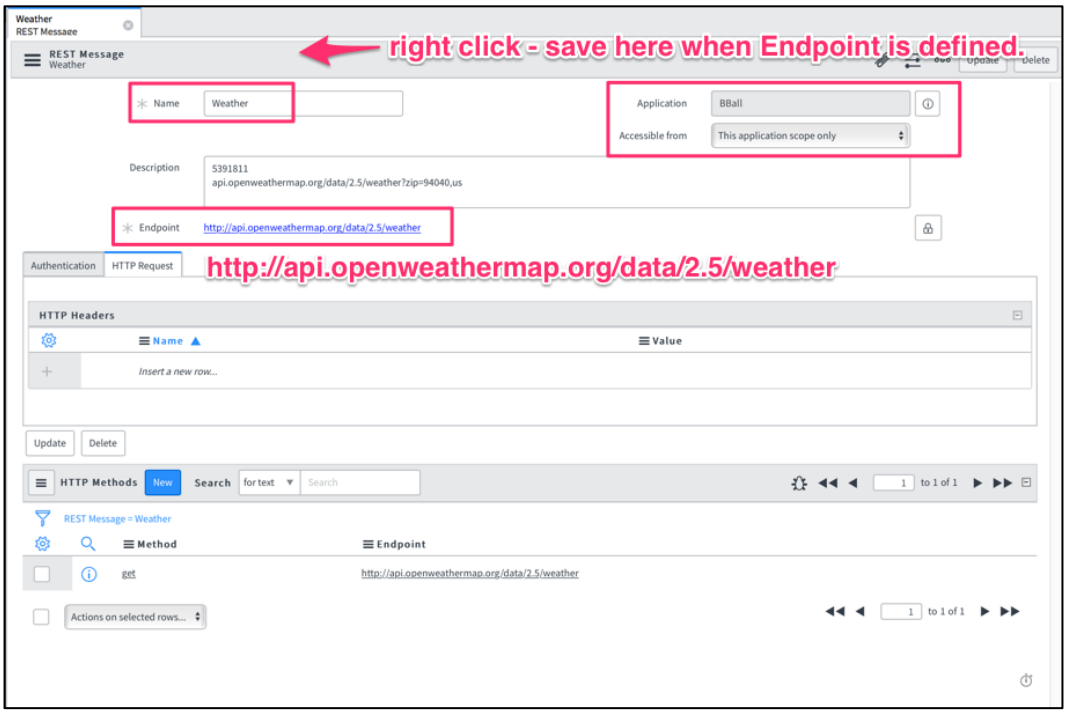
Create a REST Message

1. Create a new **application file** in Studio of type REST Message.



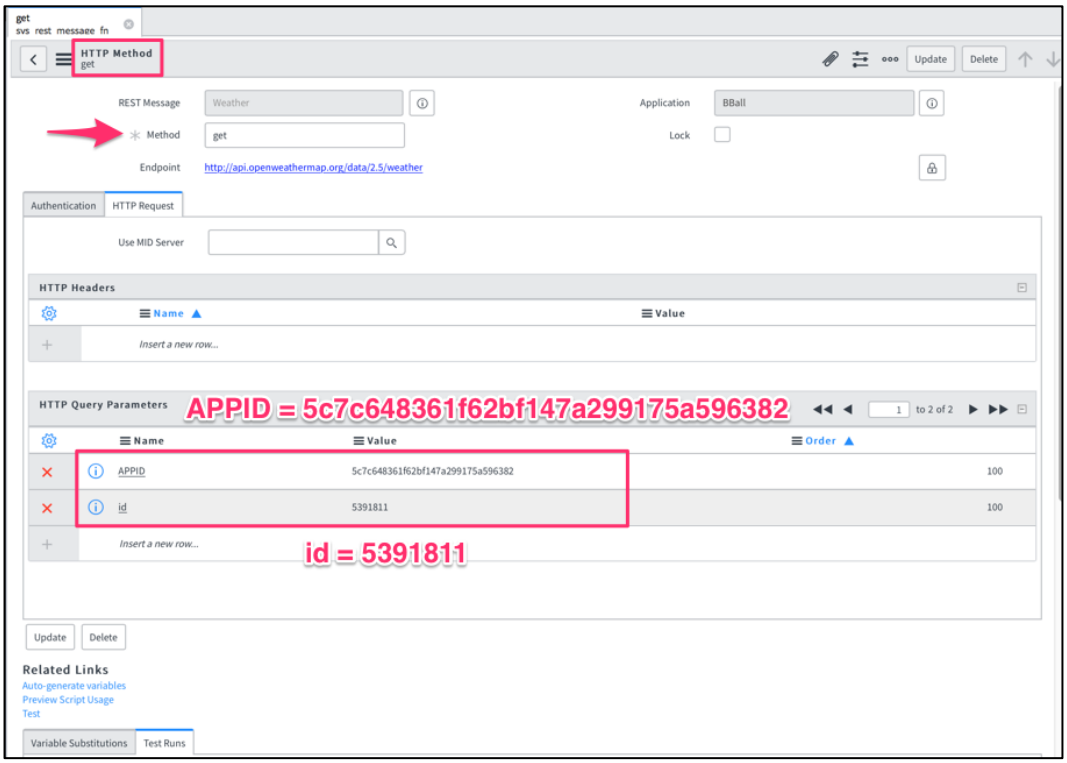
©2016 ServiceNow, Inc. All rights reserved.

2. Give this file a name of **Weather** and the endpoint as described in this image:



®2016 ServiceNow, Inc. All rights reserved.

3. Right click and **save** the file. This auto generates the HTTP methods. Open the **get** method.



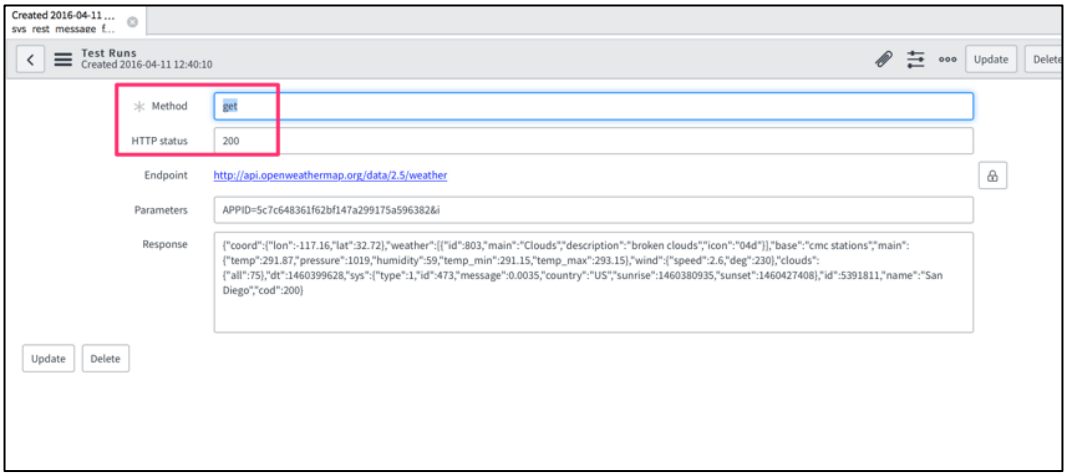
4. Add two HTTP Query Parameters: **APPID = 5c7c648361f62bf147a299175a596382** and **id = 5391811**.

5. Right click and **save** from the get method form.

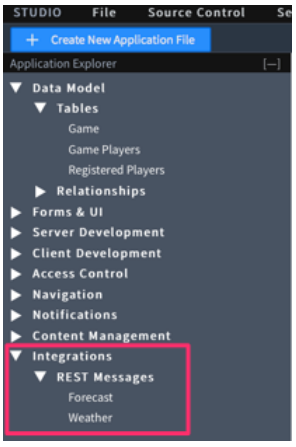
®2016 ServiceNow, Inc. All rights reserved.

Lab Success Verification

1. From the get method form click the **Test** related link. You should get back a **200** response if you did this REST message correctly.



2. There should be two REST Messages defined in the Studio application explorer at this point (one for forecast weather and one for current weather).



©2016 ServiceNow, Inc. All rights reserved.

Lab Goal

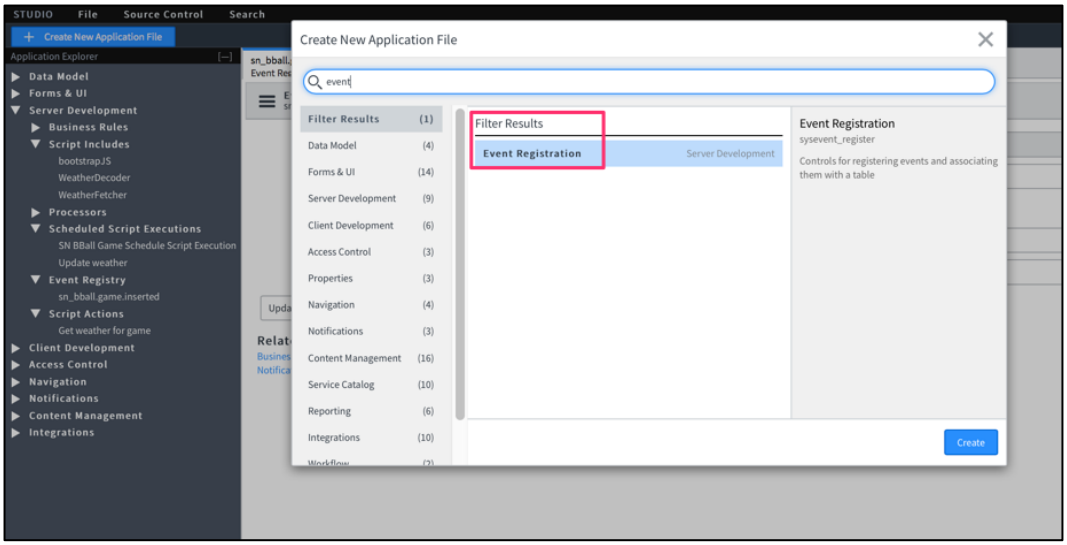
Lab 3.1 Event Handling

In our efforts to show you more and more of the platform from an advanced point of view, we think it is important to realize the ServiceNow event handling capabilities of the platform.

This lab (building off what we just accomplished) explains how to register and fire an event when a new Game is created. This event is responsible for getting the projected weather for that game and updating the game table record with that weather information.

Create and Handle an Event Within the ServiceNow Platform

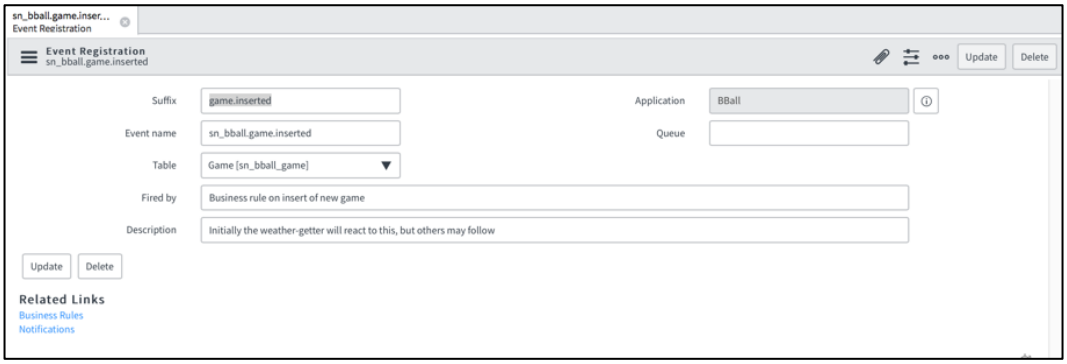
1. Create a new **application file** in Studio for Event Registration.



®2016 ServiceNow, Inc. All rights reserved.

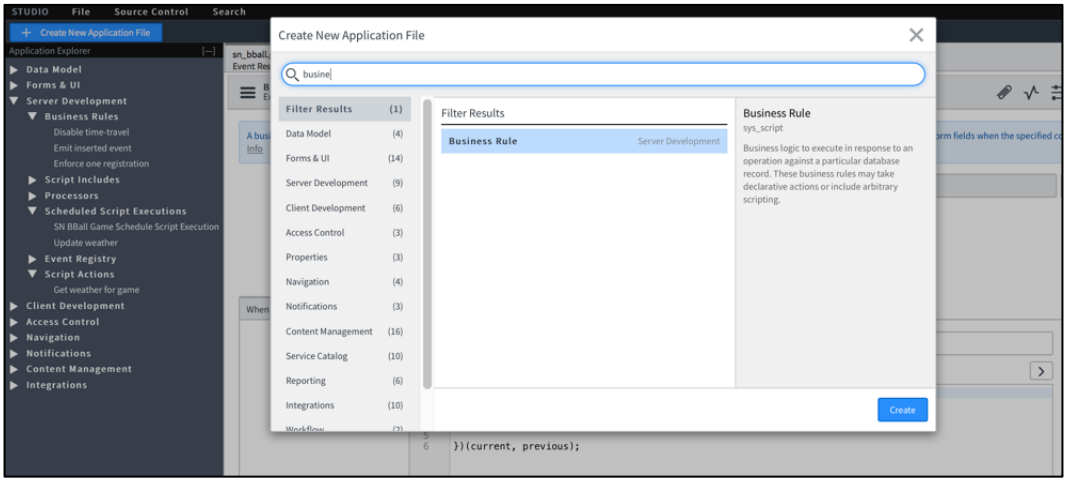
2. Be careful filling out this form. This is the number one place people get into trouble when defining an event. The fields are free text and the spelling matters.

   Suffix: **game.inserted**
   Event name: **sn_bball.game.inserted**
   Table: **Game [sn_bball_game]**
   Fired by: **Business rule on insert of new game**
   Description: **Initially the weather-getter will react to this, but others may follow**



3. **Update/Save**.

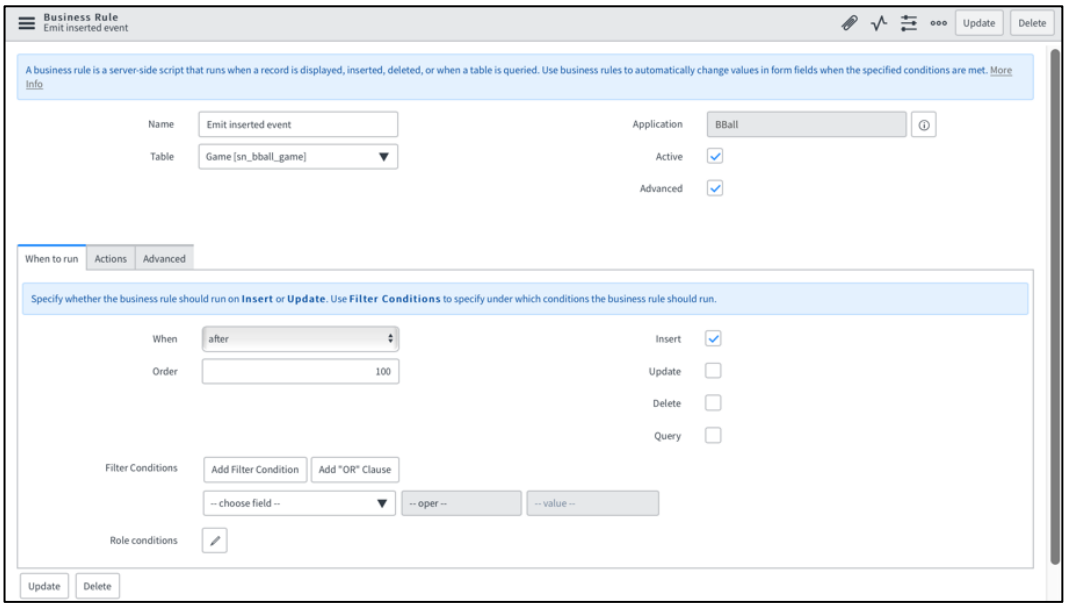
4. Create a new **business rule application file**.



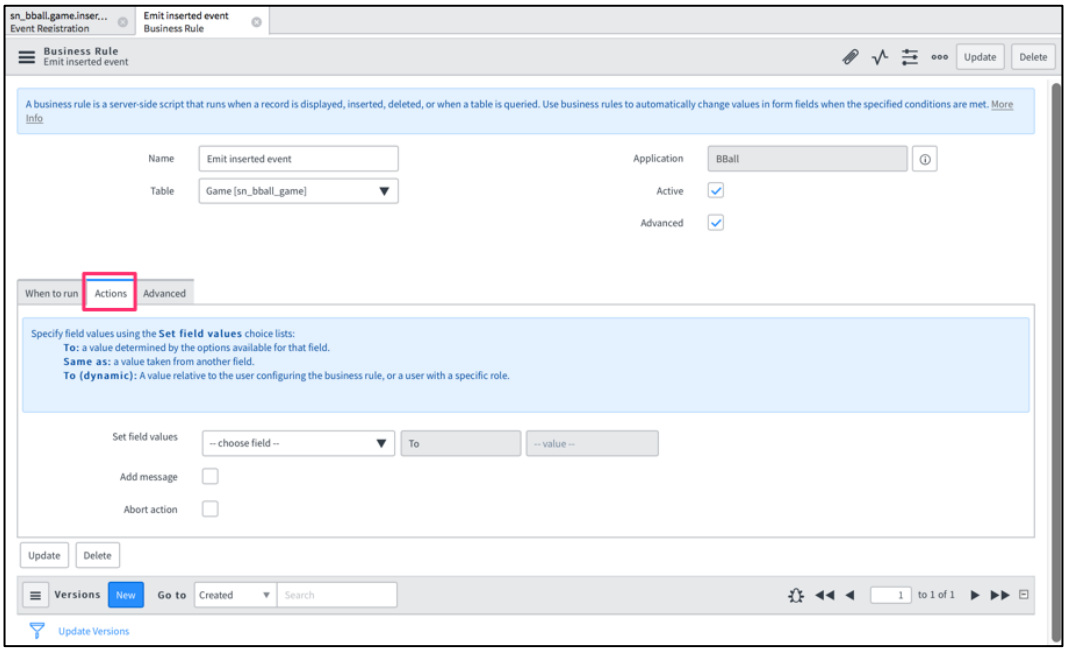
®2016 ServiceNow, Inc. All rights reserved.

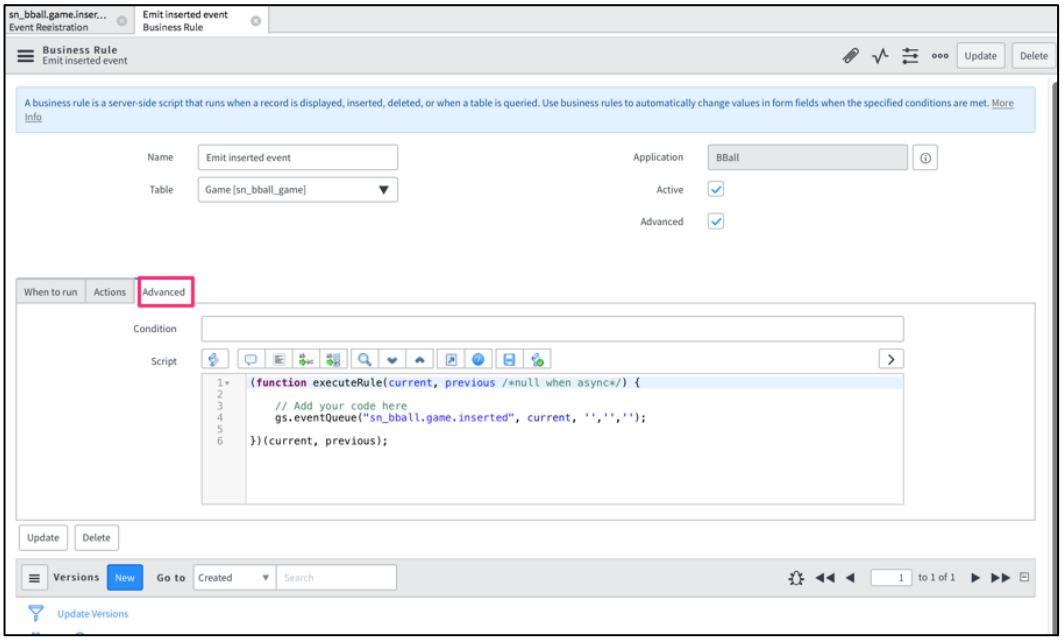
5. The next three images show how to fill out the business rule:

   Name: **Emit inserted event**
   Active: **true**
   Advanced: **true**
   Table: **Game [sn_bball_game]**
   When to run -> When: **after**
   When to run -> Insert: **true**
   Advanced -> Script: **gs.eventQueue("sn_bball.game.inserted", current, '','');**



®2016 ServiceNow, Inc. All rights reserved.

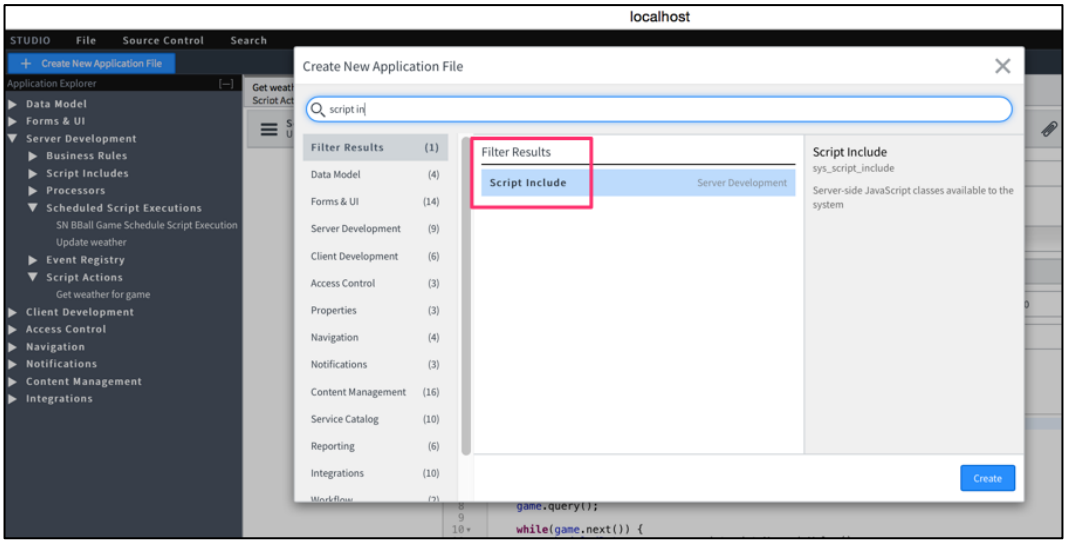




®2016 ServiceNow, Inc. All rights reserved.

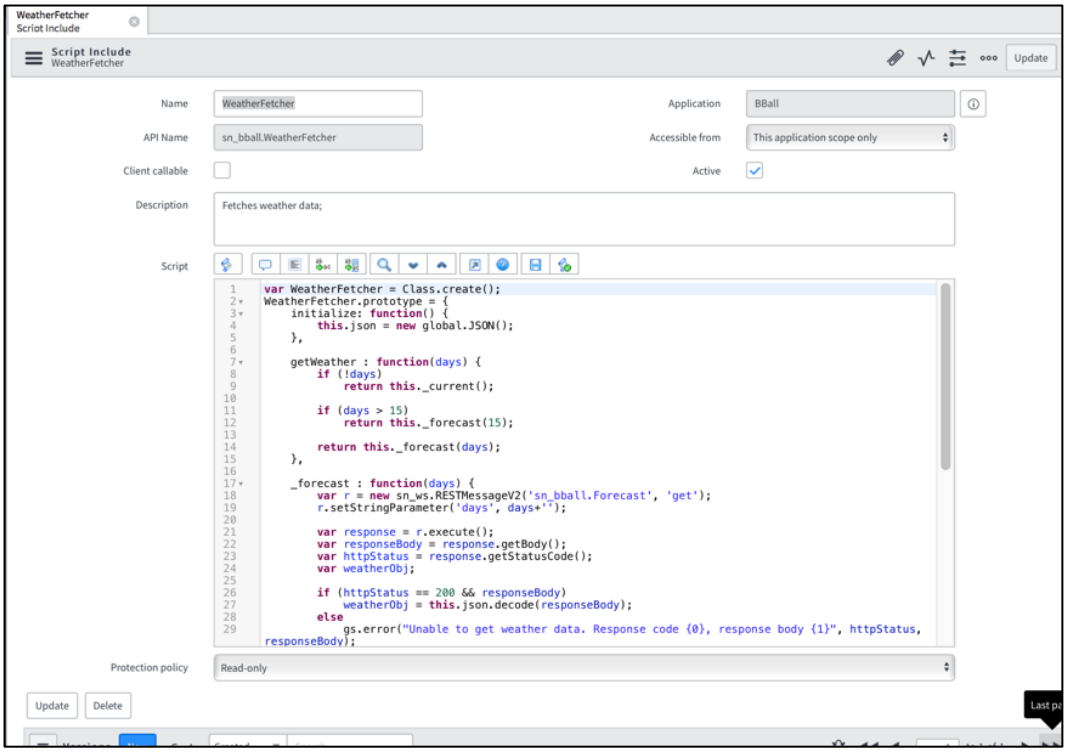
6. Now create a script include that calls the REST Message we defined earlier. You do not have to type all the script here. We just want to show how the concept of events and shared libraries work in ServiceNow from the server side. There is a file within your current branched application you are linked to that you use to can copy/paste.

   Create a new **file** for script include.



©2016 ServiceNow, Inc. All rights reserved.

7. Name the script include **WeatherFetcher**. Notice that the script body gives you a starting point. **Copy and paste the code given to you and place it within the Script Include.** The script is responsible for calling the open weather API (either for a forecast weather report or current weather report). This is how we would use the previous defined REST Message in scripting.



8. Now create a Script Action to be executed whenever the event is fired. Create a new **script action application file**.

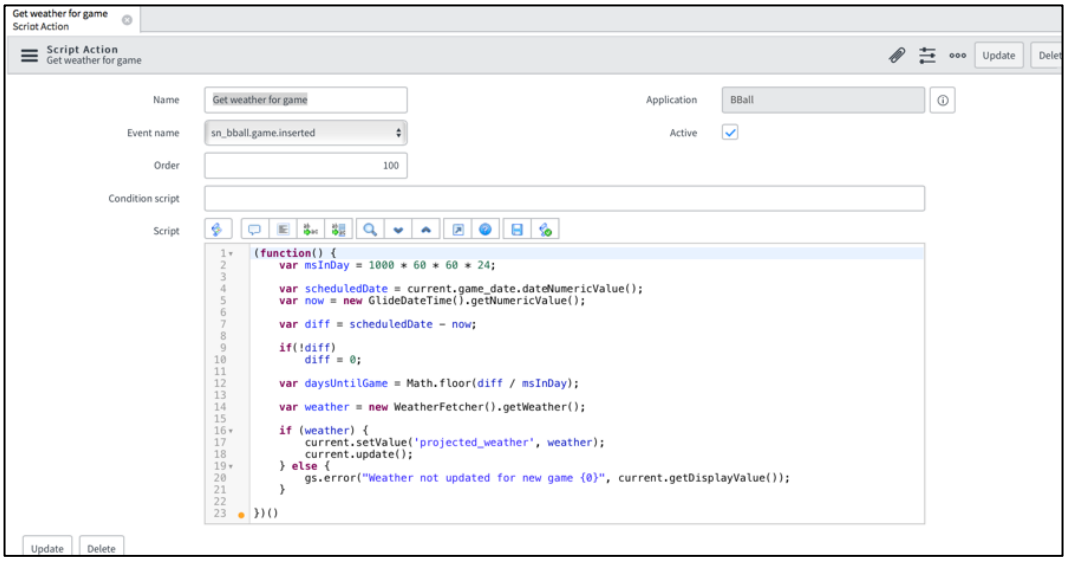
©2016 ServiceNow, Inc. All rights reserved.

9. Name the action and choose the event name defined earlier:

   Name: **Get weather for game**
   Event name: **sn_bball_game.inserted**
   Active: **true**
   Script: **Copy from provided script**

   These scripts have clever ways of dealing with dates you can use.

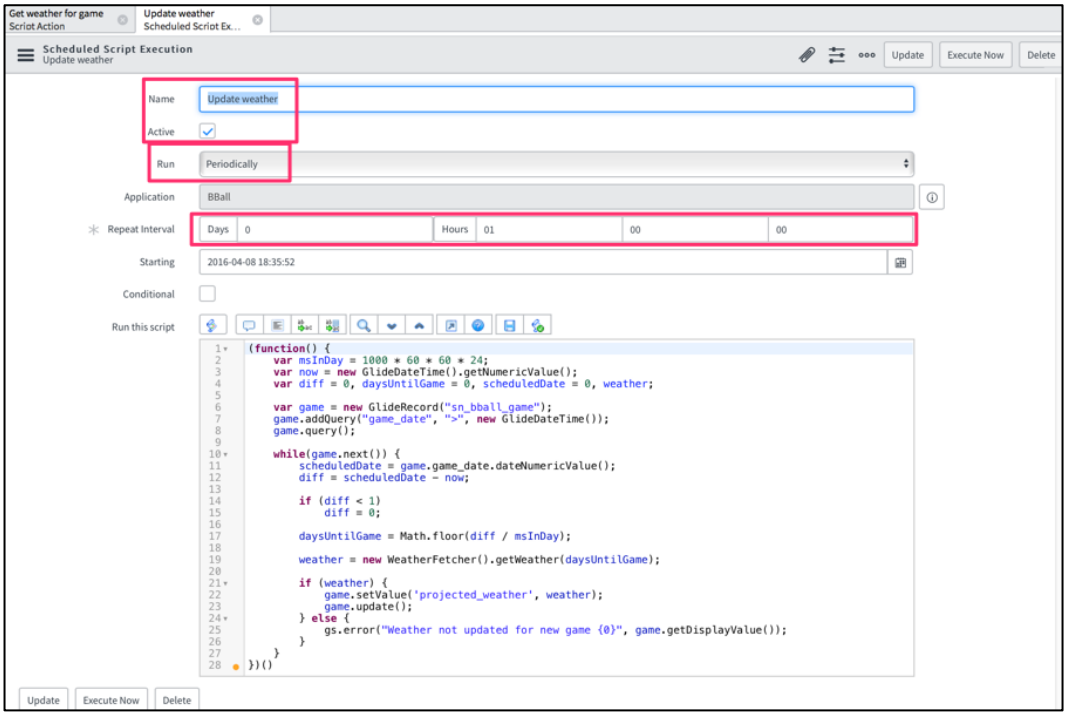
   Pro Tip: Using this event-driven architecture is a nice way for scoped applications to talk to other apps restricted by their own scope.



©2016 ServiceNow, Inc. All rights reserved.

10. Finally, create a **scheduled script** that runs once an hour, checking for current weather conditions for the Game. Create a scheduled script **execution application file**.

    Name: **Update weather**
    Active: **true**
    Run: **Periodically**
    Repeat Interval: **Hours 1**
    Script to run: **Provided by your instructors to copy. Notice what the script is doing**.



Lab Success Verification

1. Execute Now runs the script and updates the weather.

2. Insert new game and get the projected weather.

©2016 ServiceNow, Inc. All rights reserved.

Lab Goal

Lab 4.1 Custom UI with AngularJS

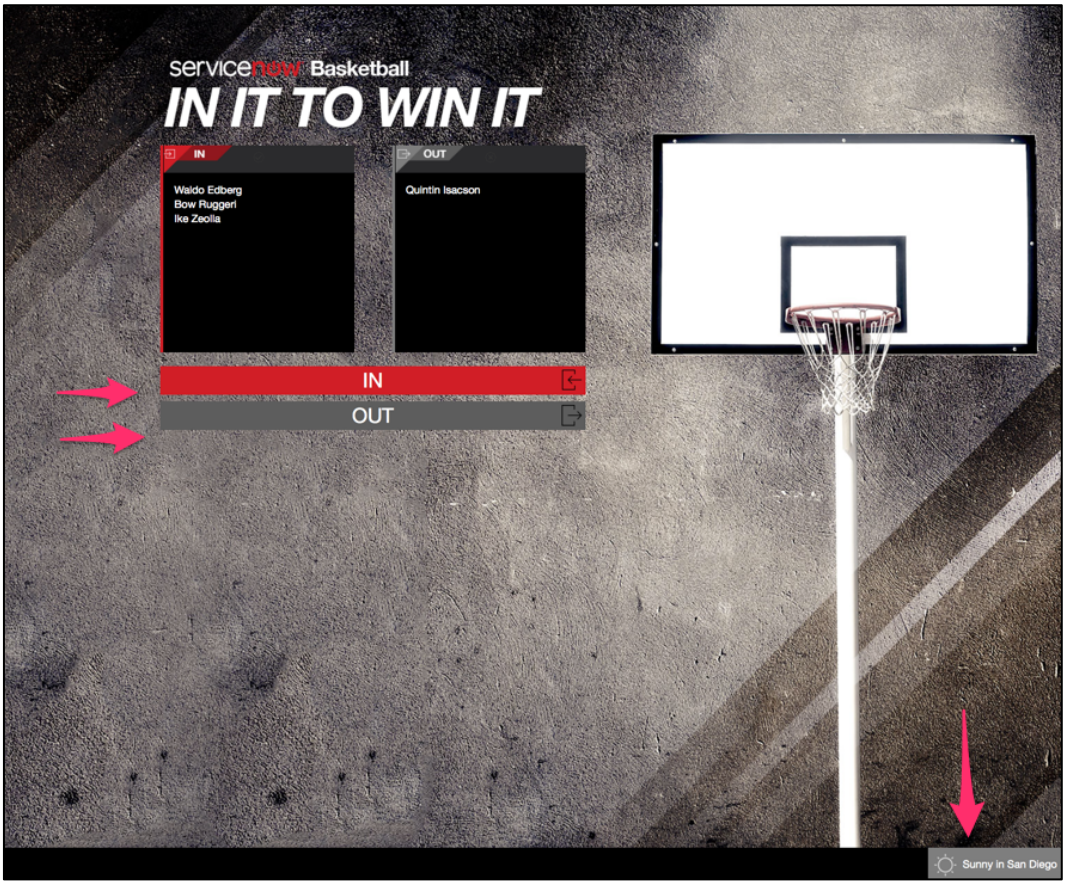
This lab explains how to use AngularJS in your ServiceNow instance and how to build out a custom UI.

"What is all the hoopla all about?" you ask – well, the code to do a lot of really cool browser (mobile friendly) stuff is already built within the Angular libraries and framework. We want to take advantage of that and turn the heat up and make our application look more modern.

This lab explains how to load AngularJS libraries into ServiceNow.

This lab explains how to abstract your own angular code into UI Scripts and shows you the development process to continually build new features for your custom UI.

Here is the Custom UI you build for the basketball games. It is a completely interactive page.

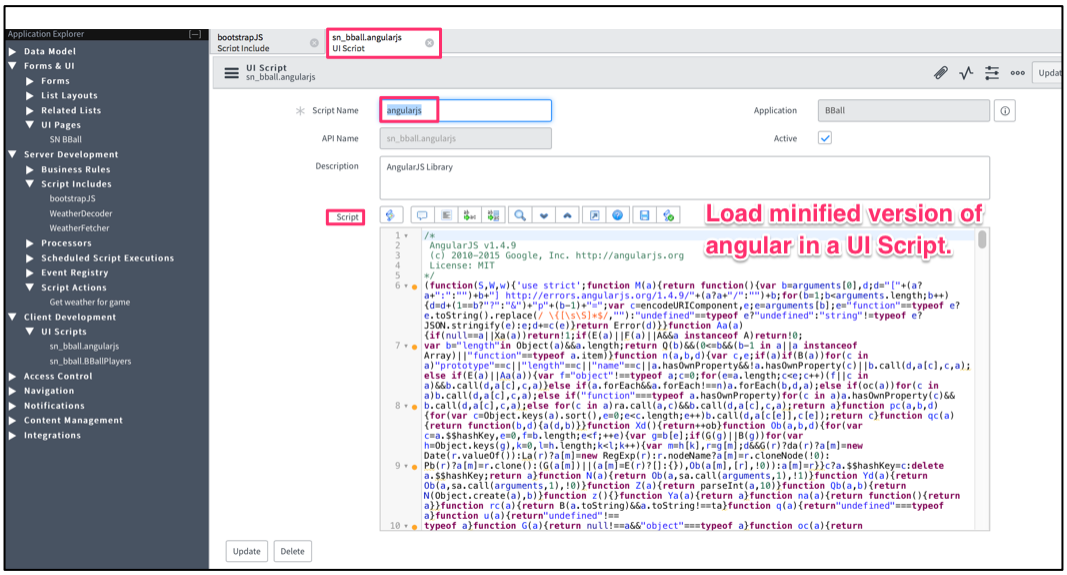


©2016 ServiceNow, Inc. All rights reserved.

Create a Custom UI

1. There are a couple of things we already set up for you, but you should know what we did if you try to do this yourself.

   First, load the Angular library into ServiceNow as a **UI Script**. Create a new **UI Script application file** and Name it **angularjs**. Use the minified version of angular in this script.

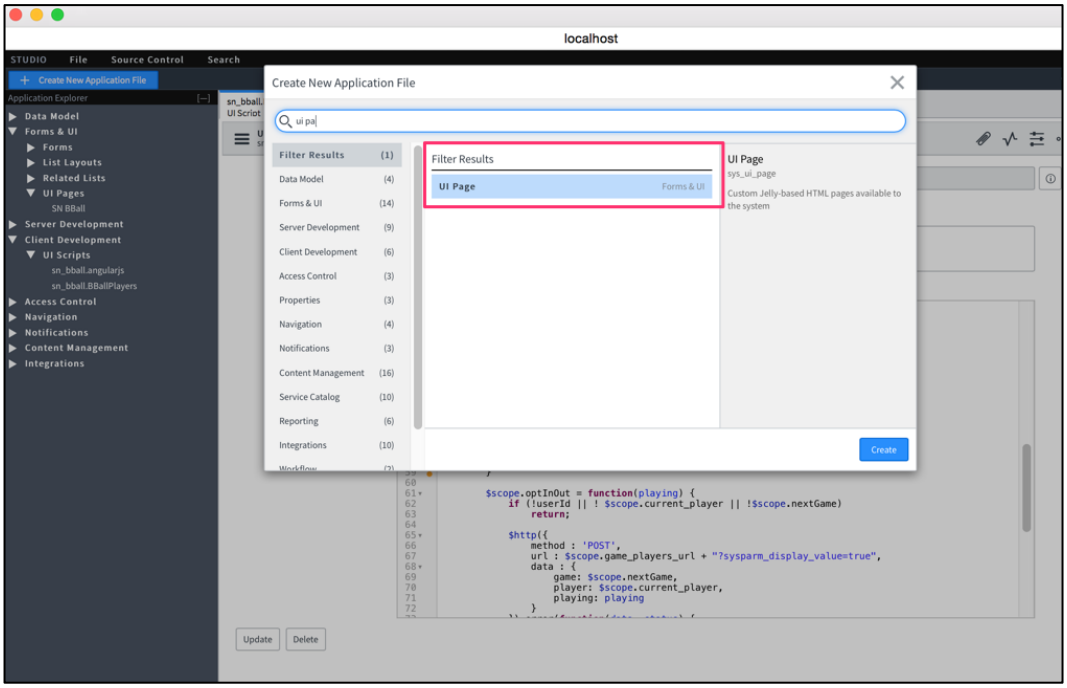


   The last thing we did was to create the CMS Style Sheets. These could be merged into one application file, but for sake of time we did the Style Sheets for you.

©2016 ServiceNow, Inc. All rights reserved.

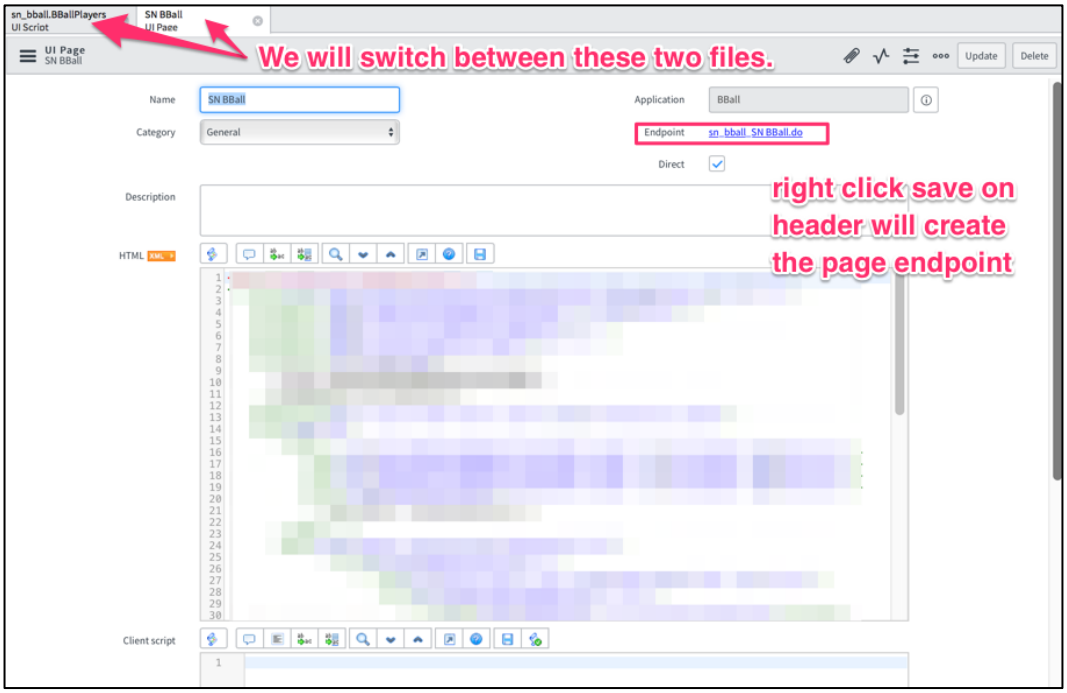
2. For the rest of this lab, you work back and forth (with instructor demo) from the UI page and a UI Script.
   Remember – we do not care about the scripting here (so we are going to make that part easy to copy; but make sure you pick up on the concepts and relationships).

   Create a **UI page**.

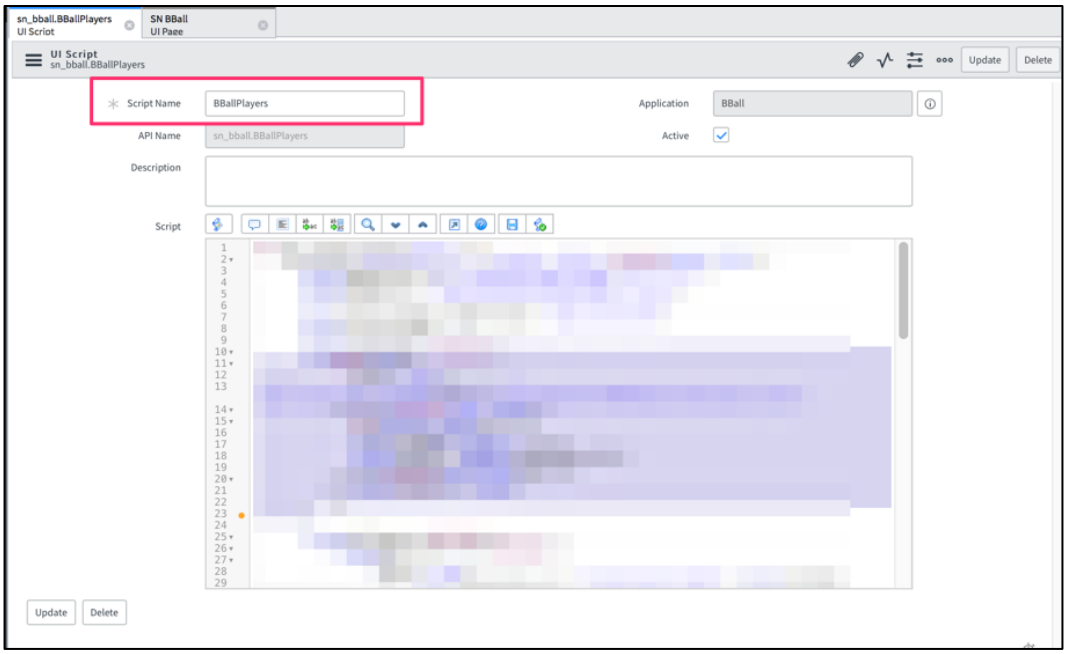


®2016 ServiceNow, Inc. All rights reserved.

3. Name the UI page **SN BBall** and right-click **Save** to get the endpoint created. We work within the HTML section in a little bit. Do not concern yourself with this at the moment.



4. Create a **UI Script application file** and name it **BBallPlayers**. This is where to add the Angular code.



©2016 ServiceNow, Inc. All rights reserved.

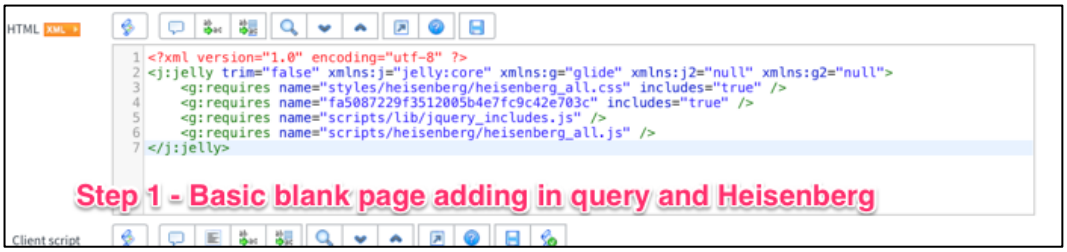
5. Delete all the **default-generated script**.

6. Right-click on header and click **Save**.

7. At this point you have a do-nothing Custom UI page that is blank (with exception of the standard jelly syntax there by default).

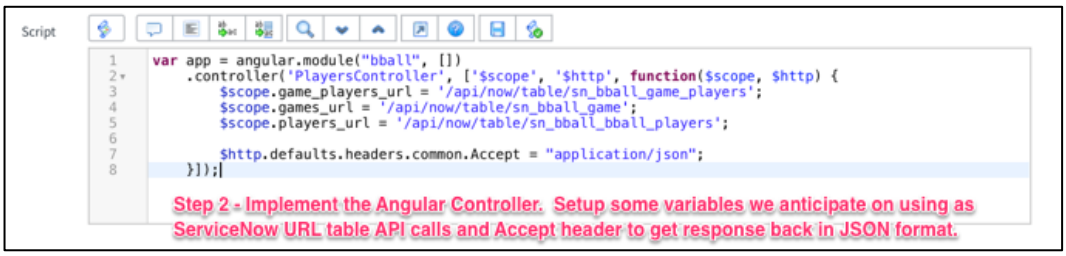


Add the basic required libraries to support jQuery and Heisenberg.



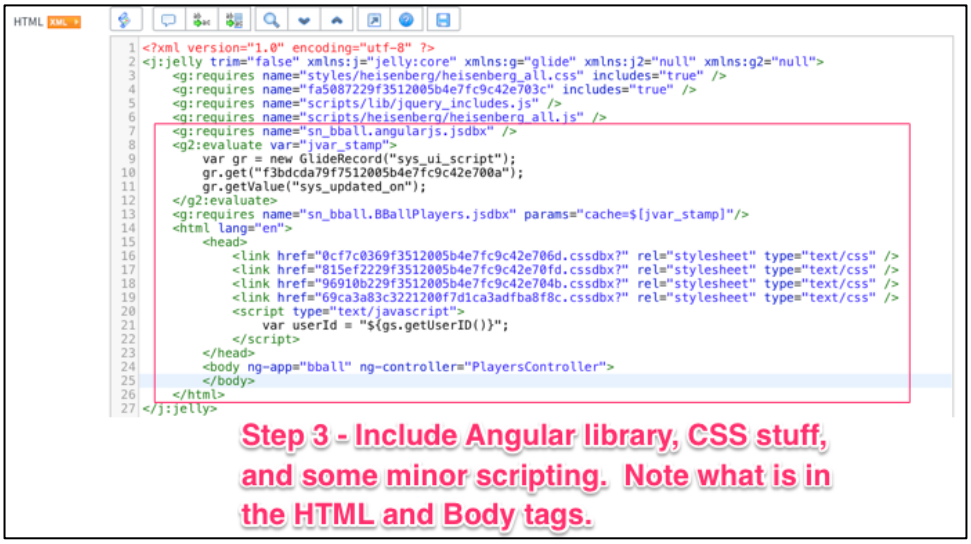
The UI page is still a blank do-nothing page, but now it has more libraries to work with.

8. In the UI Script (bouncing back and forth between your UI page and UI Script as stated earlier) create the **Angular Controller** module and define some of the **table URL API** calls to be used.

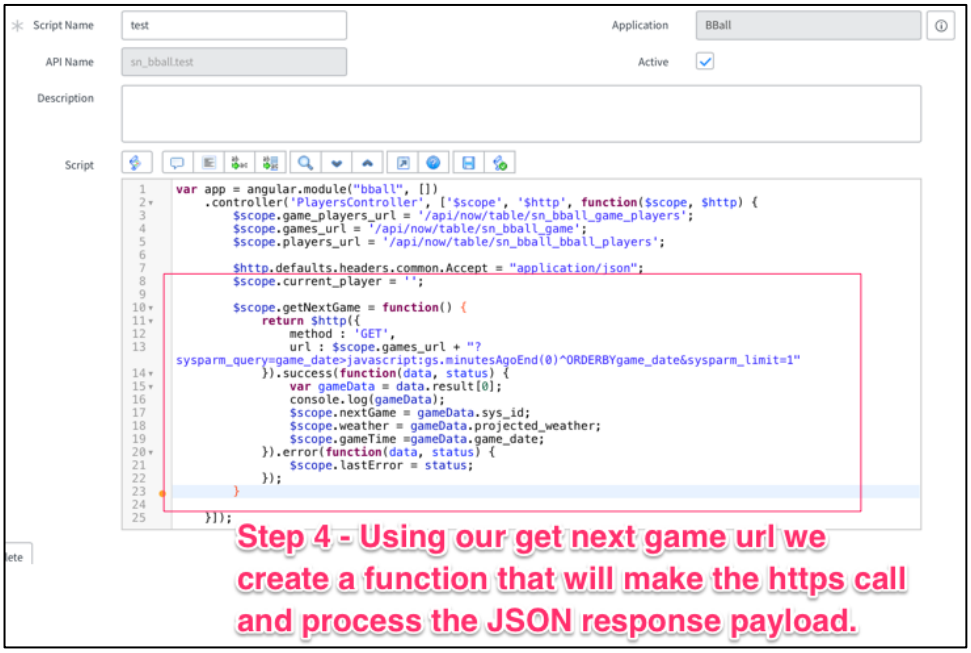


©2016 ServiceNow, Inc. All rights reserved.

9. Include **Angular** and some minor **Jelly scripting** in the UI page. Also invoke the **Angular Controller** in the UI Script in the body of this HTML.



10. Back to the UI Script, create a **function** that gets the next upcoming game.



©2016 ServiceNow, Inc. All rights reserved.

11. Back to the UI page, add **panels** for the current players using **ng-repeat**.



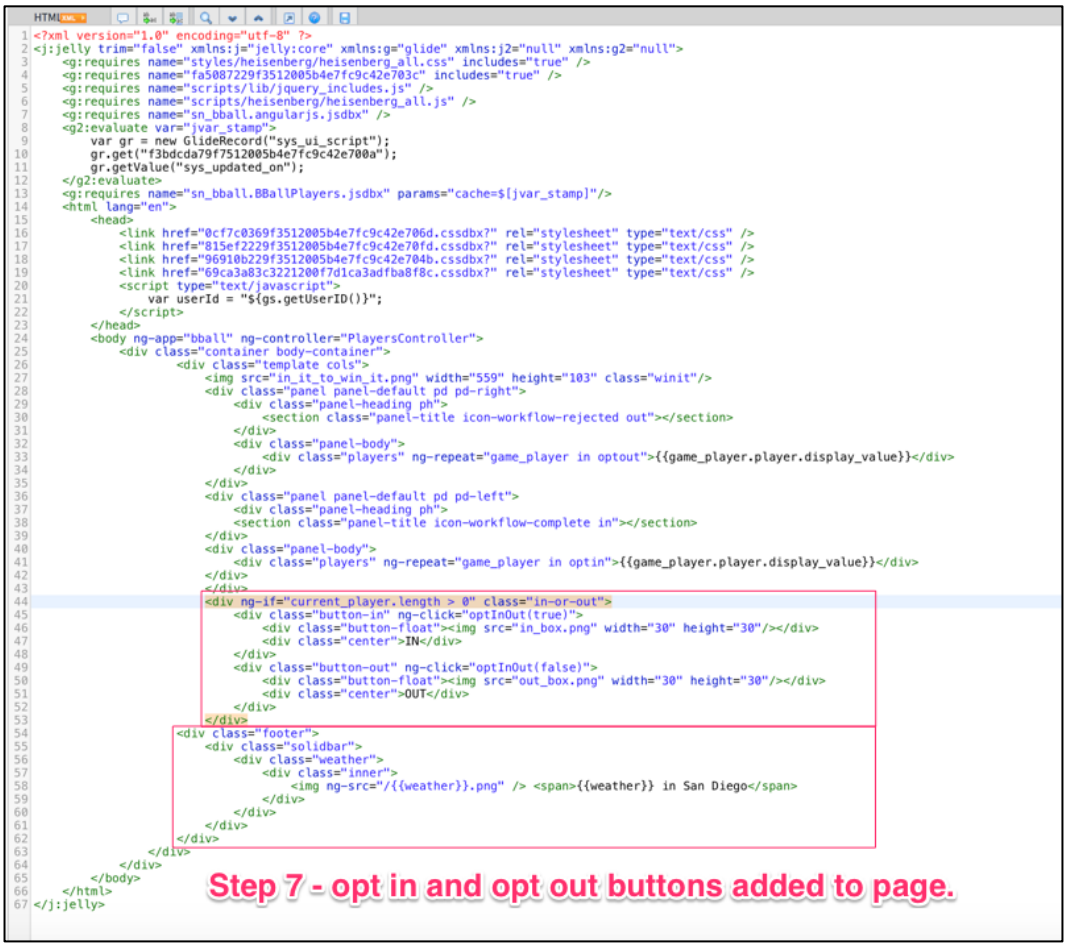
©2016 ServiceNow, Inc. All rights reserved.

12. In the UI Script, implement the rest of the **REST based functions** to use in the UI page.



©2016 ServiceNow, Inc. All rights reserved.

13. Add the **opt in** and **opt out** buttons to the UI page.



```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:requires name="styles/heisenberg/heisenberg_all.css" includes="true" />
    <g:requires name="fa5087229f3512005b4e7fc9c42e703c" includes="true" />
    <g:requires name="scripts/lib/jquery_includes.js" />
    <g:requires name="scripts/heisenberg/heisenberg_all.js" />
    <g:requires name="sn_bball.angularjs.jsdbx" />
    <g2:evaluate var="jvar_stamp">
        var gr = new GlideRecord("sys_ui_script");
        gr.get("f3bdcda79f7512005b4e7fc9c42e700a");
        gr.getValue("sys_updated_on");
    </g2:evaluate>
    <g:requires name="sn_bball.BBallPlayers.jsdbx" params="cache=${jvar_stamp}"/>
    <html lang="en">
        <head>
            <link href="0cf7c0369f3512005b4e7fc9c42e706d.cssdbx?" rel="stylesheet" type="text/css" />
            <link href="815ef2229f3512005b4e7fc9c42e70fd.cssdbx?" rel="stylesheet" type="text/css" />
            <link href="96910b229f3512005b4e7fc9c42e704b.cssdbx?" rel="stylesheet" type="text/css" />
            <link href="69ca3a83c3221200f7d1ca3adfba8f8c.cssdbx?" rel="stylesheet" type="text/css" />
            <script type="text/javascript">
                var userId = "${gs.getUserID()}";
            </script>
        </head>
        <body ng-app="bball" ng-controller="PlayersController">
            <div class="container body-container">
                    <div class="template cols">
                        <img src="in_it_to_win_it.png" width="559" height="103" class="winit"/>
                        <div class="panel panel-default pd pd-right">
                            <div class="panel-heading ph">
                                <section class="panel-title icon-workflow-rejected out"></section>
                            </div>
                            <div class="panel-body">
                                <div class="players" ng-repeat="game_player in optout">{{game_player.player.display_value}}</div>
                            </div>
                        </div>
                        <div class="panel panel-default pd pd-left">
                            <div class="panel-heading ph">
                            <section class="panel-title icon-workflow-complete in"></section>
                        </div>
                        <div class="panel-body">
                            <div class="players" ng-repeat="game_player in optin">{{game_player.player.display_value}}</div>
                        </div>
                        </div>
                        <div ng-if="current_player.length > 0" class="in-or-out">
                            <div class="button-in" ng-click="optInOut(true)">
                                <div class="button-float"><img src="in_box.png" width="30" height="30"/></div>
                                <div class="center">IN</div>
                            </div>
                            <div class="button-out" ng-click="optInOut(false)">
                                <div class="button-float"><img src="out_box.png" width="30" height="30"/></div>
                                <div class="center">OUT</div>
                            </div>
                        </div>
                    <div class="footer">
                        <div class="solidbar">
                            <div class="weather">
                                <div class="inner">
                                    <img ng-src="/{{weather}}.png" /> <span>{{weather}} in San Diego</span>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </body>
    </html>
</j:jelly>
```

Step 7 - opt in and opt out buttons added to page.

©2016 ServiceNow, Inc. All rights reserved.

14. Finally, finish the UI Script to call for player's data and current player information.




```
            }).success(function(data, status) {
                $scope.optin = data.result;
            }).error(function(data, status) {
                $scope.lastError = status;
            })
        }).then(function() {
            $http({
                method : 'GET',
                url : $scope.game_players_url + "?sysparm_query=^game=" + $scope.nextGame + "^playing=0^ORDERBYDESCgame&sysparm_display_value=true"
            }).success(function(data, status) {
                $scope.optout = data.result;
            }).error(function(data, status) {
                $scope.lastError = status;
            })
        })
    }

    $scope.getCurrentPlayer = function() {
        if (!userId)
            return;

        $http({
            method : 'GET',
            url : $scope.players_url + "?sysparm_query=^player=" + userId + "&sysparm_display_value=true"
        }).success(function(data, status) {
            $scope.current_player = data.result[0].sys_id;
        }).error(function(data, status) {
            $scope.lastError = status;
        });
    }

    $scope.optInOut = function(playing) {
        if (!userId || ! $scope.current_player || !$scope.nextGame)
            return;

        $http({
            method : 'POST',
            url : $scope.game_players_url + "?sysparm_display_value=true",
            data : {
                game: $scope.nextGame,
                player: $scope.current_player,
                playing: playing
            }
        }).error(function(data, status) {
            $scope.lastError = status;

            //try updating the existing record?
            $http({
                method : 'GET',
                url : $scope.game_players_url + "?sysparm_query=^player=" + $scope.current_player +
                    "^game=" + $scope.nextGame + "&sysparm_display_value=true"
            }).success(function(data, status) {

                $http({
                    method : 'PUT',
                    url : $scope.game_players_url + "/" + data.result[0].sys_id,
                    data : {
                        playing : playing
                    }
                })
            })
        });

        $scope.getPlayers();
    }

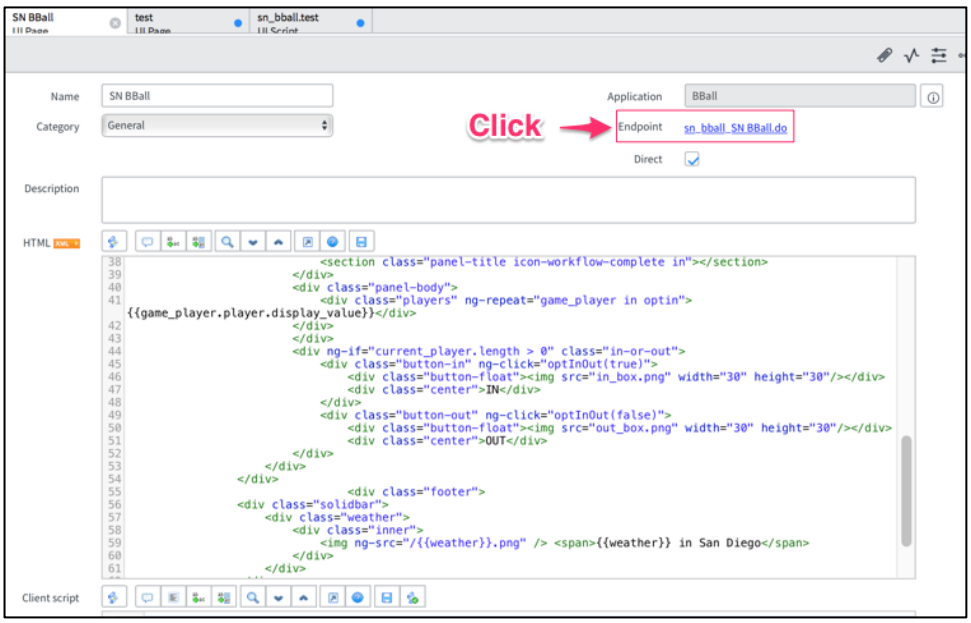
    $scope.getPlayers();
    $scope.getCurrentPlayer();

}]);
```

®2016 ServiceNow, Inc. All rights reserved.

Lab Success Verification

1. At the end of the Angular exercise you can click the UI page endpoint and see the page.





©2016 ServiceNow, Inc. All rights reserved.