Service Portal Lab 1 - Guide

Lab 1.1: Clone Stock Widget 2

Lab 1.2: Adding Options to the Widget 4

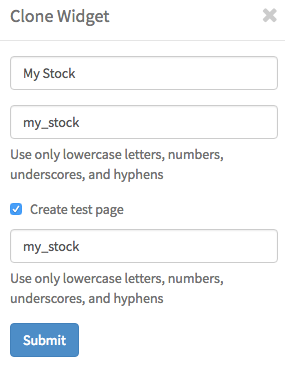
Lab 1.3: Adding Widget specific CSS-SCSS 7

Lab 1.4: Challenge – Working with Unspecified Options 8

# Lab 1.1: Clone Stock Widget

This lab will teach you how to clone an existing OOB widget (Stock) and then applying some basic HTML/CSS changes (no scripting), as well as abstracting some features by using Widget Options. This Lab is just a generic (and very simple) example to cover some of the basic widget concepts.

Go to **Service Portal -> Service Portal Configuration**, open the **Widget** **Editor** and open the **Stock** widget

Click on the menu and choose **Clone “Stock”**. Give the new widget a name of **My Stock** and check the **Create test page.** Then click **Submit**.  
  


**Hint:** You can save the widget by pressing Command + S.

Open the test page in a new window or tab without a portal associated. Remember the **$sp.do?id=*<page\_id>*** trick? ☺

**Lab Validation:**

Your new test page with your cloned widget will look something like this. Test it out by adding a known stock code (e.g. NOW).

**Hint:** meanwhile, Yahoo changed/deprecated their Stock API, so you will not see that graph in your widget anymore, but the price should still show. We will continue this lab either way, you simply won’t see the graph shown in the image below.



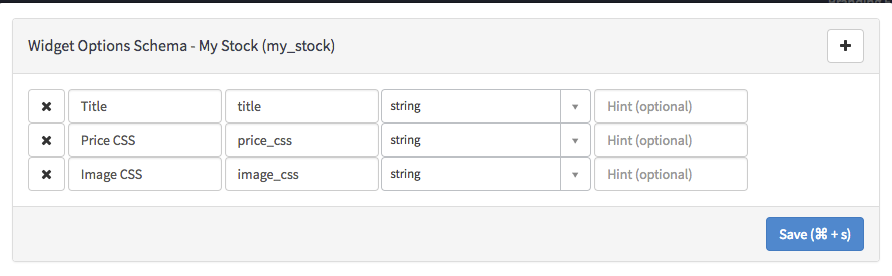
# Lab 1.2: Adding Options to the Widget

Let’s add some options to the widget.

Back in the Widget Editor, click on the hamburger menu and select **Edit option schema.**

Create 3 new options:

* **Label**: Title **Name**: title **Type**: string
* **Label**: Price CSS **Name**: price\_css **Type**: string
* **Label**: Image CSS **Name**: image\_css **Type**: string



Click **Save**.

In order to get access to those options within the HTML Template section, you need to use AngularJS double curly brace notation. Example: **{{options.title}}**. Since the options you created are not dynamic (the value does not change once loaded) you could technically utilize Angular’s one-time binding notation by prefixing it with a double colon (**::**)**.** Either way will work just fine ☺

Let’s now add those options to the HTML template:

1. Add a **h2** tag to the top of the HTML to display the title:

<h2>**{{::options.title}}**</h2>

1. Locate the **span** element where the price is displayed and the style attribute to the existing <span> with the Price CSS option:

<span style="**{{::options.price\_css}}**">{{c.data.price | currency:"$"}}</span>

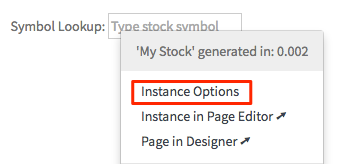
1. Locate the **img** element where the stock image is displayed and add a new style with the Image CSS option:

<img style**={{::options.image\_css}}** ng-src="http://chart.finance.yahoo.com/z?s={{c.data.symbol}}&t=1d&z=l"/>

Click **Save**.

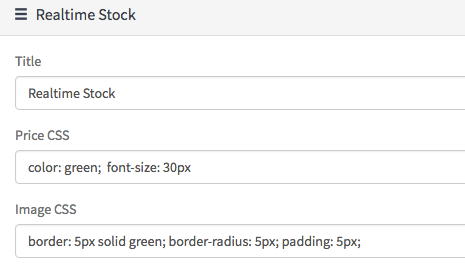
Open or reload the test page again – you’ll notice nothing has changed yet. The reasons for this is, that our options are now available to the widget, but we haven’t set any values in the widget instance yet.

Right-click + CTRL on the widget instance and select **Instance Options**:

****

Enter a value for the **Title** and then some CSS style to **Price CSS** and **Image CSS** option. Click **Save**.

* Title:  **Realtime Stock**
* Price CSS: **color: green; font-size: 30px**
* Image CSS: **border: 5px solid green; border-radius: 5px; padding: 5px;**

****

The page is reloaded and you can immediately see the title you added above. Our styles will only appear, once you provide a stock symbol.

**Lab Validation:**

Your test page should now look like this.



# Lab 1.3: Adding Widget specific CSS-SCSS

Now we are going to add some widget specific CSS-SCSS styling.

Back in the **Widget Editor**, create a new class called named **myTitleClass** and specify any style you want. (Optional) if you feel confident you can also try and use SCSS syntax variables:

.myTitleClass {

font-size: 25px;

color: $brand-danger;

font-weight: bold;

text-decoration: underline;

}

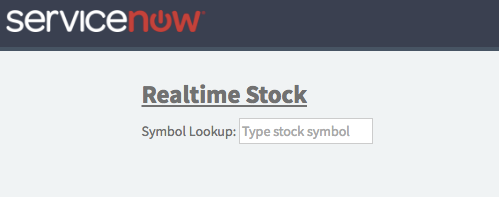
Add the new class to the **h2** element tag:

<h2 **class="myTitleClass"**>{{::options.title}}</h2>

Click **Save**.

**Lab Validation:**

Open your test page within a portal (e.g. **sp**). The widget should look something like this with your new class applied:

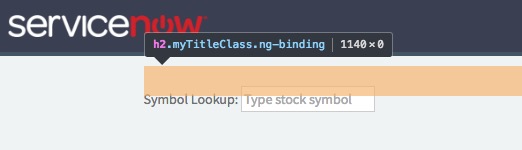


# Lab 1.4: Challenge – Working with Unspecified Options

There may be a situation in which there are no values for the widget options.

For this kind of scenario, it’s best to add an additional check to ensure the option has a value.

Modify the instance options and clear the text for the **Title**. Notice how even though the title is not set, the h2 tag is still rendered and occupies a visible “empty” space on the page. **Hint**: You can use your browser’s Inspect functionality for this.



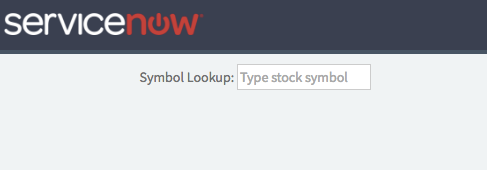
Now we will leverage AngularJS to hide the title, in case there is no value for it.

Within this widget, there is already an example on how to achieve this. Have a quick peek at the <span> element in front of the **${Requesting stock price}** text.

This element is using an AngularJS directive called **ng-if**. There’s also other alternatives (ng-hide and ng-show) that could accomplish sort of the same; the difference and best use cases for each of these will be discussed later during the class ☺

So for your challenge, use the ng-if directive, see if you can remove the h2 element altogether from the page **if** the Title option is not set.

Once you’re done, the page should look something like this with NO visible empty space. Remember to also inspect the HTML so to double check the h2 tag is definitely no longer there.



**…And with that, you’ve successfully completed Lab 1 ☺**