CREATORCON™

May 9–11 • Orange County Convention Center • Orlando, FL

Lab Guide

Service Portal and Mobile

Dave Slusher

Lab instance: https://clabs.link/sp-mobile-pnq

Default Login / Password:

admin / pnq-cc17

itil / pnq-cc17

employee / pnq-cc17

This

Page

Intentionally

Left

Blank

Lab Goal

Lab 1
Creating a new widget

During this course, we will be creating a Service Portal experience that looks great in the ServiceNow mobile app – an experience that integrates deeply with native components and leverages native APIs.

In this course, we will create a Service Portal page, add a widget that scans bar codes and creates incidents from them, and add that page to our Mobile App homepage. Additionally, we will create a special 'Escalate Incident' UI Action on the Incident List that will conditionally open a Service Portal page to provide more functionality.

Let's get started!

Create a new widget

1. In the navigator to the left, search for **Service Portal Configuration.** Click it to open the Service Portal Configuration page.

2. From here, select the **Widget Editor**. From here, we will create a new widget, as well as a page for that widget to live in.

3. Select **Create a new widget** on the editor. In the dialog that will appear, you will provide three things: a name for your new widget, an ID for the widget, and once you select **Create test page**, you will also provide a page ID that your widget will live on. Creating test pages is an easy way to quickly bootstrap a page in Service Portal.

Name: **K17 Barcode Scanner**
ID: **k17_barcode_scanner**
Page ID: **k17_barcode_scanner**

Our page is now available by navigating to **/$sp.do?id=k17_barcode_scanner**. Since we need a way to reach the page on our mobile device, let's create a link to this page in the navigator.

Create a Navigator Module

From the Platform UI, search for **Navigator Apps** in the navigator and select it. From the list of navigator apps, select **Service Portal**. Now, in the related list, click **New.** Provide the following configuration:

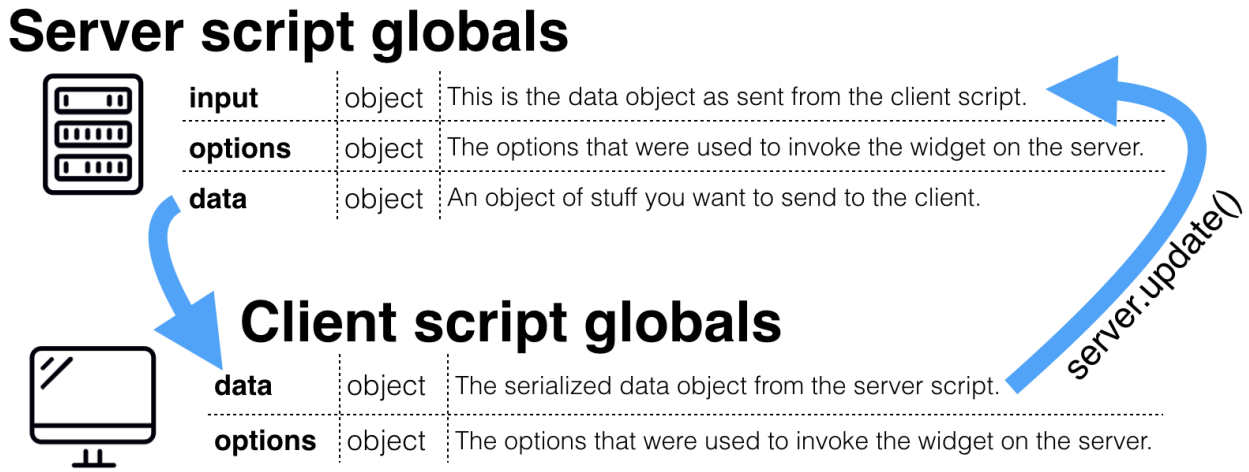
Name: **Create Incident**
Path: **/$sp.do?id=k17_barcode_scanner**
Path Relative to Route: **Checked**

Hit **Submit**, and you're done!

Widget 101

A high level overview of widgets (just in case you need a refresher)

The following graphic shows what variables are globally available in each scripting environment and how they are initialized.



Server Script

1. An empty **data** object is initialized.
2. Use the **input** or **options** objects to get any data that was sent from the client controller or used to initialize the widget from the server.
3. After the server script executes, the **data** object is json serialized and sent to the client controller.

Client Script

1. Everything from the server's data object can be accessed via **c.data** or $**scope.data**. (You will learn more about c and $scope later)
2. If you change a value in the data model, use **server.update()** to post the entire data object to the server script as **input**.
   Note: After calling **server.update()** the client script's **data** object is automatically overwritten with the server's data object.
3. Use **c.options** or **scope.options** to see what values were used to invoke the widget. This object is readonly and making changes will have not be visible in the server script.

Lab Goal

Lab 2
Utilizing the Cabrillo API

Good work – you created an empty screen! However, that isn't going to do any of our users much good. Let's use the power of the Cabrillo API to build a truly mobile experience! Cabrillo is special because it gives us, as widget developers, access to a number of helpful Native mobile constructs and tools like navigation buttons, isNative checks, and the camera. There's something you can't do on the desktop web!

The first pass of our widget will be simple – scan a barcode, then display the contents of the barcode on the screen.

Build your widget

1. Here, we're going to exercise those copy-paste muscles. There's quite a bit of code that makes up this widget, so here's a convenient place to get the code and paste it into your widget: http://mobile.servicenow.com

2. Let's take a closer look at some of the Cabrillo APIs that we're leveraging here.

cabrillo.isNative()

Since we know that our barcode scanning experience is only going to work in a native mobile environment, we can feature-gate our widget by calling isNative(). This way, our desktop users will be greeted by a helpful error message if they somehow manage to visit our page from a desktop computer. We could even provide an alternate experience for those users!

cabrillo.viewLayout.setBottomButtons()

With this API, we can add a custom native button to the bottom of the screen, and tie an action to it. Pretty useful!

cabrillo.camera.getBarcode()

The star of the show in this widget – by calling this API, a barcode scanner provided by Cabrillo is summoned and, upon recognizing a barcode, will return a promise that resolves with a value derived from the barcode.

Testing Our Widget

To take the widget for a test drive, open the navigator on your mobile app by pressing the list icon in the bottom-left corner of the screen, and search for Create Incident. Press the resulting link and scan the following QR code on the page that is presented to you:

Lab Goal

Lab 3
Creating an Incident from the Bar Code

We now have a fully-functional service portal page for scanning bar codes, but it doesn't do anything useful with that information yet. Let's update our scanner widget to automatically create an incident for the scanned asset. Since we will want to prompt the user for additional details about the broken coffee machine, we should also provide another screen to the user. This is a great use case for Cabrillo modals!

Build your widget

1. This time, for convenience, we included the 'Create Incident' widget and page out of the box in your instance. All you need to do is open up your widget editor and locate the widget titled **K17 Asset Incident Confirmation**

2. You'll notice that our widget, sadly, is empty inside. Let's give it some meaning by pasting in some code, which can be found at http://mobile.servicenow.com

3. Since we're adding in support for opening up a modal, we will want to update our original bar code scanner with some new content. Open up **K17 Barcode Scanner** and copy-paste in the code for the next step, again found at http://mobile.servicenow.com

4. Let's take a closer look at the additional Cabrillo APIs that we're leveraging in this lab.

cabrillo.modal.presentModal()

If you have spent a lot of time building with Service Portal or with other web development frameworks in the past, you'll know that creating modal behaviors can be pretty tricky. With Cabrillo, it's easy! You can pass in any arbitrary page link (Like, say, another service portal page) and the page will be natively presented in a modal.

cabrillo.viewLayout.setNavigationBarButtons()

With this API, we can add custom native buttons to the mobile app's navigation bar and tie an action to each button. This API works just like the setBottomButtons() that you've already used. Too easy!

cabrillo.viewLayout.showSpinner()

Showing a native spinner HUD has never been simpler. Just call showSpinner() to show a spinner, do whatever asynchronous work you need to do (like saving a record) and when you're finished, call hideSpinner().

cabrillo.modal.dismissModal()

When it's time to dismiss the modal, the presented page can call dismissModal() and pass back data into your modal-calling widget. Powerful stuff!

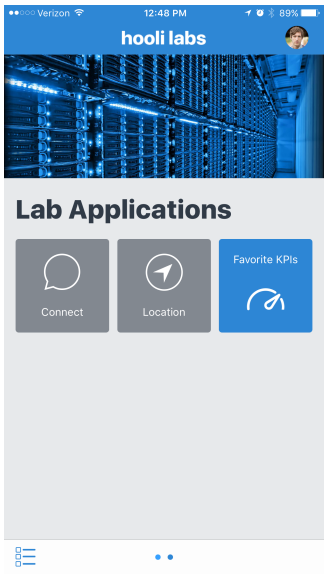
Lab Goal

We now have a fully-functional service portal page for scanning bar codes and turning them into incidents, but that doesn't do us any good if our users cannot easily access it. Let's make use of the built-in Mobile App home screen and add a link to our service portal page there.

Lab 4
Configuring our mobile home screen

Modifying a mobile home page

1. From the navigator in the platform UI, search for **Home Page** – select Home Pages under the Home Screen dropdown
2. From the list, select the home page titled **Lab Applications**. This is the home screen that we will add our content to!
3. Currently, our home screen looks something like this:



We will add some more items to this screen so that our users can easily create and view their incidents.

4. Let's create a new section for our links to live in. In the **Sections** related list, select **New.**
5. In the form, provide the following information:
   Title: **Incidents**
   Hide Title: **True**

Module Style: **Regular**
Active: **True**

6. Submit the form, and now we have a new section! Let's give it some content. Click into the new section we have created and, in the related list entitled **Modules**, click **New.** Provide the following information to the module:
   Title: **Create Incident**
   Subtitle: **Make a request or report a problem**
   Module: **SEE STEP 7**
   Icon: **Edit**
   Background color: **#2d86d5**
   Content Style: **Light Content**
7. For the Module field, we can provide the module that we created back in Lab 1! How convenient!
8. Now we have a link that our users can use to create an incident with our fancy new Service Portal page! Let's also provide them a list of incidents so they can see the incidents that they created. We provide this out of the box in your instances to save a bit of time (configuration of this is more of the same). Click **Edit** in the Modules related list, and select the mobile module entitled **My Incidents**.
9. Since we made an update to our mobile app, we will need to manually reconnect to our instance inside the app in order to see our changes. This is due to the metadata caching that the app does behind-the-scenes to keep things quick and snappy. To refresh the app, tap your user profile in the top-right corner, then tap **Switch Instance**. Re-select your instance name from the list that appears on the screen.
10. Your mobile home screen should now look something like this – well done!

Lab Goal

Lab 5
List UI
Actions

Now that we have a beautiful mobile home page, lets further enhance that Incident list. To start, we are going to create a mobile List UI Action that will enable our users to quickly escalate and de-escalate their incidents, without having to navigate to a form! We will also provide a hook for conditionally opening a Service Portal page when certain types of incidents are escalated.

Creating a Mobile UI Action

1. Search for UI Action in the platform UI Navigator, and select the item titled **UI Actions – Mobile**
2. From this list, click **New**. Provide the following configuration to the form:
   Name: **Escalate**
   Action name: **escalate**
   Table: **Incident**
   List button: **Checked**
   Show for: **Any**
   Condition: **current.impact != '1' && current.urgency != '1'**
   Hide when disabled: **Checked**
   Background: **#FF402C**
   Script: **Provided at GitHub**

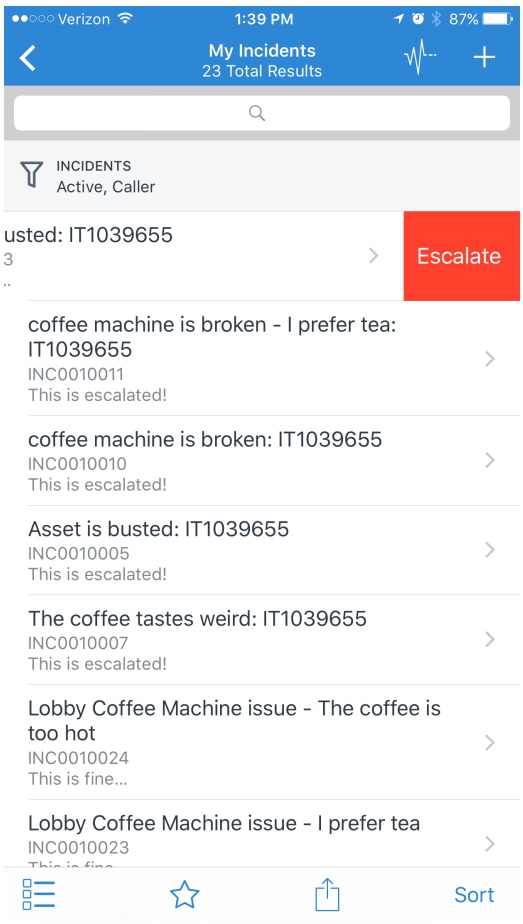
```
current.impact = '1';
current.urgency = '1';
current.update();

var isCoffeeIncident = current.short_description.includes('coffee');

if (isCoffeeIncident) {
    action.setRedirectURL('/$sp.do?id=k17_escalate_coffee_incident');
}
```

Once complete, **Submit** the form.

Congrats! Now we have a useful mobile list action for our users.



Lab Goal

The escalate / deescalate UI action works great, and it works on any incident. But we want better for our users! We want to build them a great experience where, if the user escalates an incident related to the coffee machine, we can recognize their immediate need of coffee and prompt them to order a coffee delivery! These are the types of dynamic experiences that we can create with the mobile app and with Service Portal. Let's augment our UI action and really score big with our users.

Lab 6
Cabrillo Location API

Build your widget

1. This time, for convenience, we included the 'Create Incident' widget and page out of the box in your instance. All you need to do is open up your widget editor and locate the widget titled **K17 Coffee**

2. Once more, we're going to paste in some code, which can be found at mobile.servicenow.com

3. Let's take a closer look at the additional Cabrillo API that we're leveraging in this lab.

cabrillo.geolocation.getCurrentLocation()

As the name suggests, by using native mobile device GPS, we can retrieve the current latitude and longitude of a user through a simple API call. This API returns a promise containing the device location for an employee on-the-go.