

Lab Guide

Creating A Custom Service Portal Widget: PayPal Integration

Frank Schuster

This
Page
Intentionally
Left
Blank

Intro

The objective of this lab is to learn how to create a custom Service Portal widget that allows a user to pay something (e.g. a Requested Item) via Credit Card. Some of the features included in the widget are:

- Payment Modal (Bootstrap form)
- REST Integration into the PayPal Credit Card API by using JSON objects
- Utilizing the actions in the widget to update the Requested Item record
- Pass data back and forth between HTML, Client & Server

Widgets can contain HTML (AngularJS Directive), CSS (SASS), Client Scripts (AngularJS Controller) & Server Scripts (e.g. GlideRecord, Web-Service calls etc.). Widgets can be added to multiple pages in multiple portals.

Note: This Lab requires a PayPal account! If you do not have one, you will have to create a PayPal account moving forward. If you do not want to do this you can utilize the attributes from an existing PayPal account, but then you will not be able to debug your API calls within PayPal itself.

All necessary components to reproduce this lab can be found in a Github repository:

<https://github.com/schusterf/service-portal-hacklab>

- ServicePortalLab-Backend.xml (already deployed to your Lab Instance, no need to download)
- Widget-HTML.html
- Widget-ng-template.html (Angular ng-template > Related List on the Widget record)
- Widget-Client.js
- Widget-Server.js
- ServicePortalLab-Widget_UpdateSet.xml (Update Set that contains the complete Widget – only use if you do not want to build it yourself or if you were not able to finish it)
- Accepted Cards & Custom Loading Spinner Images

The Backend update set that contains everything you need in terms of preparing the system for building out your widget:

- PayPal Catalog Item
- PayPal Workflow
- Backend adjustments (Requested Item table)

Lab Goal

This section is a prerequisite for the actual creation of the widget. First you need to make sure that you have a PayPal account that you can use for development efforts. If you do not wish to use your own PayPal account, please proceed directly to Lab 2.1 and leverage the pre-configured ClientID & Secret, that you need for the integration.

Lab 1.1 Configure the PayPal account

1. Navigate to: <https://developer.paypal.com/>
2. Click **Log In** and use your regular PayPal credentials to log in
3. If you do not have a PayPal account: now is the time to create one 😊. If you do not want to create one skip this step and utilize the Client ID & Secret, which are included in the **Widget-Server.js** file. You will **not** be able to debug your calls within the PayPal developer portal if needed.
4. After you successfully logged in, navigate to **Dashboard** and on the side to **My Apps & Credentials**. Click **Create App** within the **REST API apps** section (you have to scroll down a bit).

The screenshot shows the PayPal Developer Dashboard. The left sidebar has a 'My Apps & Credentials' link highlighted with a red box. The main content area is titled 'REST API apps' and contains a 'Create App' button, also highlighted with a red box. Below the button is a table with one entry: 'servicenow-hacklab' of type 'REST'. The table has columns for 'App Name', 'Type', and 'Actions'. Below the table, there is a section for 'NVP/SOAP API apps' with a 'Manage NVP/SOAP API apps' button.

App Name	Type	Actions
servicenow-hacklab	REST	

5. Create a new App by providing a name for it – notice your sandbox developer account will contain your original PayPal email-address and **-facilitator**.

Create an app to receive REST API credentials for testing and live transactions.


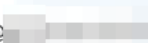
 Features available for live transactions are listed in your [account eligibility](#).

Application Details

App Name

servicenow-hacklab-test

Sandbox developer account

-facilitator@

As a reminder, all apps created under your account should be related to your business and the type of business it conducts.

By clicking the button below, you agree to [PayPal Developer Agreement](#).

Create App

6. After you created the App PayPal will redirect you right into the newly created application. For requesting an OAuth token from PayPal out of ServiceNow you will need the **Client ID** and the **Secret** (click **Show** to actually show the Secret) later on when you create the Server Script.

SANDBOX API CREDENTIALS

Sandbox account

-facilitator@

Client ID

AaRUqMvjYBZ1FE8dEt2mamy3WlHaZJU4SRehZGkv3A46tSPND2A2EvSSCeg6

Secret

Hide

Note: When you generate a new secret, you still maintain the original secret. The maximum number of client secrets is two. A client secret is either in enabled or disabled state.

Created	Secret	Status	Action
Apr 26, 2017	EHgHpMx3HTzgZOf4mG_WU_tPTHdosgkOSDoSgwMQgUwqg6ChFPdRRns-SP3KZpzc1uz	Enabled	Disable


Generate New Secret

7. At the bottom of the page, make sure that “Accept payments” is checked.

SANDBOX APP SETTINGS

Return URL- Users are redirected to this URL after live transactions. Allow up to three hours for the change to take effect. [Show](#)

App feature options

☒ **Accept payments** Accept one-time and subscription payments from PayPal members using PayPal and direct credit card processing. [Advanced options](#) 

☐ **Invoicing** Issue invoices for payments owed, manage partial balances due, and enable custom net payment terms.

☐ **Payouts** Send batch payments to multiple PayPal accounts at once. You can vary the amount by recipient and select if you'd like it delivered by phone number or email.

☐ **PayPal Here** Process swiped/card-present card transactions.

☐ **Log In with PayPal** Identity service that enables your customers to log in with their PayPal login.

[Save](#) [Cancel](#)

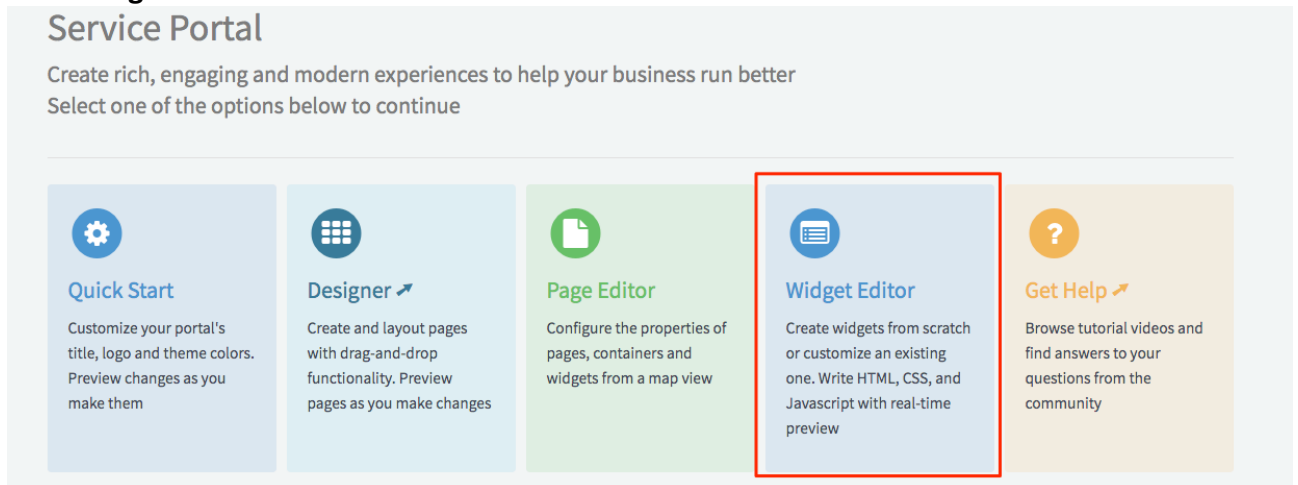
Lab Goal

As a first step, we will create the Widget through the Service Portal Configuration UI. This will then create a new record on the Widget [sp_widget] table.

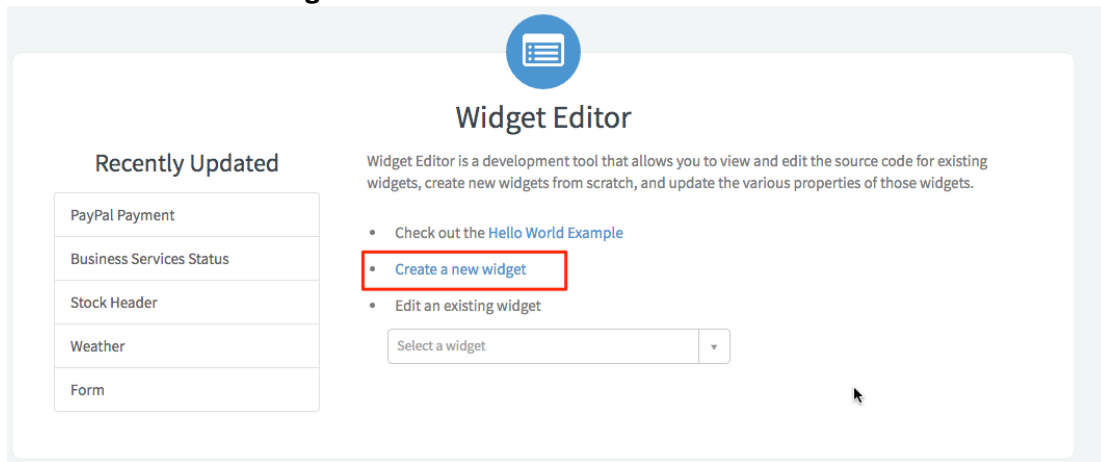
Lab 2.1 Creating the Widget

Creating the Service Portal widget

1. Navigate to **Service Portal** -> **Service Portal Configuration**.
2. Click **Widget Editor**.



3. Click **Create a new widget**.



4. In the modal that will come up, provide a name for the widget, e.g. **PayPal Payment**.

Lab Goal

In this lab, you will learn how to create the HTML for your widget by using Bootstrap and Angular directives.

The HTML consists of two parts: a payment button within a panel and a modal, that is being opened on click of the payment button.

The relevant content can be found in the following Github repository:

- <https://github.com/schusterf/service-portal-hacklab>
- Files for this lab:
 - **Widget-HTML.html** (Widget HTML)
 - **Widget-ng-template.html** (Angular ng-template)

Lab 2.2 Creating the HTML (Angular Directive)

Creating the Payment button

1. Copy & Paste the **Widget-HTML.html** content into the **HTML Template** section of your Widget and click **Save**. **Hint:** you can click on the HTML file in the Github repository and then open it in **Raw** mode, that allows you to easily copy & paste the code, rather than downloading it.
2. Comment out line 2 in the HTML. Since we do not have a Server Script right now, that ng-if would always default to **false**, which would hide the Payment Button during the lab validation of this step. Line 2 & 3 of your HTML should then look like this (we will revert this step later, when the server script is added):

```
<!-- <div ng-if="data.showPayment"> -->  
<div>
```

Explanation:

data.showPayment is a variable that is set by the server script (**Lab 2.4**) – it evaluates if the current stage of the Requested Item is “Payment Requested”. In case you are **not** using the Catalog Item from the update set with the according workflow and that **exact** stage, simply remove the **ng-if** statement from the outer <div> and the Widget will always show up.

The button tag will call a function on the Client Controller, which will then open up a Bootstrap Modal (from an ng-template).

Create the Payment form (inside the Modal)

At this point you only have a button, but it would not trigger anything.

The HTML for the Modal is included in the same repository:

- <https://github.com/schusterf/service-portal-hacklab>
- File: **Widget-ng-template.html**

To proceed, you will need to download the two images in the Github folder: Loading Spinner & Accepted Cards.

Step 1:

Navigate to **System UI > Images** and create two new image records (categorize them as **Service Portal**):

- accepted-cards.png (use the Download button or do a right-click + Save Image as)
- portal-loading-spinner.svg (in the Github repo, open the spinner and do a right-click + Save Image as)

Make sure you name the images exactly like above, otherwise you will have to change those names in the Modal code.

It is a good practice to leverage Angular templates as the container for the actual modal code, to keep the widget HTML lightweight.

Step 2:

1. Navigate to **Service Portal > Widgets**, search for the **PayPal Payment** Widget and open it.
2. Once the record is open, scroll down to the **Related** Lists, look for the **Angular ng-templates** Related List and click **New**.
3. Provide an **ID** of **paymentModal** and copy & paste the code snippet from the Github file in the **Template** field.
4. Click **Save/Submit**.

Explanation:

The Angular ng-template holds the actual Payment form. It could very well be created in the Widget itself, but from a maintenance perspective, this is the better way.

Bootstrap forms can be constructed very easily by e.g. using a Bootstrap form builder: <http://bootsnipp.com/forms> or by searching for form snippets, i.e. on bootsnipp.com or other Bootstrap template pages. There are also plenty of Bootstrap Tools out there, that can help you design content, e.g. Bootstrapstudio.io or PineGrow Web Designer.

If you would want to build your own form, the **mandatory** fields for your integration would be:

- Type (e.g. VISA, MasterCard) [ng-model: c.data.credit_card_type]
- First Name [ng-model: c.data.first_name]
- Last Name [ng-model: c.data.last_name]
- Number [ng-model: c.data.credit_card_number]
- Expires (Month) [ng-model: c.data.expire_month]
- Expires (Year) [ng-model: c.data.expire_year]

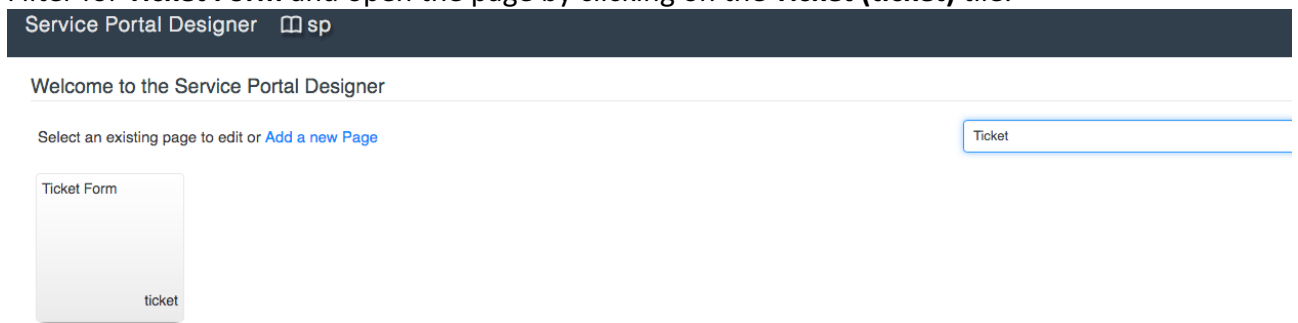
The **optional** field for your integration is:

- (CVV2) – seems to be optional in the PayPal credit card API, can be provided but it is not mandatory

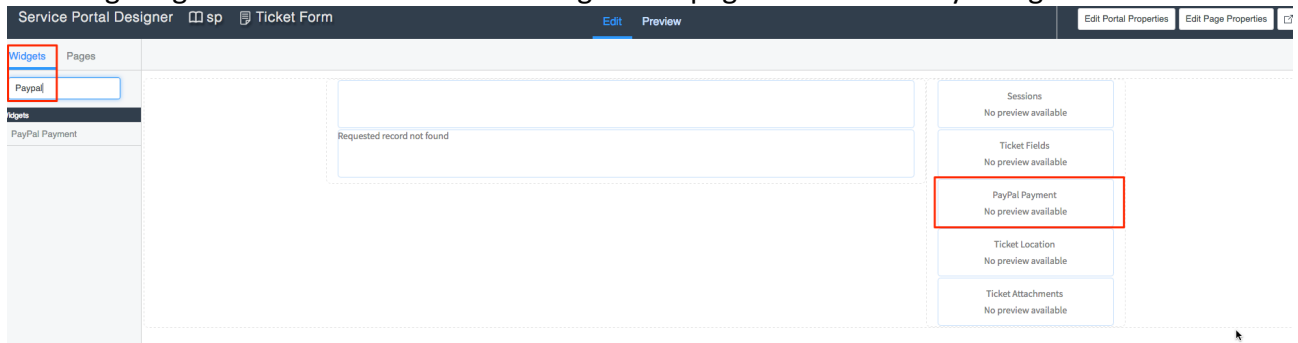
When writing your own HTML do not forget to define an **ng-model** attribute for each input field, since we have to bind each HTML element to the **data** object, in order to being able to use it in the Server Script later.

Adding the Widget to the Ticket Page & Lab Validation

1. To test the HTML of your created widget you will add it to the Page **Ticket Form** with the ID **ticket**.
2. Open the Service Portal Designer from the **Service Portal > Service Portal Configuration** page.
3. Filter for **Ticket Form** and open the page by clicking on the **Ticket (ticket)** tile.

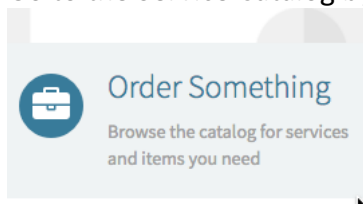


4. Filter for the PayPal widget by using the search field when **Widgets** is selected. Drag & Drop the widget right below the **Ticket Fields** widget. The page is automatically being saved.

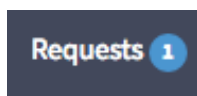
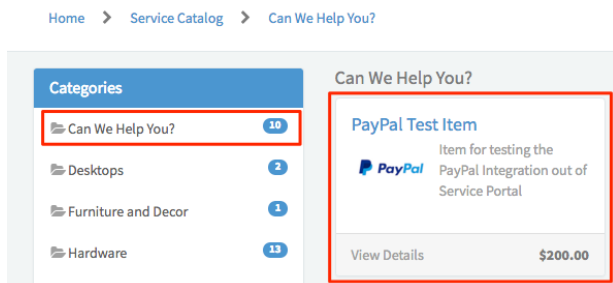


5. In order to test that change you made to the form, you will have to order the **PayPal Test Item**. Go to the Service Portal index page.
<https://yourInstance.service-now.com/sp>

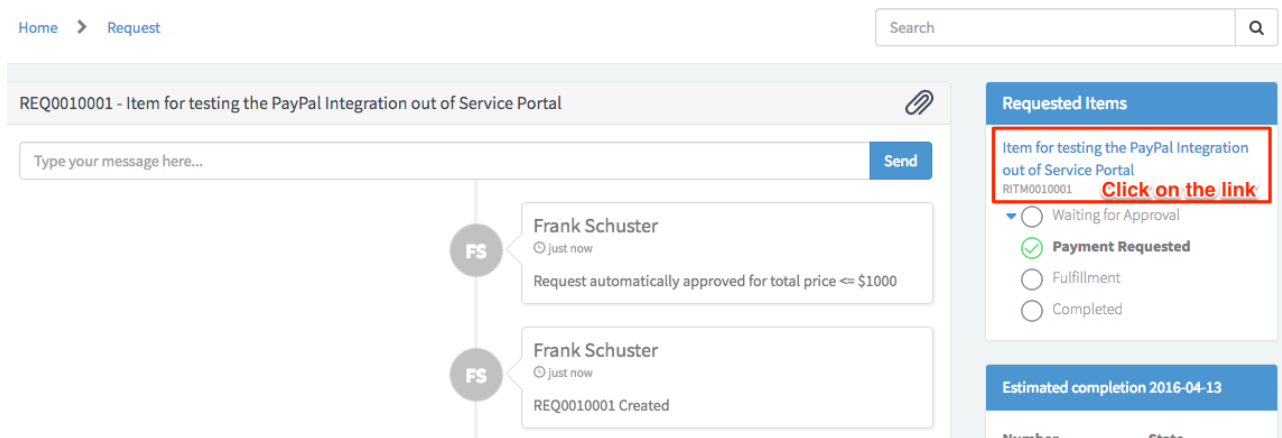
- Go to the Service Catalog by clicking on **Order Something**.



- Pick the item that came with the Git repository (**PayPal Test Item** – within the category **Can we help you?**) and submit your request. After you have submitted your request the top navigation should dynamically change without reloading the page and show a **Request (number of requests)** menu item.



- Click on the Request and drill down into the actual Requested Item by clicking on the name of the item in the right hand side **Requested Items** widget.



9. On the right side you will find your **PayNow** widget. If your Catalog Item has Catalog variables they will appear as “Options” – minimize those to see by clicking on the down arrow in the widget. A click on the **PayNow** button will not show the modal yet, since we leverage UI Bootstrap to render the modal. The Button actually just calls a function in the client script, which will then open the modal. The client script will be created in the next step.

The screenshot displays a ServiceNow interface. At the top, there is a breadcrumb navigation bar with 'Home' and 'Requested Item', and a search bar. Below this is a 'Conversation' section with a text input field 'Type your message here...' and a 'Send' button. A message bubble from 'Frank Schuster' (4m ago) contains the text 'RITM0010001 Created'. A green 'Start' button is visible. To the right, a 'Your request has been submitted' summary box shows details for request RITM0010001, including its state (Open), priority (4 - Low), creation time (4m ago), update time (4m ago), and price (\$200.00). Below this, a note states 'Tickets are picked up within 4 hours (M-F 9-5)'. Further down, a 'Make a Payment' section features a green 'PayNow' button, which is highlighted with a red rectangle. Below the payment section are 'Location' and 'Attachments' sections, with the latter showing a 'Drop files here' prompt.

Lab Goal

This lab introduces you to the concept of passing information from your HTML section to the Client Script (Angular Controller). In the Client script you can e.g. close the modal after a payment has been made successfully.

Write the Client Script

Within the Github repository, open the following file:

- **Widget-Client.js**
 1. In the Widget Editor, check the **Client Script** checkbox in order for the script section to show up. When writing the Controller, it can be helpful to display the HTML as well, since there are functions and variables that you will have to use in your controller.
 2. Copy & Paste the content from the file into the Client Script part of the Widget and click **Save**.

Explanation:

The **payNow** function is being called, when a user clicks **the Pay NOW!** Button in the HTML (the call is fired on submit of the <form>). **c.server.update()** will cause your Server Script to run again (it also runs onLoad of the widget).

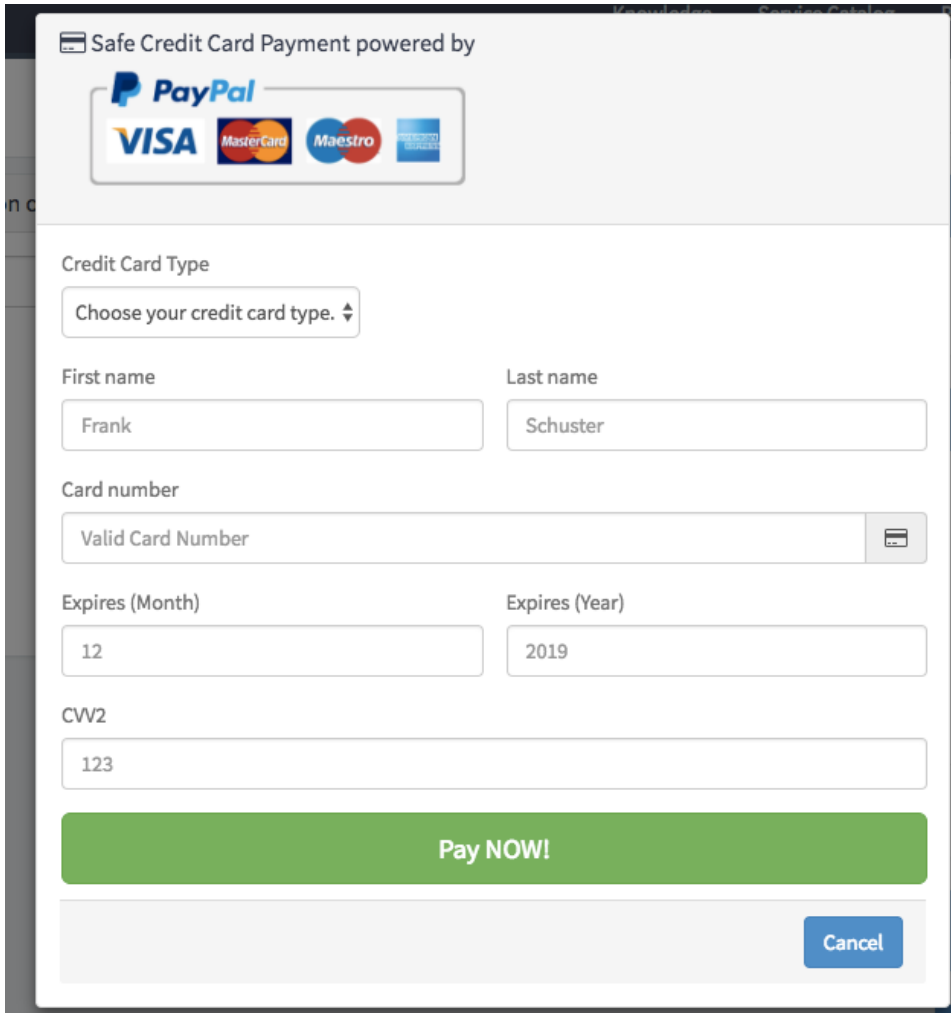
The Client Script also initializes a **loadingIndicator** variable, so we can display a loading spinner, while the server is working (for more on creating custom loading spinners read the following [Community post](#)).

The **openPaymentModal** function uses **UI Bootstrap** to open up the **paymentModal** content, related to the Widget.

Lab 2.3 Writing the Client Script

Lab Validation:

Go back to your Ticket Form for that Request or refresh the page. When you click **PayNow**, you should now see the following screen:



The screenshot shows a payment form titled "Safe Credit Card Payment powered by". Below the title are logos for PayPal, VISA, MasterCard, and Maestro. The form contains the following fields and controls:

- Credit Card Type:** A dropdown menu with the text "Choose your credit card type." and a small upward/downward arrow icon.
- First name:** A text input field containing the value "Frank".
- Last name:** A text input field containing the value "Schuster".
- Card number:** A text input field containing the placeholder text "Valid Card Number". To the right of the field is a small icon of a credit card.
- Expires (Month):** A text input field containing the value "12".
- Expires (Year):** A text input field containing the value "2019".
- CW2:** A text input field containing the value "123".
- Pay NOW!:** A large green button with the text "Pay NOW!" in white.
- Cancel:** A blue button with the text "Cancel" in white, located at the bottom right of the form.

Lab Goal

This lab introduces you to the concept of Server Scripts in Service Portal widgets. In those you can query tables (by using GlideRecord), create integrations and update your **data** object for further use on the client and within the HTML.

Write the Server Script

The Server Script can be found in the **Git repository** and is named:

- **Widget-Server.js**
 1. Check the **Server Script** checkbox in order for the Server script section to show up. Same as for the client script: if needed add the other sections, but for this section you might want to only show the Server Script section.
 2. Copy & Paste the content from the file into the **Server Script** part and click **Save**.
 3. Should you use your own PayPal account, now is the time to substitute the PayPal Client ID & Secret within the **getPayPalToken()** function. In Line 106, replace the first parameter with your Client ID and the second parameter with your Secret.
 4. The last step is to revert the changes in the HTML. Remove the standard <div> element in line 3 and comment the ng-if <div> back in (remove the <!-- and --> characters), click **Save** when you're done.

Explanation:

By utilizing the **\$sp.getRecord()** function you get access to the actual GlideRecord that you are working on now (which is the Requested Item). We do need the sys_id & the table for further processing of the GlideRecord.

The next block in the Server Script will evaluate the **stage field of the Requested Item record** and set the **showPayment** variable to false. **showPayment** will be accessed by the Angular directive in order to determine if the widget should be displayed or not.

Lab 2.4 Writing the Server Script

Reminder: if you are not using the backend update set your orderable Catalog Item needs to have a Workflow assigned that also includes that specific stage! Value should be **payment_requested**, otherwise simply adjust the Server Script to your matching value.

input is a ServiceNow variable that contains everything related to the data object that you filled with your HTML input field values within the Angular controller and/or HTML.

The first time you load the page, the **input** variable will be empty/undefined. You also do not want to trigger the integration onLoad of the page, which is why you check for input **AND** gr. Only if your input is present and you are actually having a GlideRecord object, then you want to proceed.

Now that you have your input defined you can start preparing your payment **JSON** object for the PayPal request. You also utilize the GlideRecord object to retrieve the price, currency code and the company of the **Requested for** user.

Once the PayPal object is prepared according to the PayPal standards, we can finally the PayPal API.

Before you can make the Payment call you have to get an OAuth 2.0 token from PayPal that will authorize you to send a Payment call.

PayPal Developer API: How PayPal uses OAuth 2.0

(<https://developer.paypal.com/docs/integration/direct/paypal-oauth2/>)

The token is embedded in the PayPal response so you extract it via String manipulation. A better way to do this would be using Regular Expressions (Bonus points awarded for doing so 😊)

The obtained token will now be used in the request to the PayPal credit card API. Wrap the parts of building the JSON object and sending your request into one **try/catch** block, to catch any integration errors.

Lab Goal

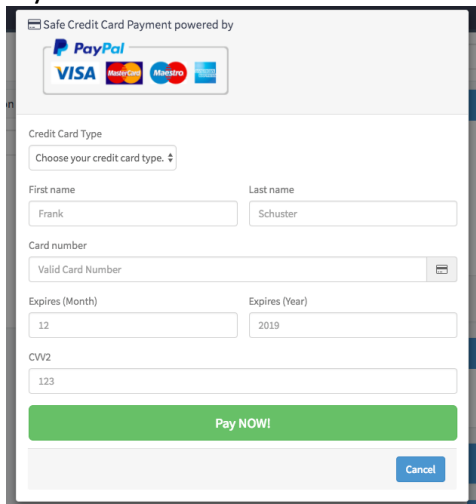
This lab introduces will guide you through testing the PayPal integration within the Service Portal.

Testing the PayPal integration

To test the integration, you will have to repeat the steps from **Lab 2.1 (Adding the Widget to the Requested Item Page & Lab Validation)**.

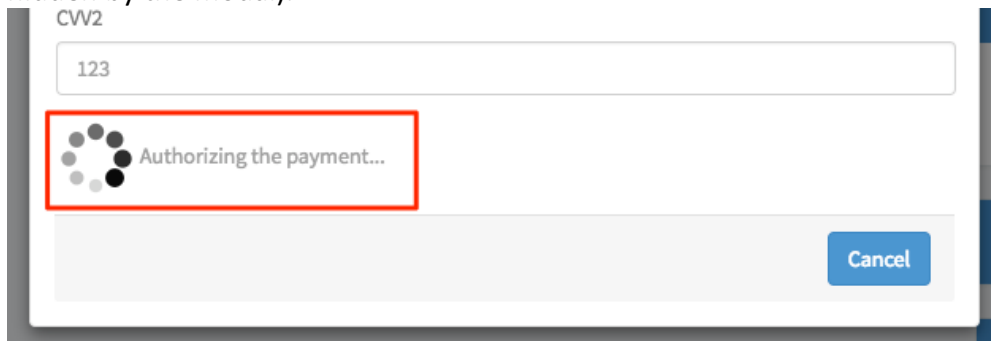
Lab 3.1 Testing the Integration

1. Refresh the Ticket form page and click on **PayNow**, so that the modal will come up and you have to enter data in the modal. In order to test it successfully you will have to utilize the PayPal Sandbox API data.

A screenshot of a PayPal payment modal. At the top, it says "Safe Credit Card Payment powered by" followed by the PayPal logo and logos for VISA, MasterCard, American Express, and Discover. Below this is a "Credit Card Type" dropdown menu with the text "Choose your credit card type." and a small upward arrow. There are two input fields for "First name" (containing "Frank") and "Last name" (containing "Schuster"). Below these is a "Card number" input field with the placeholder "Valid Card Number" and a small card icon. There are two input fields for "Expires (Month)" (containing "12") and "Expires (Year)" (containing "2019"). Below these is a "CVV2" input field with the placeholder "123". At the bottom, there is a large green button labeled "Pay NOW!" and a smaller blue button labeled "Cancel".

2. The demo data is the following:
Credit Card Type: Pick **VISA**
First name: Betsy
Last name: Buyer
Number: 4485187963679205
Expire Month: 11
Expire Year: 2018
CVV (optional in that use case): 874

3. Click the **Pay NOW!** button. You will notice, that the **paymentModal** also includes an additional loading spinner (in addition to the OOTB three dots in the portal header, that are hidden by the modal).



4. It will take a few seconds for the request to complete. If the request fails an error box will appear right in the modal. If everything was completed successfully the modal will close and add additional comments will be added to the Requested Item. A success message will be displayed for 3 seconds within the modal.

Safe Credit Card Payment powered by

PayPal VISA MasterCard Maestro

Warning! Payment couldn't be completed successfully. Please try again. ✕

Credit Card Type
VISA

First name: Betsy
Last name: Buyer

Card number: 4485187963679204

Expires (Month): 11
Expires (Year): 2018

CW2: 123

Pay NOW!

Cancel

Safe Credit Card Payment powered by

PayPal VISA MasterCard Maestro

Thank you! We have received your payment, this window will automatically close in 3 ✕ seconds.

Credit Card Type
VISA

First name: Last name:

Summary

These labs guided you through the creation of a new Service Portal widget.

You utilized AngularJS & Bootstrap within the HTML & Client Script parts of the widget.

You also learned how easy it is to call an external API via REST out of a Service Portal widget and to process a REST response within the Server Script part.

Within the Server Script you updated the Requested Item record with the Payment date/time, which could drive a Workflow (“Wait for Payment Requested date/time”).

Now you could create a little application to manage the PayPal integration (e.g. Properties for ClientID & Secret, Adding the token to a table and retrieving it when needed instead of calling the OAuth API every time etc.). The loading indicator is also tied to all activities in Portal, it would be more appropriate to tie the loading indicator to the actual transaction (i.e. only display it between start of the REST call <> end of the REST call).