

# Lab Guide

## Creating A Custom Service Portal Widget: PayPal Integration

Frank Schuster

This  
Page  
Intentionally  
Left  
Blank

# Lab Goal

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

## Lab 1.1 Creating a Service Portal widget

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/frankschuster/service-portal-hacklab>
- ServicePortalLab-Backend.xml (already deployed to your Lab Instance)
- Widget-HTML.html
- Widget-ng-template.html (Angular ng-template > Related List on the Widget record)
- Widget-Client.js
- Widget-Server.js
- ServicePortalLab-Widget.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)

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)

## Step 1: Prepare the Backend

Before you start building your widget you need to prepare the ServiceNow backend for the input later on. **We have already applied that update set on the HackLab instance, so you will not have to go through the following 4 steps.**

1. To do that, you will simply have to apply the Update Set “ServicePortalLab-Widget.xml”
2. Navigate to **System Update Sets > Retrieved Update Sets**
3. Click **Import Update Set from XML** and pick the downloaded record
4. Preview & Commit the Update Set

## Step 2: Configuring PayPal

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.

1. Navigate to: <https://developer.paypal.com/>
2. Click **Log In** and use your normal 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 contains the following menu items: **Dashboard**, **My Apps & Credentials** (highlighted with a red box), **My Account**, **Sandbox** (with sub-items: Accounts, Notifications, API Calls, IPN Simulator, Webhooks Events), **Mock** (with sub-item: Webhooks Simulator), and **Live** (with sub-items: API Calls, Webhooks Events). The top navigation bar includes the PayPal Developer logo, links for Docs, Reference, and Support, a search bar, and buttons for Dashboard and Log Out. The main content area is titled 'REST API apps' and includes a note about live transaction eligibility. A 'Create App' button is highlighted with a red box. Below it 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.

PayPal Developer Docs Reference Support Search

Dashboard Log Out

**Dashboard**

**My Apps & Credentials**

My Account

**Sandbox**

Accounts

Notifications

API Calls

IPN Simulator

Webhooks Events

**Mock**

Webhooks Simulator

**Live**

API Calls

Webhooks Events

### REST API apps

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

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

Create App

App Name	Type	Actions
<a href="#">servicenow-hacklab</a>	REST	


### NVP/SOAP API apps

Legacy PayPal apps (e.g., Adaptive Payments, Adaptive Accounts, Adaptive Permissions) are now called NVP/SOAP API apps.

Manage NVP/SOAP API apps

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.


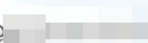
 Features available for live transactions are listed in your [account eligibility](#).

### 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.

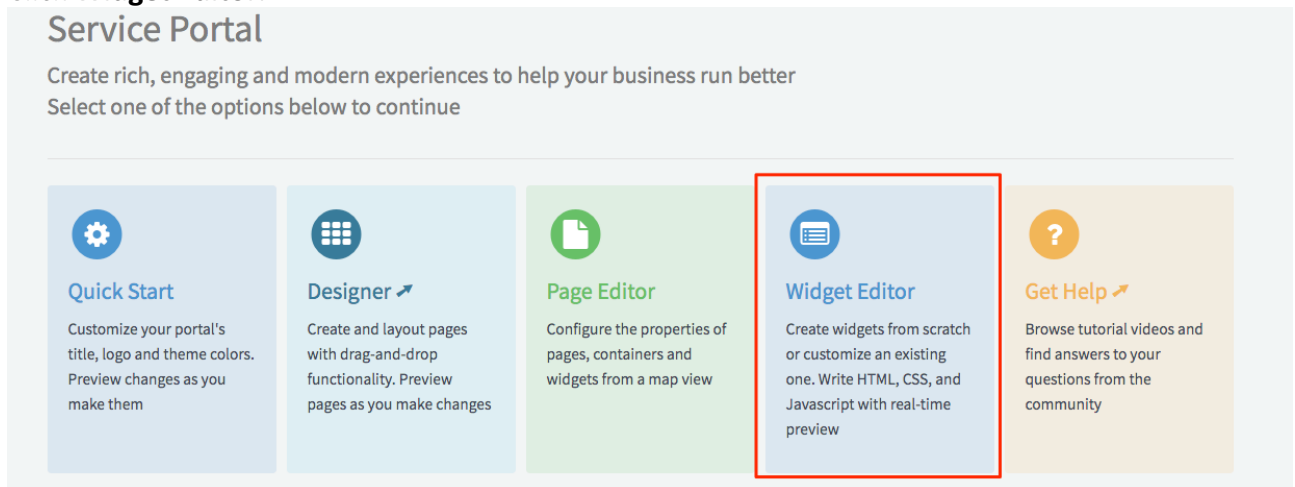
[Save](#) [Cancel](#)



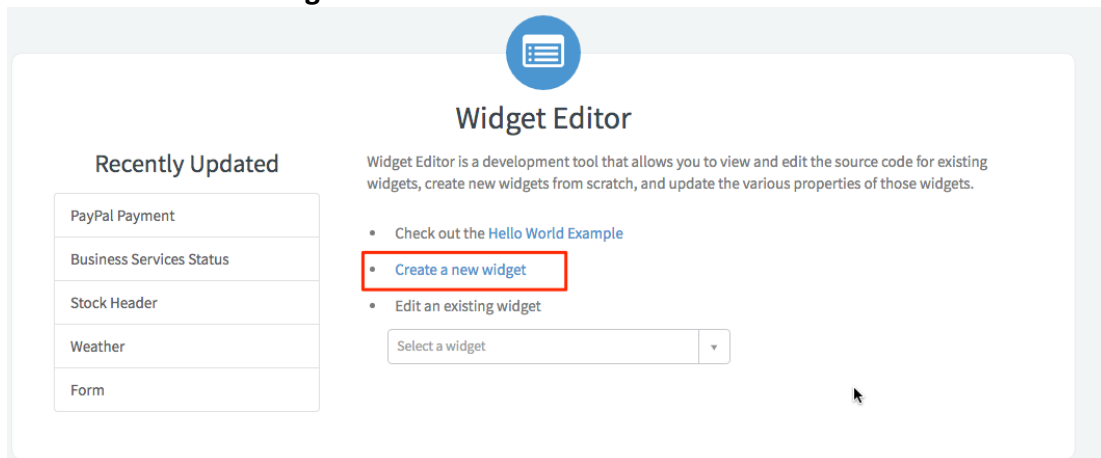
## Creating the Service Portal widget

This section guides you through the process of creating a new Service Portal widget including the HTML, Client Script & Server script parts.

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/frankschuster/service-portal-hacklab>
- Files for this lab:
  - **Widget-HTML.html** (Widget HTML)
  - **Widget-ng-template.html** (Angular ng-template)

## Lab 2.1 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**.

#### Explanation:

**data.showPayment** is a variable that is set by the server script (**Lab 4.1**) – 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/frankschuster/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. Navigate to **System UI > Images** and create two new image records (categorize them as **Service Portal**):

- accepted-cards.png
- portal-loading-spinner.svg

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.

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**.

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.

You do **not** need to create a form right now, since you leverage the content from the example HTML in the Github Repo.

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]

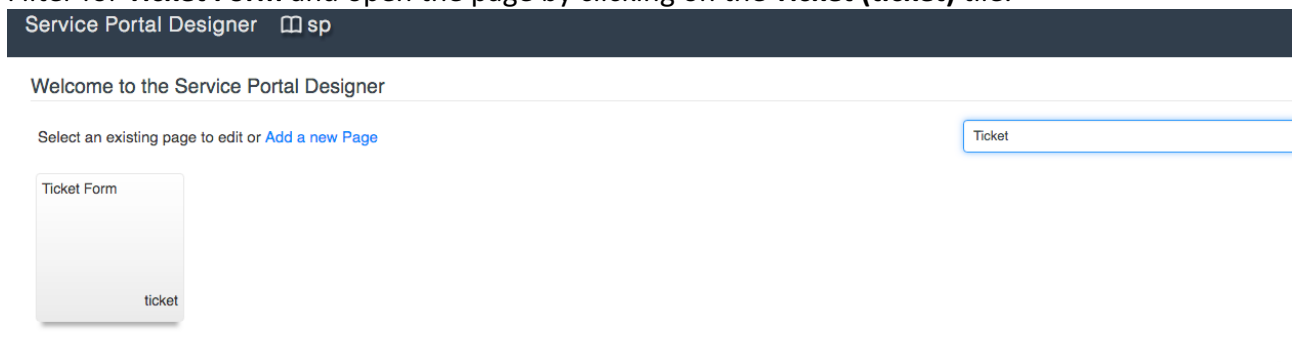
The **optional** field for your integration is:

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

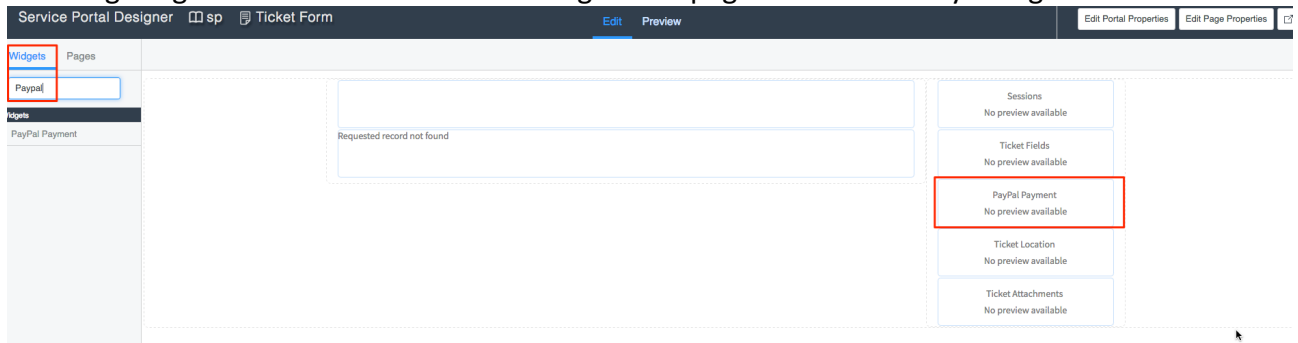
When writing your own HTML do not forget to define an **ng-model** attribute for each input field, since we have to bind each HTML element to the **data** object, in order to being able to use it in the Server Script later.

## 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.

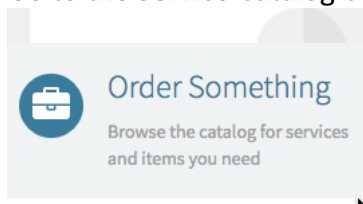


4. Filter for the PayPal widget by using the search field when **Widgets** is selected. Drag & Drop the widget right below the **Ticket Fields** widget. The page is automatically being saved.

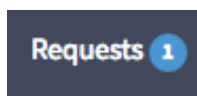
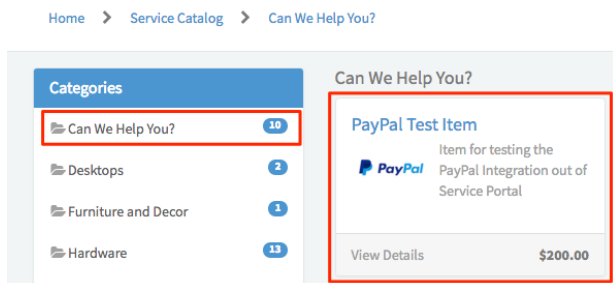


5. In order to test that change you made to the form, you will have to order the **PayPal Test Item**. Go to the Service Portal index page.  
<https://yourInstance.service-now.com/sp>

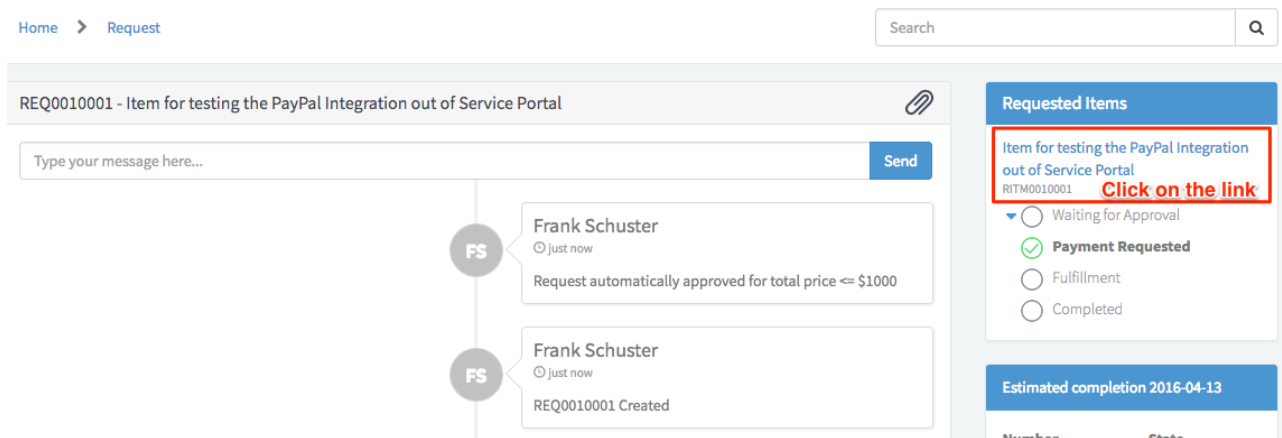
6. Go to the Service Catalog by clicking on **Order Something**.



7. 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.



8. 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. Click on the **PayNow** button to show the Modal.

home > Requested Item

Search

Conversation

Type your message here... Send

FS

Frank Schuster  
4m ago  
RITM0010001 Created

Start

Your request has been submitted

Number	State
RITM0010001	Open
Priority	Created
4 - Low	4m ago
Updated	Price
4m ago	\$200.00

Tickets are picked up within  
4 hours (M-F 9-5)

Make a Payment

PayNow

Location

Attachments

Drop files here

10. The payment modal should look like the following if you have used the code from the repository – if not simply make sure you have the required fields for the API call:

Safe Credit Card Payment powered by

PayPal

VISA MasterCard Maestro

Credit Card Type

Choose your credit card type. ▾

First name

Frank

Last name

Schuster

Card number

Valid Card Number

Expires (Month)

12

Expires (Year)

2019

CVV2

123

Pay NOW!

Cancel



# 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 3.1 Writing the Client Script

# 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**.

### 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.

**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.

## Lab 4.1 Writing the Server Script

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.

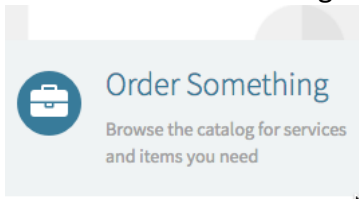
# Lab Goal

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

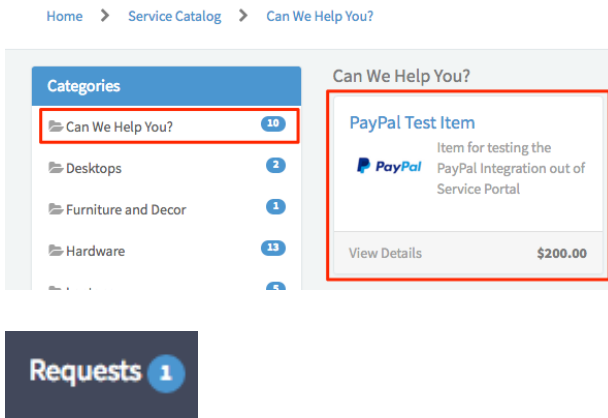
## Testing the PayPal integration

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)**.

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



4. 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.



## Lab 5.1 Testing the Integration

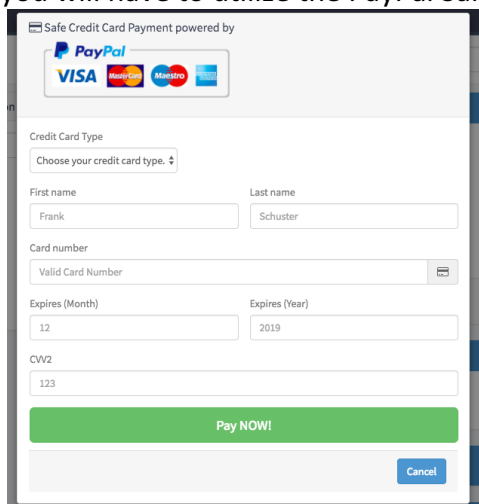
- Click on the Request and drill down into the actual Requested Item by clicking on the name of the item in the right hand side **Requested Items** container.

The screenshot shows the ServiceNow Request page. At the top, there's a breadcrumb "Home > Request" and a search bar. Below this, the request title is "REQ0010001 - Item for testing the PayPal Integration out of Service Portal". A message input field with a "Send" button is present. The conversation history shows two messages from "Frank Schuster" (FS) at "just now": "Request automatically approved for total price <= \$1000" and "REQ0010001 Created". On the right, the "Requested Items" section lists the item "Item for testing the PayPal Integration out of Service Portal" with ID "RITM0010001". It has a status of "Payment Requested" (indicated by a green checkmark) and an "Estimated completion 2016-04-13". Below this is a table with columns "Number" and "State".

- On the right side you will find your **PayNow** widget. If your Catalog Item has Variables they will appear as **Options** – minimize those to see the Widget. Click on the button to show the Modal.

The screenshot shows the ServiceNow Request Item page. At the top, there's a breadcrumb "Home > Requested Item" and a search bar. Below this, the conversation history shows a message from "Frank Schuster" (FS) at "4m ago" saying "RITM0010001 Created". A "Start" button is visible. On the right, the "Your request has been submitted" section shows details: Number "RITM0010001", State "Open", Priority "4 - Low", Created "4m ago", Updated "4m ago", and Price "\$200.00". Below this is a note: "Tickets are picked up within 4 hours (M-F 9-5)". The "Make a Payment" section is highlighted with a red box, showing a "PayNow" button. Below this are sections for "Location" and "Attachments" with a "Drop files here" area.

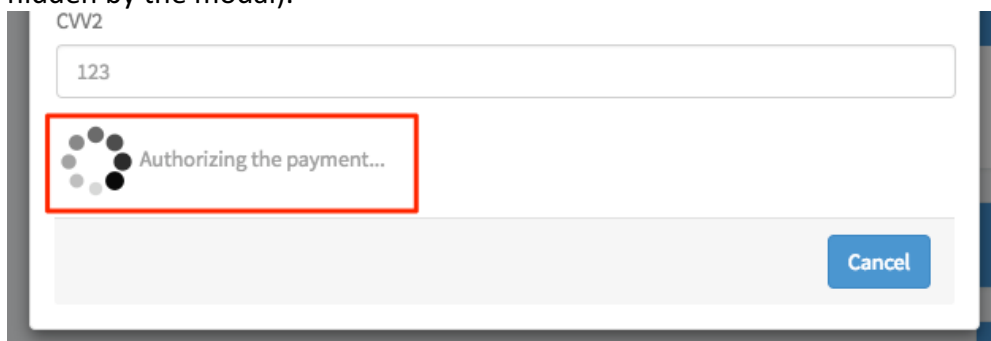
7. The modal will come up and you have to enter data in the modal. In order to test it successfully you will have to utilize the PayPal Sandbox API data.



The screenshot shows a 'Safe Credit Card Payment powered by PayPal' modal. It features the PayPal logo and logos for VISA, MasterCard, and American Express. The form includes a 'Credit Card Type' dropdown menu, 'First name' and 'Last name' text boxes, a 'Card number' text box with a 'Valid Card Number' label, 'Expires (Month)' and 'Expires (Year)' text boxes, and a 'CVV2' text box. A large green 'Pay NOW!' button is at the bottom, with a smaller blue 'Cancel' button to its right. The modal is set against a dark background.

8. 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

9. 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).



This screenshot shows a close-up of the payment modal during a transaction. The 'CVV2' text box contains the value '123'. Below it, a red rectangular box highlights a loading spinner icon (a circle of dots) and the text 'Authorizing the payment...'. A blue 'Cancel' button is visible at the bottom right of the modal.

10. 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.

A screenshot of a 'Safe Credit Card Payment powered by PayPal' modal. At the top, it displays the PayPal logo and logos for VISA, MasterCard, Maestro, and American Express. Below this, a red warning box with a close icon (X) contains the text: 'Warning! Payment couldn't be completed successfully. Please try again.' The form fields are as follows: 'Credit Card Type' is a dropdown menu showing 'VISA'; 'First name' is a text box with 'Betsy'; 'Last name' is a text box with 'Buyer'; 'Card number' is a text box with '4485187963679204'; 'Expires (Month)' is a text box with '11'; 'Expires (Year)' is a text box with '2018'; and 'CVV2' is a text box with '123'. At the bottom, there is a large green button labeled 'Pay NOW!' and a smaller blue button labeled 'Cancel'.

A screenshot of the same 'Safe Credit Card Payment powered by PayPal' modal, but now showing a success message. A green box with a close icon (X) contains the text: 'Thank you! We have received your payment, this window will automatically close in 3 seconds.' The form fields are partially visible below: 'Credit Card Type' is a dropdown menu showing 'VISA', and 'First name' and 'Last name' labels are visible.

## 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).