Modules implementation
v 1.0.0

Modules in Shopizer

Shopizer uses different modules as part of the solution in order to allow multiple different implementations for a same functionality. There are 2 main objectives in module concept:

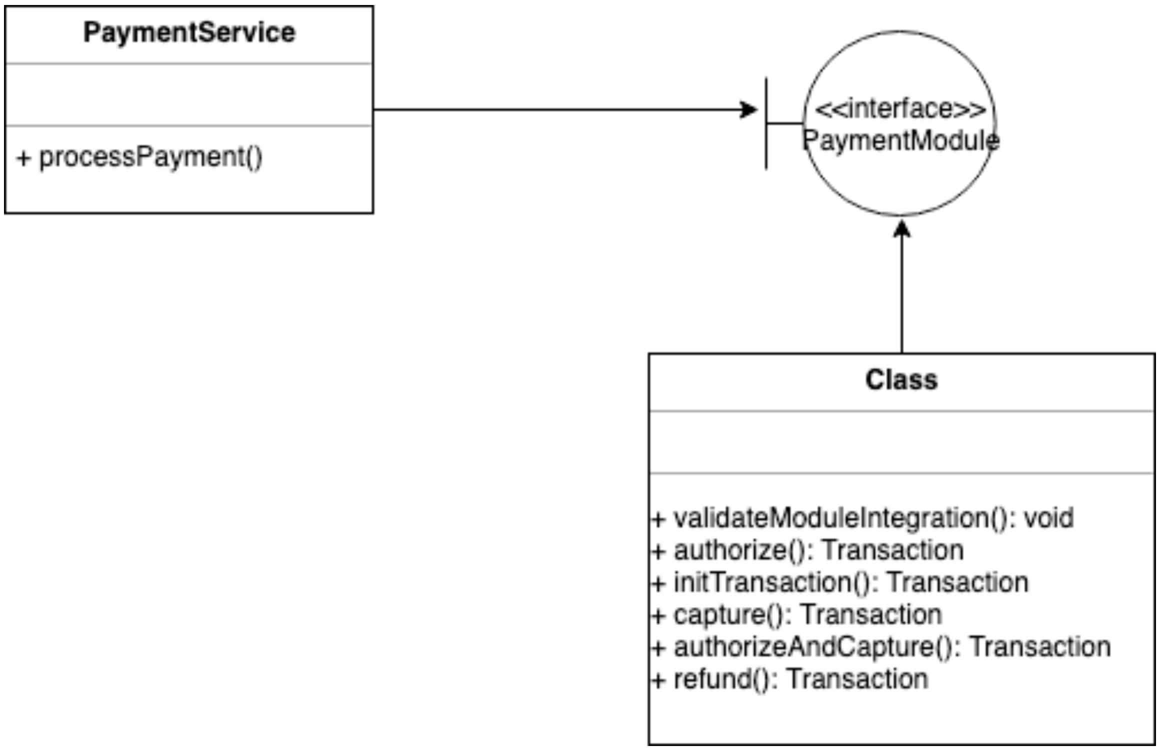
- Have a functionality having multiple possible implementation

  This is possible for functionality such as 'payment' or 'shipping' where a multitude of payment or shipping modules may be offered to a customer (example Stripe, Paypal…) or a decision made at runtime requiring to use a specific module (example pricing modules).

- Have the possibility to change a specific implementation without changing the interface.

  This concept allows changing from one implementation to another, for example email implementation to use javax.mail or AWS SES.

Payment modules


PaymentService
+ processPayment()
<<interface>>
PaymentModule
Class
+ validateModuleIntegration(): void
+ authorize(): Transaction
+ initTransaction(): Transaction
+ capture(): Transaction
+ authorizeAndCapture(): Transaction
+ refund(): Transaction

com.salesmanager.core.business.services.paymentsPaymentServiceImpl.processPayment() does the logic of preparing and validating payment to be processed during checkout. It then invokes selected PaymentModule implementation corresponding to the payment option user selected to use.

Creation of a new PaymentModule

A payment gateway is a merchant service provided by payment service providers that authorizes credit card or direct payments processing for e-businesses, online retailers, bricks and clicks, or traditional brick and mortar. A payment gateway needs to be configured in Shopizer in order to process online payments.

There are many payment gateway solutions available for integrating with Shopizer. Our recommendation is to look for PCI compliant Hosted Tokenization (HT) solution.

HT is a solution for online e-commerce merchants that do not wish to handle credit card numbers directly on their websites and uses token technology as a correspondence alias between the e-commerce site and the service provider instead of the credit card number. Shopizer does integrate with HT PCI solutions such as Stripe, Braintree and Paypal.

Most payment service provides those apis:

- **authorize**: Transaction is only authorized and not charged to the card. This is generally used when selling hard goods like Amazon. The card is verified and the transaction is authorized for making sure there is enough credit available. When using the authorize method then the transaction has to be captured just before shipping the item.
- **capture**: This method is used when an authorization transaction has been made on the card before. Capture transaction will charge the amount on the credit card.
- **authorize** and capture: This method is used when authorization and capture happen during the same transaction. A common use case for this is for selling digital products having to charge the amount before sending download link to customer. This can be also used when selling hard goods, it's a dynamic configuration from Shopizer immediately available after changing charge function.
- **refund**: This methods refunds entirely or partially a previous transaction.

Implementing your own payment module

When you need to create a new integration with another payment gateway than the one already available in Shopizer you need to perform the following steps:

Selection of your payment gateway

A payment gateway will require opening a service account and link that to your bank account. Usually they have a test and a production environment and will invite you to perform all possible tests with their test environment before deciding to use the production environment. They also have an api that you will have to connect with your payment module. That api will require authentication using one to multiple keys that you will have to be saved in Shopizer when this payment method is configured.

- Define a payment service as a module

Once the selection of a payment gateway is done the next step is to define a payment module corresponding to this payment service. Payment methods are defined in shopizer/sm-core/src/main/resources/reference/integrationmodules.json in json format. A new payment service will require the following fields:

```
{
          "module": "PAYMENT",
          "code": "stripe",
          "type":"creditcard",
          "version":"1.0.1",
          "regions": ["US","CA","GB","AU","FI","DK","IE","NO","SE"],
          "image":"stripe.png",
          "configuration":[{"env":"TEST","scheme":"https","host":"www.stripe.com","port":"443","uri":"/"},{"env":"PROD"
,"scheme":"https","host":"www.stripe.com","port":"443","uri":"/"}]
}
```

Property	Description
module	PAYMENT
code	A unique code that identifies the new payment service
type	CREDITCARD("creditcard"), FREE("free"), COD("cod"), MONEYORDER("moneyorder"), PAYPAL("paypal")
version	Refers to payment service api version
regions	Country (ISO code 2) in which this payment method is available
image	Image of payment service Needs to be added to shopizer/sm-shop/src/main/webapp/resources/img/payment
configuration	An array that contains scheme: http | https *host: <server url> *port: <port> *uri: <uri> exemple /payment *env: TEST | PROD config1: additional infos config2: additional infos

Here is a json template file example

```
{
          "module": "PAYMENT",
          "code": "<YOUR CODE>",
          "type":"creditcard",
          "version":"1.0",
          "regions": ["US","CA","GB","AU","FI","DK","IE","NO","SE"],
          "image":"<SERVICE IMAGE>.<EXTENSION>",
          "configuration":[{"env":"TEST","scheme":"https","host":"<HOST NAME TEST
ENVIRONMENT>","port":"<PORT>","uri":"/"},{"env":"PROD","scheme":"https","host":"<HOST NAME PRODUCTION
ENVIRONMENT>"," ","port":"<PORT","uri":"/"}]
}
```

Another example

```
{
          "module": "PAYMENT",
          "code": "mytest",
          "type": "creditcard",
          "version": "104.0",
          "regions": ["*"],
          "image": "icon-paypal.png",
          "configuration": [{
                    "env": "TEST",
                    "scheme": "https",
                    "host": "www.sandbox.mytest.com",
                    "port": "80",
                    "uri": "/cgi-bin/webscr?cmd=&token=",
                    "config1": "beta-version"
          }, {
                    "env": "PROD",
                    "scheme": "https",
                    "host": "www.production.mytest.com",
                    "port": "80",
                    "uri": "/cgi-bin/webscr?cmd=&token=",
                    "config1": "RELEASE"
          }]
}
```

Once your template file is complete with payment service integration details

Validate the json format with json lint tool: https://jsonlint.com/
If the json format is valid then it has to be inserted in Shopizer database as module metadata for this new service.

This is possible by posting the request to system create module api using curl, javascript or with your favorite POST tool.

Example with cUrl

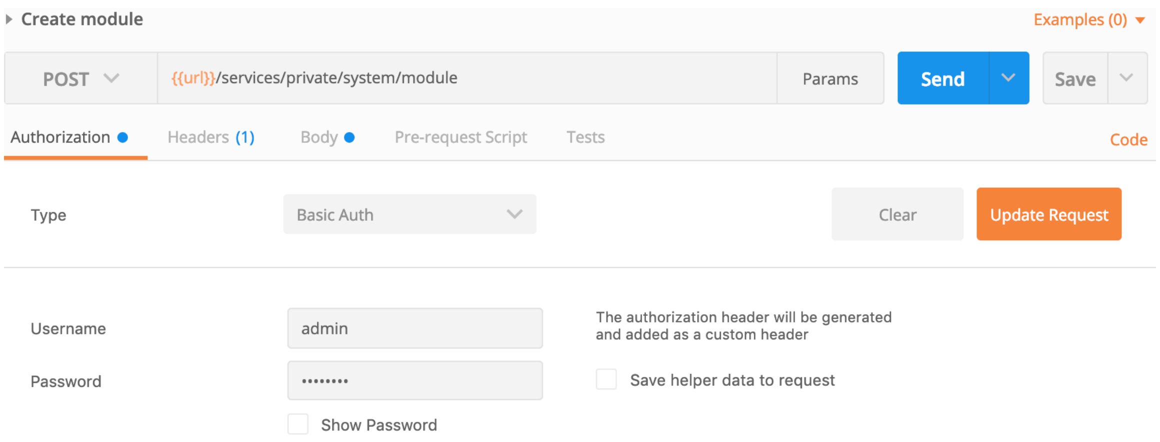
```
curl -X POST \
  http://localhost:8080/services/private/system/module \
  -H 'authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
  -H 'cache-control: no-cache' \
-d '{
          "module": "PAYMENT",
          "code": "mytest",
          "type": "creditcard",
          "version": "104.0",
          "regions": ["*"],
          "image": "icon-paypal.png",
          "configuration": [{
                    "env": "TEST",
                    "scheme": "https",
			"host": "www.sandbox.mytest.com",
			"port": "80",
			"uri": "/cgi-bin/webscr?cmd=&token=",
			"config1": "beta-version"
		}, {
			"env": "PROD",
			"scheme": "https",
			"host": "www.production.mytest.com",
			"port": "80",
			"uri": "/cgi-bin/webscr?cmd=&token=",
			"config1": "RELEASE"
		}]
}'
```

Example with javascript

```
var settings = {
  "async": true,
  "crossDomain": true,
  "url": "http://localhost:8080/services/private/system/module",
  "method": "POST",
  "headers": {
    "authorization": "Basic YWRtaW46cGFzc3dvcmQ=",
    "cache-control": "no-cache"
   },
  "data": "{\"module\": \"PAYMENT\",\"code\": \"mytest\",\"type\": \"creditcard\",\"version\": \"104.0\",\"regions\":
[\"*\"],\"image\": \"icon-paypal.png\",\"configuration\": [{\"env\": \"TEST\",\"scheme\": \"https\",\"host\":
\"www.sandbox.mytest.com\",\"port\": \"80\",\"uri\": \"/cgi-bin/webscr?cmd=&token=\",\"config1\": \"beta-version\"},
{\"env\": \"PROD\",\"scheme\": \"https\",\"host\": \"www.production.mytest.com\",\"port\": \"80\",\"uri\": \"/cgi-
bin/webscr?cmd=&token=\",\"config1\": \"RELEASE\"}]\n}"
}

$.ajax(settings).done(function (response) {
  console.log(response);
});
```

Example using postman



{{url}}/services/private/system/module

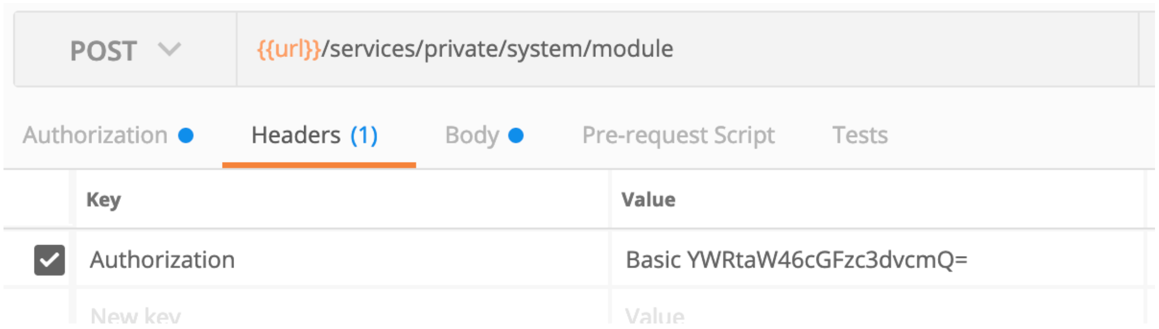
Type: Basic Auth

Username: admin

Password: ••••••••

The authorization header will be generated and added as a custom header

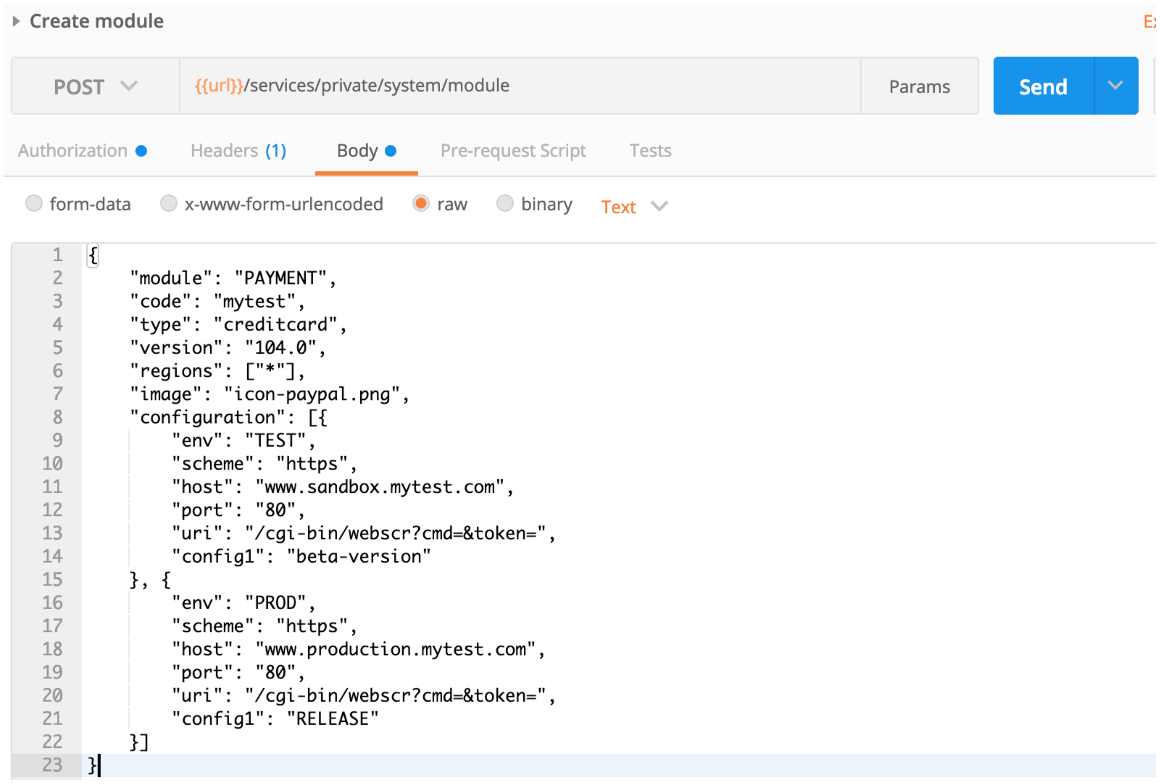
Username and password are required to perform this operation. Any user in Shopizer with admin or api role can perform this. Default username in shopizer is admin and default password is password. Authentication for this service is Basic and requires base64 encoding.


POST {{url}}/services/private/system/module

Authorization ● Headers (1) Body ● Pre-request Script Tests

Key	Value
Authorization	Basic YWRtaW46cGFzc3dvcmQ=
New key	Value

Request body contains json module json block


Create module

POST {{url}}/services/private/system/module

Params Send

Authorization ● Headers (1) Body ● Pre-request Script Tests

form-data x-www-form-urlencoded [✓ raw] binary Text

```
{
    "module": "PAYMENT",
    "code": "mytest",
    "type": "creditcard",
    "version": "104.0",
    "regions": ["*"],
    "image": "icon-paypal.png",
    "configuration": [{
        "env": "TEST",
        "scheme": "https",
        "host": "www.sandbox.mytest.com",
        "port": "80",
        "uri": "/cgi-bin/webscr?cmd=&token=",
        "config1": "beta-version"
    }, {
        "env": "PROD",
        "scheme": "https",
        "host": "www.production.mytest.com",
        "port": "80",
        "uri": "/cgi-bin/webscr?cmd=&token=",
        "config1": "RELEASE"
    }]
}
```

- Implement code for this new module

Next task consist of creating your new payment module implementation logic. Payment modules can be packaged in Shopizer or as a standalone maven dependency. For creating a dependency project it requires maven skills and ability to host your module on Nexus sonatype or other Maven central repository. Creating a module packaged in Shopizer is the easiest option, which basically requires the creation of your implementation class in com.salesmanager.core.business.modules.integration.payment.impl

Class name must be your module code in camel case, so for instance if you created **mytest** as module code then your implementation should be **MyTestPayment.class**

```
package com.salesmanager.core.business.modules.integration.payment.impl;

import java.math.BigDecimal;
import java.util.List;
import com.salesmanager.core.model.customer.Customer;
import com.salesmanager.core.model.merchant.MerchantStore;
import com.salesmanager.core.model.order.Order;
import com.salesmanager.core.model.payments.Payment;
import com.salesmanager.core.model.payments.Transaction;
import com.salesmanager.core.model.shoppingcart.ShoppingCartItem;
import com.salesmanager.core.model.system.IntegrationConfiguration;
import com.salesmanager.core.model.system.IntegrationModule;
import com.salesmanager.core.modules.integration.IntegrationException;
import com.salesmanager.core.modules.integration.payment.model.PaymentModule;

public class MyTestPayment implements PaymentModule {

  @Override
  public void validateModuleConfiguration(IntegrationConfiguration integrationConfiguration,
     MerchantStore store) throws IntegrationException {
  }

  @Override
  public Transaction initTransaction(MerchantStore store, Customer customer, BigDecimal amount,
     Payment payment, IntegrationConfiguration configuration, IntegrationModule module)
     throws IntegrationException {
          return null;
  }

  @Override
  public Transaction authorize(MerchantStore store, Customer customer, List<ShoppingCartItem> items,
     BigDecimal amount, Payment payment, IntegrationConfiguration configuration,
     IntegrationModule module) throws IntegrationException {
          return null;
  }

  @Override
  public Transaction capture(MerchantStore store, Customer customer, Order order,
     Transaction capturableTransaction, IntegrationConfiguration configuration,
     IntegrationModule module) throws IntegrationException {
          return null;
  }

  @Override
  public Transaction authorizeAndCapture(MerchantStore store, Customer customer,
     List<ShoppingCartItem> items, BigDecimal amount, Payment payment,
     IntegrationConfiguration configuration, IntegrationModule module)
     throws IntegrationException {
          return null;
  }

  @Override
  public Transaction refund(boolean partial, MerchantStore store, Transaction transaction,
     Order order, BigDecimal amount, IntegrationConfiguration configuration,
     IntegrationModule module) throws IntegrationException {
          return null;
  }
}
```

Since your module class implements PaymentModule you will have to implement a few required methods required during checkout process.

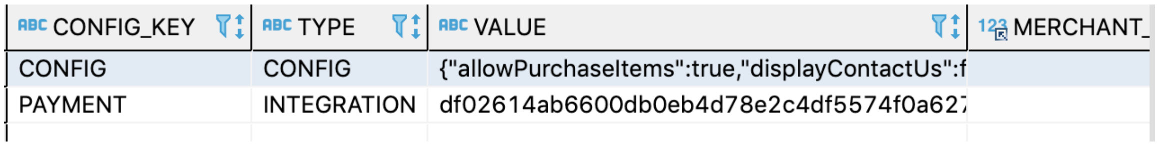
Method	Description
validateModuleConfiguration	This method is invoked when a store user saves

	or updates merchant specific credentials, tokens or other parameters to be sent to the payment service
initTransaction	Invoked before any transaction. This method is useless for most implementation. In that case just return null;
authorize	Integration point between Shopizer's authorize action and payment api authorization endpoint.
capture	Integration point between Shopizer's cature action and payment api authorization endpoint.
authorizeAndCapture	Integration point between Shopizer's authorizeAndCapture action and payment api authorization endpoint.
refund	Integration point between Shopizer's refund action and payment api authorization endpoint.

There are a few PaymentModule samples available here
https://github.com/shopizer-ecommerce/shopizer/tree/master/sm-core/src/main/java/com/salesmanager/core/business/modules/integration/payment/impl

Most of Payment Gateway have unit tests and examples in java that you can plug in each of the ne generated module code.

Merchant credentials used for doing authentication in payment gateway are encrypted using sha encryption when they are saved in the database. From the database point of view, payment information is grouped in a single encrupted database row.



CONFIG_KEY	TYPE	VALUE	MERCHANT_
CONFIG	CONFIG	{"allowPurchaseItems":true,"displayContactUs":f	
PAYMENT	INTEGRATION	df02614ab6600db0eb4d78e2c4df5574f0a627	

Using the administration api and administration tool are the only way to modify payment integration data.

Payment integration data can be retrieved this way

```
IntegrationConfiguration configuration = paymentService.getPaymentConfiguration("mytest", merchantStore);
Map<String,String> keys = configuration.getIntegrationKeys();
```

Payment integration data can be saved this way

```
IntegrationConfiguration configuration = new IntegrationConfiguration();
configuration.setCode("mytest");
configuration.setActive(true);
configuration.setEnvironment(IntegrationConfiguration. TEST_ENVIRONMENT);
configuration.setDefaultSelected(true);
configuration.getIntegrationKeys().add("asecretkey","value");
...

paymentService.savePaymentModuleConfiguration(configuration, merchantStore);
```

- Unit test your new payment module

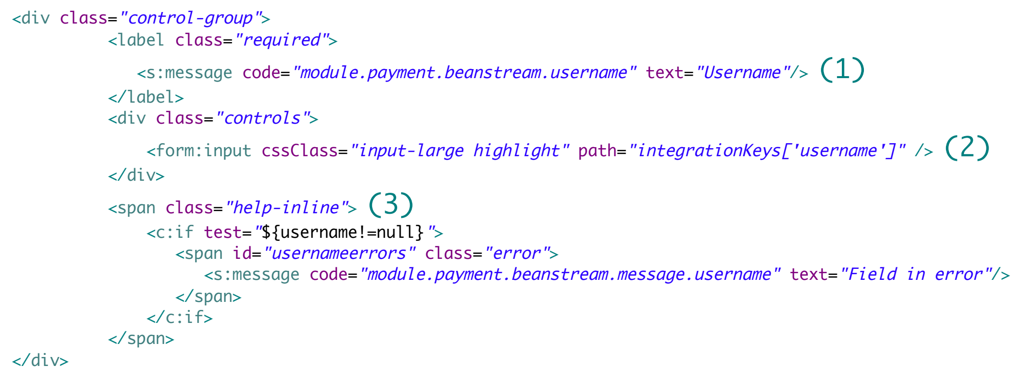
Unit testing a payment module is the best way to test all possibilities and variations of real time scenario. There are existing tests in sm-core component that can be used as starting point for unit testing your module.

- Create an admin ui for being able to use your module

Last milestone consist in creating an administration page for storing module credentials and keys. Shopizer has an existing administration framework for easing the creation of a dedicated page for your new payment module.

Existing payment administration pages can be used as starting point for creating your page. Your new page has to be created in sm-shop/src/main/webapp/pages/admin/payment

Administration page must be named as the payment module code "mytest.jsp". Payment module administration page must contain keys and credentials sections like this:

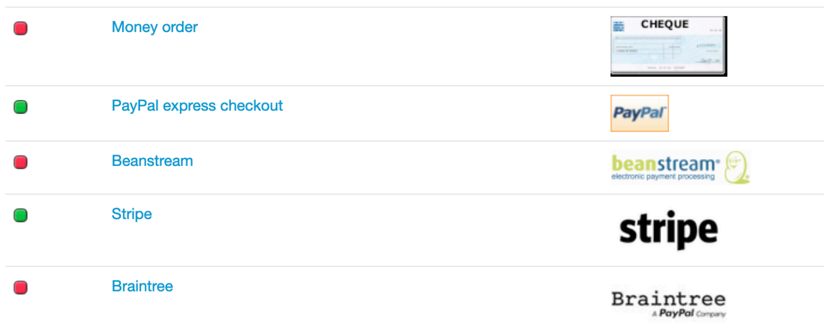


```
<div class="control-group">
          <label class="required">
             <s:message code="module.payment.beanstream.username" text="Username"/> (1)
          </label>
          <div class="controls">
              <form:input cssClass="input-large highlight" path="integrationKeys['username']" /> (2)
          </div>
          <span class="help-inline"> (3)
              <c:if test="${username!=null}">
                 <span id="usernameerrors" class="error">
                    <s:message code="module.payment.beanstream.message.username" text="Field in error"/>
                 </span>
              </c:if>
          </span>
</div>
```

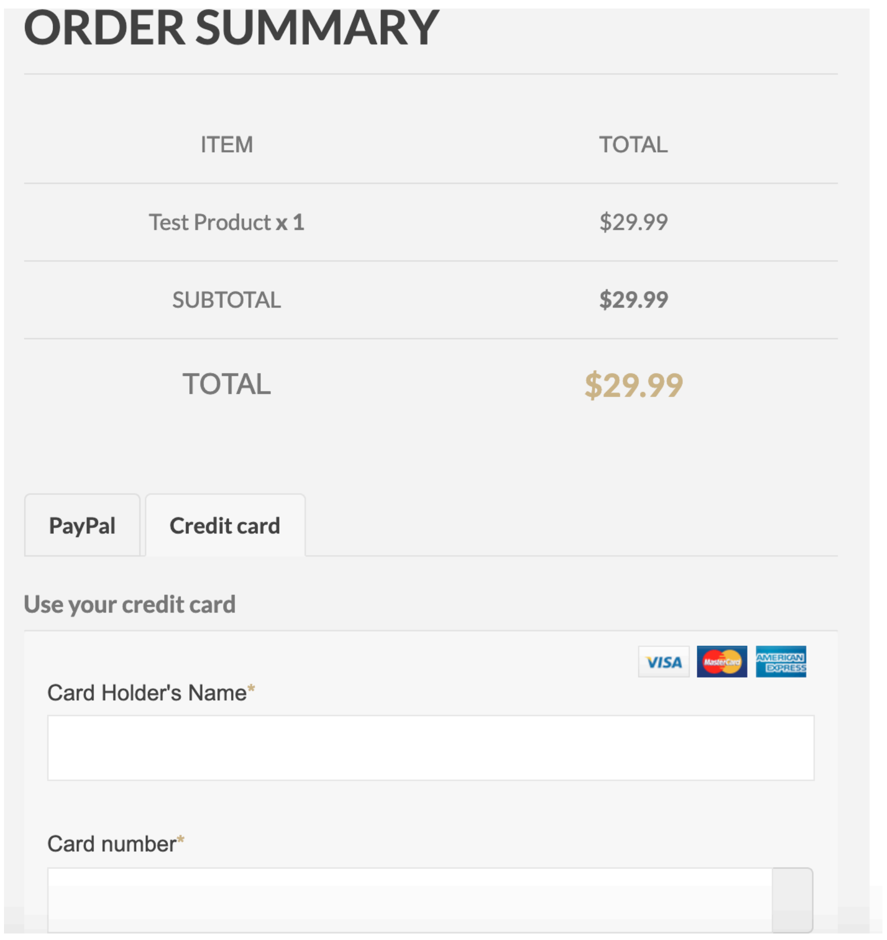
1- Labels must be defined in resource bundles
2- Keys and credentials are defined this way
3- Section for handling empty key / credential submission

Nothing more is required from UI perspective.

Final step is to make sure the module UI is displayed in Payment method list administration page. Make sure keys and credentials can be saved and retrieved and finally perform complete integration test with your new payment module.



🟥	Money order	CHEQUE
🟩	PayPal express checkout	PayPal
🟥	Beanstream	beanstream electronic payment processing
🟩	Stripe	stripe
🟥	Braintree	Braintree A PayPal Company



ORDER SUMMARY

ITEM	TOTAL
Test Product **x 1**	$29.99
SUBTOTAL	**$29.99**
TOTAL	**$29.99**

Use your credit card

Card Holder's Name*

Card number*