- PHP PAYMENTS WITH -

OMNIPAY

- DREW MCLELLAN -

- CONFOO MONTREAL 2017 -

- PHP PAYMENTS WITH -

OMNIPAY

- DREW MCLELLAN -

- CONFOO MONTREAL 2017 -

Hello!

I'm Drew McLellan.

Lead dev on Perch CMS.

@drewm github.com/drewm

Nasty bespoke checkout methods.

Promotions and shipping and carts.

Ecommerce sucks.

Payment gateways really suck!

OMG taxes!

Payment gateways suck.

Payment gateways

Each gateway has its own requirements.

Most are badly designed.

Most are poorly documented.

Most have horrible SDKs.

All are idiosyncratic in some way.

Payment gateways

We all end up building solutions that are tightly coupled to a given gateway's solution.

This makes it really hard to change gateway, to move code from project to project, or add additional payment options.

Who changes payment gateway anyway?

Case study: me

2009: Launched Perch CMS on PayPal

2010: Switched to PayPoint.net with PayPal option

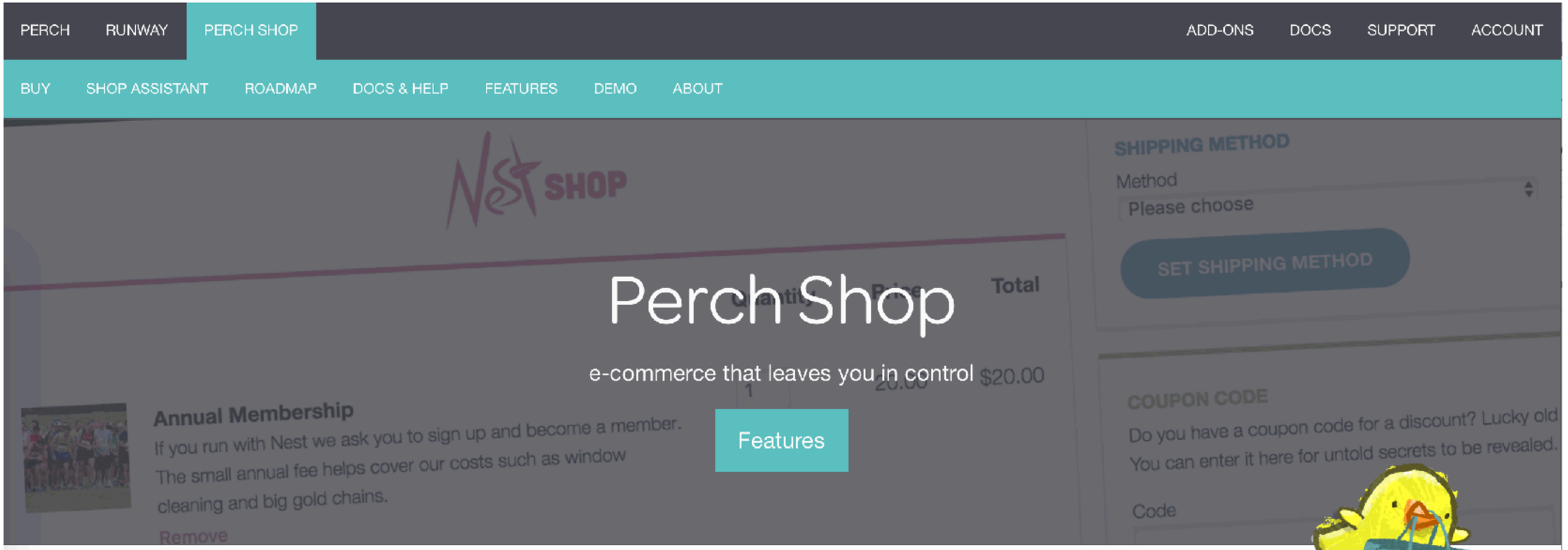
2011: Added our own PayPal integration back

2012: Switched to SagePay + PayPal

2014: Switched to Stripe + PayPal

… and then we built an ecommerce product

Shop add-on for Perch CMS



An e-commerce toolkit

If you have ever tried to build a store you will be familiar with the problem of needing to work within restrictive templates,

Perch Shop

We wanted to support as many payment gateways globally as we could.

We didn't *want* to support any gateways, really.

An abstraction layer sounded like a great idea.

Enter Omnipay.

omnipay.thephpleague.com

Omnipay

Omnipay is a payment processing library for PHP.

It acts as an abstraction layer between your code and the implementation details of using a payment gateway API.

It has drivers for many different gateways.

Omnipay will fix your payment gateway problems like **PDO** fixes your MySQL problems.

PDO for Payments

Omnipay gives you a consistent API across different implementations.

That makes it easy to move code from project to project, and means less code needs to be changed if the underlying gateway changes.

PDO for Payments

Omnipay won’t make your MSSQL queries run on Postgres. (So to speak.)

Different gateways still have different process flows, and different weird requirements.

Omnipay just eases some of the pain and unifies the interface.

Gateway support

Gateway drivers

Payment gateways are supported by drivers - a basic Adaptor pattern.

Omnipay core provides the framework.

Each gateway then has its own driver.

There are official, third party and then custom gateway drivers.

Official Gateways

2Checkout
Authorize.Net
Buckaroo
CardSave
Coinbase
Dummy
eWAY
First Data
GoCardless
Manual
Migs
Mollie
MultiSafepay
Netaxept (BBS)
Netbanx
PayFast
Payflow
PaymentExpress (DPS)
PayPal
Pin Payments
Sage Pay
SecurePay
Stripe
TargetPay
WorldPay

Third-party gateways

Agms

Alipay

Barclays ePDQ

CardGate

Cybersource

Cybersource SOAP

DataCash

ecoPayz

Fasapay

Fat Zebra

Globalcloudpay

Helcim

Neteller

Network Merchants Inc. (NMI)

Pacnet

PaymentSense

PayPro

PayU

Realex

SecPay

Sisow

Skrill

Wirecard

Let’s take a look.

Set up the gateway

Calling Omnipay::create() instantiates a new gateway object.

To make that gateway object useful, we need to set the security credentials. For Stripe, that's an API key.

Other gateways have different credentials that need to be set.

```php
<?php

use Omnipay\Omnipay;

// Setup payment gateway
$gateway = Omnipay::create('Stripe');
$gateway->setApiKey('abc123');
```

Make a card payment

The gateway's purchase() method takes an amount, a currency and details of the payment card.

This can be literal card details as shown, but is often a card token.

After detailing the purchase, the send() method sends the message to the gateway.

```
// Example card data
$cardData = [
    'number'      => '4242424242424242',
    ‘expiryMonth' => '6',
    'expiryYear'  => '2016',
    'cvv'         => '123'
];

// Send purchase request
$response = $gateway->purchase([
        'amount'   => '10.00',
        'currency' => 'USD',
        'card'     => $cardData
    ])->send();
```

Make a card payment

For token payments (like when using stripe.js) you can pass in a token instead of a card.

```
// Send token purchase request
$response = $gateway->purchase([
        'amount'   => '10.00',
        'currency' => 'USD',
        'token'    => 'abcd1234'
    ])->send();
```

Payment response

The response has an isSuccessful() method to check for success.

```
// Process response
if ($response->isSuccessful()) {

    // Payment was successful
     print_r($response);

} else {

    // Payment failed
     echo $response->getMessage();
}
```

Redirects

Many gateways respond to a payment request with a URL to send the customer to.

This is often the case for payment flows where the customer gives their card details direct to the gateway and not the merchant site.

Payment response

The response has an isSuccessful() method to check for success.

Some gateways take payment off-site. Those will test true for isRedirect().

If neither is the case, the payment failed.

```
// Process response
if ($response->isSuccessful()) {

    // Payment was successful
    print_r($response);

} elseif ($response->isRedirect()) {

    // Redirect to offsite payment gateway
    $response->redirect();

} else {

    // Payment failed
    echo $response->getMessage();
}
```

Redirects

After redirection, the gateway will usually make a call back to your code to indicate whether the transaction was successful or not.

Complete after redirect

When returning from an off-site gateway, you need to complete the purchase using the same options.

Some gateways validate options to make sure the transaction hasn't been messed with.

```
$gateway->completePurchase([
        'amount'        => '10.00',
        'currency'      => 'USD',
        'transactionId' => '1234'
    ])->send();
```

Options

Options

Most actions involve an `$options` array.

It’s often quite hard to figure out what should be in it, as every gateway expects something different.

There are a few common options, however.

Options

- card
- token
- amount
- currency
- description
- transactionId
- clientIp
- returnUrl
- cancelUrl

Setting options

Options are passed into most Omnipay action methods as an associative array.

```
$response = $gateway->purchase([
        'amount'         => '10.00',
        'currency'       => 'USD',
        'card'           => [ ... ],
        'description'    => 'Event tickets',
        'transactionId' => $order->id,
        'clientIp'       => $_SERVER['REMOTE_ADDR'],
        'returnUrl'      => ‘https://.../complete-payment/',
        'cancelUrl'      => ‘https://.../failed-payment/'
    ])->send();
```

Cards

- firstName
- lastName
- number
- expiryMonth
- expiryYear
- startMonth
- startYear
- cvv
- issueNumber
- type
- billingAddress1
- billingAddress2
- billingCity
- billingPostcode
- billingState
- billingCountry
- billingPhone
- shippingAddress1
- shippingAddress2
- shippingCity
- shippingPostcode
- shippingState
- shippingCountry
- shippingPhone
- company
- email

Yay abstraction!

```
billingAddress1 ==> adrStreet
```

What can we do?

Types of transaction

Authorize (and then capture)

Purchase

Refund

Void

Authorize

Authorization is performed with the `authorize()` method. This enables us to get the transaction reference.

When we want to take the money, we use the `capture()` method.

```
gateway = Omnipay::create('Stripe');
$gateway->setApiKey('abc123');

$response = $gateway->authorize([
        'amount'   => '10.00',
        'currency' => 'USD',
        'card'     => [ ... ]
    ])->send();

if ($response->isSuccessful()) {
  $transactionId = $response->getTransactionReference();

  $response = $gateway->capture([
        'amount'        => '10.00',
        'currency'      => 'USD',
        'transactionId' => $transactionId
    ])->send();
}
```

Purchase

Very straightforward, as we've already seen.

```
// Send token purchase request
$response = $gateway->purchase([
        'amount'   => '10.00',
        'currency' => 'USD',
        'token'    => 'abcd1234'
    ])->send();


$transactionId = $response->getTransactionReference();
```

Refund

Transactions can be refunded, although the bounds within this can be performed may depend on the gateway.

```
$response = $gateway->refund([
        'amount'        => '10.00',
        'currency'      => 'USD',
        'transactionId' => 'abc123'
    ])->send();
```

Void

A transaction can generally only be voided within the first 24 hours.

```
$response = $gateway->void([
        'amount'        => '10.00',
        'currency'      => 'USD',
        'transactionId' => 'abc123'
    ])->send();
```

Token billing

Create, update and delete cards.

Creating a card gives you a `cardReference` which can be used in future transactions.

```
$response = $gateway->createCard([
      'card' => [...],
  ])->send();

$cardId = $response->getTransactionReference();
```

Token billing

Create, update and delete cards.

Creating a card gives you a `cardReference` which can be used in future transactions.

```
$gateway->purchase([
  'amount'        => '10.00',
  'cardReference' => 'abc123'
  ])->send();
```

What can’t we do?

Limitations

No recurring billing.

Not much of anything else.

e.g. getting location details

Omnipay has a `fetchTransaction()` method which returns details of the transaction.

The response is gateway dependant, so may or may not have the information we need.

If it doesn't there may not be an Omnipay method available.

Going out of scope

When you need to do something the gateway driver doesn’t provide, things can get messy.

You either need to try to extend the driver, or fall back to code outside of Omnipay.

If your requirement is common, you might want to submit a patch.

Going out of scope

What you’re trying to do might not be a goal for the project.

See also: recurring payments.

Contributing

Contributing

Gateway drivers are maintained as individual open source projects with their own maintainers.

Making a change is as easy as making a Github pull request... which is to say it's of unknown ease.

Could be accepted, or rejected, or ignored. Yay open source.

Contributing

You can develop your own gateway driver.

There are guidelines to follow if you'd like it to be adopted as official.

Yay open source.

What’s good?

What’s good

Learn one API to use with all providers

Write code that can be moved between projects

Makes the friction of switching between providers much lower

Open source: benefit from others’ work

Open source: fix and contribute back when needed

What’s bad?

What's bad?

API is abstracted, but gateway flow is not

Limited to a lowest common denominator for functionality

No recurring payments

Open source: gateways are sometimes incomplete

Open source: getting PRs accepted can be hit and miss

On balance...

Omnipay is a useful library that takes a lot of friction away.

Be aware of what problems it *isn't* solving for you, and use it for the problems it *does* solve.

Thanks!

@drewm