Payment Drivers

Payment Drivers

What is the payment driver interface in my payment driver?

All payment gateway drivers must implement the `Aero\Payment\Contracts\PaymentDriver` interface.

The `Aero\Payment\PaymentDriver` abstract class can be used as the starting point, which supports the `ECOM` operation mode by default.

It's important to register the payment driver with the payment processor, which is used by the checkout and admin modules. This should be done in a `ServiceProvider` using the following call:

```
\Aero\Payment\PaymentProcessor::registerDriver('my_gateway', \Acme\PaymentDriver::class);
```

What are the capturing methods in my payment driver?

here are two types of capturing methods:

- **Automatic**– the payment is both authorised and captured when the customer clicks the pay button - **Manual**– the payment is authorised when the customer clicks the pay button, the funds are ring-fenced and captured at a later date

How do I cancel a payment in my payment driver?

If capturing of the payment is done at a later stage to the initial authorisation, the payment gateway should support the ability to cancel the transaction. The driver should therefore implement a `cancel` method.

The `cancel` method should return a `Aero\Payment\Responses\PaymentResponse` instance.

```
public function cancel(Payment $payment)
{
    $response = new \Aero\Payment\Responses\PaymentResponse();

    // request to cancel the transaction using the 3rd party API
    $capture = AcmePaymentsInc::cancelTransaction($payment->reference);

    // check if the transaction status is "cancelled"
    if ($capture->status !== AcmePaymentsInc::STATUS_CANCELLED) {
        // set an error on the response
        $response->setError('The transaction could not be cancelled.');

        return $response;
    }

    // internally mark the aero payment as cancelled
    $payment->cancel();

    // mark the response as successful
    $response->setSuccessful(true);

    return $response;
}
```

How do I capture a payment in my payment driver?

If the gateway provides the option to defer capturing the payment (authorising and capturing are performed at different times), the `capture` method should be implemented. This captured amount may be equal to or less than the authorised amount, and is always passed to the method in the lowest form of currency (pennies, cents, etc.).

The `capture` method should return a `Aero\Payment\Responses\PaymentResponse` instance.

```
public function capture(int $amount, Payment $payment)
{
    $response = new \Aero\Payment\Responses\PaymentResponse();

    // request to capture the transaction using the 3rd party API
    $capture = AcmePaymentsInc::captureTransaction($payment->reference, $amount);

    // check if the transaction status is "captured"
    if ($capture->status !== AcmePaymentsInc::STATUS_CAPTURED) {
        // update the status of the aero payment to "failed"
        $payment->update([
            'state' => \Aero\Payment\Models\Payment::FAILED,
        ]);

        // set an error on the response
        $response->setError('The transaction failed to be captured.');

        return $response;
    }

    // internally capture the amount for the payment
    $payment->capture([
        'amount' => $amount,
    ]);

    // mark the response as successful
```

```
    $response->setSuccessful(true);

    return $response;
}
```

Aero will attempt to auto-capture any authorised payments against an order when the order is marked as dispatched.

How do I refund a payment in my payment driver?

If the payment gateway provides the ability to externally make refund requests, the payment driver can use the `Aero\Payment\SupportsRefunding` trait.

The `refund` method should then be implemented, which returns an instance of `Aero\Payment\Responses\PaymentResponse`.

```
public function refund(int $amount, Payment $payment)
{
    $response = new \Aero\Payment\Responses\PaymentResponse();

    // request to refund the transaction using the 3rd party API, with a given amount
    $capture = AcmePaymentsInc::refundTransaction($payment->reference, $amount);

     // check if the transaction status is "refunded"
    if ($capture->status !== AcmePaymentsInc::STATUS_REFUNDED) {
       // set an error on the response
        $response->setError('The transaction could not be refunded.');

        return $response;
    }

    // internally refund the amount for the aero payment
    $payment->refund([
        'amount' => $amount,
    ]);

    // mark the response as successful
    $response->setSuccessful(true);

    return $response;
}
```

What are operating modes in my payment driver?

There are two modes a gateway can operate:

- `ECOM` – a standard ecommerce online transaction, typically originating from the storefront -
  `MOTO` – a transaction that is taken via telephone or through the mail, typically originating from the backend admin

As the processing environment differs when the customer isn't present (i.e. no 3D-secure for telephone orders), stores may require different merchant credentials for each mode.

ECOM mode

In order to indicate the gateway supports processing storefront orders, ensure the `Aero\Payment\SupportsEcomMode` trait is used.

MOTO mode

In order to indicate the gateway supports processing telephone and mail orders, ensure the `Aero\Payment\SupportsMotoMode` trait is used.

How do I support express checkout in my payment driver?

If the payment gateway stores customer information, this can be used to populate the contact information, billing and/or shipping address.

To specify that the payment gateway supports the express checkout flow, use the `Aero\Payment\OffersExpressCheckout` trait.

Since the express flow may not always be selected, the payment gateway must indicate if the customer is using the express checkout. This could be achieved by storing the status in their session:

```
public function isExpress(): bool
{
    return session()->has('acme_using_express_checkout');
}
```

Can I restricting availability of my payment driver?

It is likely that a payment gateway could have restrictions on when it can be used. For example, a finance based provider may require a certain order total amount, be restricted by geographical location or the items being purchased.

To determine if the gateway should be offered for the customer's session, use the `availableForCart` method within the payment driver:

```
public function availableForCart(Cart $cart): bool
{
    $minAmount = setting('my_gateway.min_order_amount');

    if ($minAmount !== null) {
        return $cart->total()->inc()->value >= $minAmount;
    }

    return true;
}
```

How do I return a payment response in my payment driver?

To unify all payment gateway driver communication, an `Aero\Payment\Responses\PaymentResponse` class is used. This common class must be returned from the `register`, `complete`, `capture`, `refund` and `cancel` methods.

Setting the payment view

Many payment gateways use an iframe or JavaScript code to bridge the store with the gateway provider. In order to output this to the customer on the checkout, or telephone operator for MOTO payments, the `PaymentResponse` can define a view to output. To do so, pass the view to the `setView` method. If no view is set, a JSON response will be returned.

Passing data to the payment view

If there is data to be passed to the view or JSON response, the `setData` method should be used.

Marking as successful

To indicate that the action carried out in the method was successful, then the response should be flagged as successful calling the `setSuccessful` method on the `PaymentResponse`.

Adding error messages

If an error occurred during the action, these messages can be passed to the `PaymentResponse` using the `setError` method.

Redirecting the user

If further action is required where the user needs to be redirected to a different URL (to complete 3D-secure on the card issuers website for example), a redirect can be set using the `setRedirect` method.

How do I register the transaction in my payment driver?

When a payment method is selected that uses the gateway driver, the `register` method is called. Within this method, the bootstrapping of the transaction should be carried out. This may involve connecting to the payment gateway's API to register the transaction. For example, the gateway may need to know the total amount to request from the customer's card, along with where to redirect the browser once the transaction has been processed.

The `register` method should return a `Aero\Payment\Responses\PaymentResponse` instance.

```
public function register()
{
    // make an API call to create the transaction with Acme Payments (the fake 3rd party gateway company)
    $request = AcmePaymentsInc::createTransaction([
        'merchant_reference' => $this->order->reference,
        'amount' => $this->order->total_rounded,
        'currency' => $this->order->currency_code,
        'billing_address' => [
            'line_1' => $this->order->billingAddress->line_1,
            'postcode' => $this->order->billingAddress->postcode,
            'country' => $this->order->billingAddress->country_code,
        ],
        'return_url' => route("payments-{$this->getMerchantMode()}.complete-payment"),
    ]);

    $response = new \Aero\Payment\Responses\PaymentResponse();

    // set the response as successful if the transaction status is "created"
    $response->setSuccessful($request->status === AcmePaymentsInc::STATUS_CREATED);

    // set the data to pass to the view
```

```
    $response->setData([
        'transaction_reference' => $request->reference,
    ]);

    // set the view to use (this is typically an iframe or JavaScript)
    $response->setView("acme-payments::{$this->getMerchantMode()}-form");

    return $response;
}
```

The callback URL(s) may differ depending on the operating mode. If the driver supports both `ECOM` and `MOTO` modes, the `getMerchantMode` method can be used:

```
$request = AcmePaymentsInc::createTransaction([
    // ...
    'return_url' => route("payments-{$this->getMerchantMode()}.complete-payment"),
]);
```

How do I complete the transaction in my payment driver?

Once the payment gateway has handled the customer inputting their secure details (such as card information), they will most likely be redirected back to the retailer's store. On doing so, the `complete` method is called on the payment driver, allowing for the transaction to be validated and capturing of the payment.

The `complete` method should return a `Aero\Payment\Responses\PaymentResponse` instance.

```
public function complete()
{
    $reference = $this->request->input('reference');

    // don't continue if a reference isn't provided
    abort_unless($reference, 404);

    $response = new \Aero\Payment\Responses\PaymentResponse();

    // the order total
    $amount = $this->order->total_rounded;

    // obtain the transaction from the 3rd party API
    $request = AcmePaymentsInc::getTransaction($reference);

    // validate if the currency matches
    if ($request->currency !== $this->order->currency_code) {
        $response->setError('Currency does not match.');

        return $response;
    }

    // validate if the amount matches
```

```
if ($request->amount !== $this->order->total_rounded) {
    $response->setError('Total amount does not match.');

    return $response;
}

// validate if the transaction status is "authorized"
if ($request->status !== AcmePaymentsInc::STATUS_AUTHORIZED) {
    $response->setError('The transaction was not authorized.');

    return $response;
}

// create the payment record in the aero database
$payment = $this->order->payments()->updateOrCreate([
    'reference' => $reference,
], [
    'id' => (string) \Illuminate\Support\Str::uuid(),
    'payment_method_id' => $this->method->getKey(),
    'state' => \Aero\Payment\Models\Payment::AUTHORIZED,
    'amount' => $amount,
    'currency_code' => $this->order->currency->code,
    'exchange_rate' => $this->order->currency->exchange_rate,
    'merchant_mode' => $this->getMerchantMode(),
]);

// request to capture the transaction using the 3rd party API
$capture = AcmePaymentsInc::captureTransaction($reference, $amount);

// check if the transaction status is "captured"
if ($capture->status !== AcmePaymentsInc::STATUS_CAPTURED) {
    // update the status of the aero payment to "failed"
    $payment->update([
        'state' => \Aero\Payment\Models\Payment::FAILED,
    ]);

    // set an error on the response
    $response->setError('The transaction failed to be captured.');

    return $response;
```

```
    }

    // internally capture the amount for the payment
    $payment->capture([
        'amount' => $amount,
    ]);

    // mark the response as successful
    $response->setSuccessful(true);

    return $response;
}
```

How do I display payment icons on checkout in my payment driver?

To improve brand awareness of the payment provider during the checkout process, a label view can be used. The payment driver should implement the labelView method, which returns a string. This is typically a namespaced view that lives in the payment gateway module.

```
public function labelView(PaymentMethod $method): ?string
{
    return view('my_gateway::label', compact('method'));
}
```

How do I restrict the availability of my payment driver?

It is likely that a payment gateway could have restrictions on when it can be used. For example, a finance based provider may require a certain order total amount, be restricted by geographical location or the items being purchased.

To determine if the gateway should be offered for the customer's session, use the `availableForCart` method within the payment driver:

```
public function availableForCart(Cart $cart): bool
{
    $minAmount = setting('my_gateway.min_order_amount');

    if ($minAmount !== null) {
        return $cart->total()->inc()->value >= $minAmount;
    }

    return true;
}
```