

GetSandbox Mocks

Giving internal control, around [possibly shaky] vendor API's

Lewis Cowles.

March 2022

What

The GetSandbox mocks are a way for us to take control of QA testing, both happy-paths, and coordinating complex interactions between services and third-parties.

Why

Some vendors do not allow for volume or load-testing, others simply behave in unpredictable manner, or are difficult to coordinate; requiring operator intervention to ensure a service is healthy.

The mocks also help overcome missing implementations, like checkout reporting API, and “magic values”, like special card numbers and amounts that trigger actions.

Why

Some vendors do not allow for volume or load-testing, others simply behave in unpredictable manner, or are difficult to coordinate; requiring operator intervention to ensure a service is healthy.

The mocks also help overcome missing implementations, like checkout reporting API, and “magic values”, like special card numbers and amounts that trigger actions, such as refunds.



Where

Our mocks currently run on chip internal-infrastructure.

They used to run on getsandbox.com but that required QA to edit code, and perform manual tasks. Expired regularly [request limit], etc.

Challenges

- Implementing without production access
- Implementing based on existing system(s)
- Overcoming magic values
- Adding controls & visibility
- Intentional fixtures
- Coordinating web hooks
- Request-response flows
- Coordinating sequences of interactions
- Multi-tenancy
- SQS
- Persistence
- Web-hooks
- The need to edit code

Running locally

Local set-up

1. From `Dockerfile`, find the docker image name. It will look similar to `346121089040.dkr.ecr.eu-west-2.amazonaws.com/mock-sandbox:latest`.
2. Pull the docker image (login to AWS via the command-line `awslogin`, if you haven't already).

```
docker pull 346121089040.dkr.ecr.eu-west-2.amazonaws.com/mock-sandbox:latest
```

3. Go to the `checkout-mock` app root directory - the same directory as the `Dockerfile`.
4. Run the docker container.

```
docker run -v $(pwd):/base -p 8081:80 -p 8082:90 -it 346121089040.dkr.ecr.eu-west-2.amazonaws.com/mock-sandbox:
```

Mock app is available at localhost:8081

Inspecting web-traffic

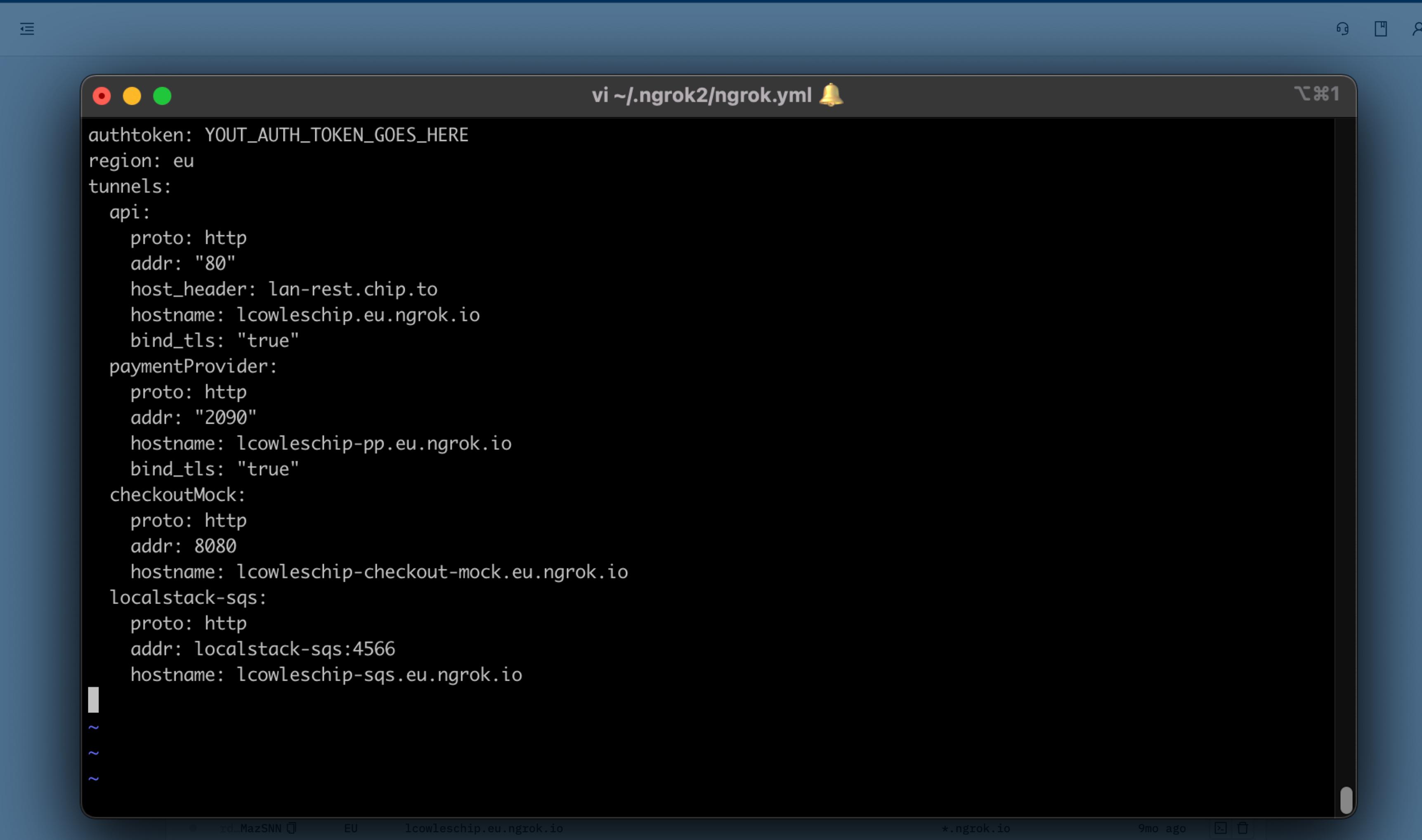
[Documenting existing systems]

The screenshot shows the ngrok dashboard interface. At the top, there's a search bar with the query "lcowles" and buttons for "API Docs" and "+ New Domain". Below the search bar is a table titled "Domains" with the following columns: ID, Region, Domain, Description, TLS, Status, and Created. The table lists ten entries, each with a small icon, the ID, the region (EU), the domain name, a brief description, the TLS setting (*.ngrok.io), the status (active), and the creation date (ranging from 2mo ago to 9mo ago). Each row also has edit and delete icons.

| ID | Region | Domain | Description | TLS | Status | Created |
|------------|--------|--|----------------------|------------|---------|---------|
| rd..kf518o | EU | lcowleschip-sqs.eu.ngrok.io | | *.ngrok.io | 2mo ago | |
| rd..gNIoF2 | EU | lcowleschip-goals.eu.ngrok.io | Goals Microservice | *.ngrok.io | 4mo ago | |
| rd..F18Thd | EU | lcowleschip-bc.eu.ngrok.io | | *.ngrok.io | 5mo ago | |
| rd..UpxqpA | EU | lcowleschip-bankconnect-mock.eu.ngrok.io | | *.ngrok.io | 6mo ago | |
| rd..Q5PnPf | EU | lcowleschip-checkout-mock.eu.ngrok.io | Checkout.com Mock | *.ngrok.io | 6mo ago | |
| rd..kmgidr | EU | lcowleschip-truelayer-mock.eu.ngrok.io | TrueLayer Mock | *.ngrok.io | 6mo ago | |
| rd..ADMKMi | EU | lcowleschip-pp-mock.eu.ngrok.io | PaymentProvider Mock | *.ngrok.io | 6mo ago | |
| rd..gORjg1 | EU | lcowleschip-back.eu.ngrok.io | | *.ngrok.io | 9mo ago | |
| rd..yD0Q0p | EU | lcowleschip-pp.eu.ngrok.io | | *.ngrok.io | 9mo ago | |
| rd..MazSNN | EU | lcowleschip.eu.ngrok.io | | *.ngrok.io | 9mo ago | |

Inspecting web-traffic

[Documenting existing systems]



A screenshot of a macOS terminal window titled "vi ~/ngrok2/ngrok.yml". The window shows a configuration file for ngrok tunnels. The file contains the following YAML code:

```
authtoken: YOUT_AUTH_TOKEN_Goes_HERE
region: eu
tunnels:
  api:
    proto: http
    addr: "80"
    host_header: lan-rest.chip.to
    hostname: lcowlleschip.eu.ngrok.io
    bind_tls: "true"
  paymentProvider:
    proto: http
    addr: "2090"
    hostname: lcowlleschip-pp.eu.ngrok.io
    bind_tls: "true"
  checkoutMock:
    proto: http
    addr: 8080
    hostname: lcowlleschip-checkout-mock.eu.ngrok.io
  localstack-sqs:
    proto: http
    addr: localstack-sqs:4566
    hostname: lcowlleschip-sqs.eu.ngrok.io
```

The terminal window has a dark mode interface with red, yellow, and green status icons at the top. The bottom of the window shows a scroll bar and three tilde symbols (~) indicating the end of the file.

Inspecting web-traffic

[Documenting existing systems]



The screenshot shows a terminal window titled "ngrok start --all". The window title bar includes the text "ngrok by @inconshreveable" and "(Ctrl+C to quit)". The main content of the terminal is as follows:

```
Session Status          online
Account                Chip Financial Ltd (Plan: Pro)
Version                2.3.40
Region                 Europe (eu)
Web Interface          http://127.0.0.1:4040
Forwarding              http://lcowleschip-checkout-mock.eu.ngrok.io -> http://localhost:8080
Forwarding              https://lcowleschip-checkout-mock.eu.ngrok.io -> http://localhost:8080
Forwarding              https://lcowleschip-pp.eu.ngrok.io -> http://localhost:2090
Forwarding              http://lcowleschip-sqs.eu.ngrok.io -> http://localstack-sqs:4566
Forwarding              https://lcowleschip-sqs.eu.ngrok.io -> http://localstack-sqs:4566
Forwarding              https://lcowleschip.eu.ngrok.io -> http://localhost:80

Connections            ttl     opn      rt1      rt5      p50      p90
                      0       0       0.00    0.00    0.00    0.00
```

At the bottom of the terminal window, there is a footer with the following information: "id: MazSNN" and "EU" on the left, "lcowleschip.eu.ngrok.io" in the center, and "*ngrok.io" and "9mo ago" on the right.

Inspecting web-traffic

[Documenting existing systems]



The screenshot shows a terminal window titled "ngrok start --all". The window title bar includes the text "ngrok by @inconshreveable" and "(Ctrl+C to quit)". The main content of the terminal is as follows:

```
Session Status          online
Account                Chip Financial Ltd (Plan: Pro)
Version                2.3.40
Region                 Europe (eu)
Web Interface          http://127.0.0.1:4040
Forwarding              http://lcowleschip-checkout-mock.eu.ngrok.io -> http://localhost:8080
Forwarding              https://lcowleschip-checkout-mock.eu.ngrok.io -> http://localhost:8080
Forwarding              https://lcowleschip-pp.eu.ngrok.io -> http://localhost:2090
Forwarding              http://lcowleschip-sqs.eu.ngrok.io -> http://localstack-sqs:4566
Forwarding              https://lcowleschip-sqs.eu.ngrok.io -> http://localstack-sqs:4566
Forwarding              https://lcowleschip.eu.ngrok.io -> http://localhost:80

Connections            ttl     opn      rt1      rt5      p50      p90
                      0       0       0.00    0.00    0.00    0.00
```

At the bottom of the terminal window, there is a footer with the following information: "id: MazSNN" and "EU" on the left, "lcowleschip.eu.ngrok.io" in the center, and "*ngrok.io" and "9mo ago" on the right.

Inspecting web-traffic

[Documenting existing systems]

The screenshot shows the ngrok - Inspect application window. The title bar reads "ngrok - Inspect" and the address bar shows "127.0.0.1:4040/inspect/http". The main interface has tabs for "ngrok", "online" (highlighted), "Inspect", and "Status". A red circle highlights the "Filter by" input field at the top left of the request list.

All Requests

| Method | Endpoint | Status | Duration |
|--------|---------------------|-----------------|----------|
| POST | /phone/validate/pin | 502 Bad Gateway | 0.41ms |
| POST | /phone/validate | 502 Bad Gateway | 0.45ms |
| POST | /device/register | 502 Bad Gateway | 0.4ms |
| GET | /health | 502 Bad Gateway | 0.55ms |

A specific GET /health request is selected and highlighted with a black box. The details pane shows:

1 minute ago Duration 0.55ms IP 3.10.82.218

GET /health

Summary Headers Raw Binary Replay ▾

Replay with Modifications

```
GET /health HTTP/1.1
Host: lan-rest.chip.to
User-Agent: Chip; iOS; 4.1.0(288); iPhone; 13.4.1;
Accept: /*
Accept-Encoding: gzip, deflate, br
Postman-Token: 5b88b558-5a2b-45d3-abd4-70668170fc03
X-Forwarded-For: 3.10.82.218
X-Forwarded-Proto: https
X-Original-Host: lcawleschip.eu.ngrok.io
X-Wsse: UsernameToken Username="84fc114fb5432d1f0a0ae3cb57eeccb68185b51a1", PasswordDigest="0TQ20GQ20Tk5NjgxZjZjNDU4ZDZiMzY3NTM5MGU5YWEw0WE2YTk5NTU3YWFlMmU1Y2ZjY2I1ZD
```

Inspecting web-traffic

[Documenting existing systems]

The screenshot shows the ngrok - Inspect application window. The title bar reads "ngrok - Inspect" and the address bar shows "127.0.0.1:4040/inspect/http". The main interface has tabs for "ngrok", "online" (highlighted), "Inspect", and "Status". A "Documentation" link is in the top right.

A "Filter by" input field is present. On the left, a table titled "All Requests" lists four entries:

| Method | Endpoint | Status | Duration |
|--------|---------------------|-----------------|----------|
| POST | /phone/validate/pin | 502 Bad Gateway | 0.41ms |
| POST | /phone/validate | 502 Bad Gateway | 0.45ms |
| POST | /device/register | 502 Bad Gateway | 0.4ms |

Below this, a "GET /health" entry is highlighted with a black box and shows a "502 Bad Gateway" status with a duration of "0.55ms".

On the right, a detailed view for the "GET /health" request is shown. It includes fields for "Time ago", "Duration", and "IP". Below this, there are tabs for "Summary", "Headers", "Raw", "Binary" (which is selected and highlighted with a red oval), and "Replay". A "Replay with Modifications" button is also visible within this tab group.

The "Binary" tab content displays the raw HTTP request headers and body:

```
GET /health HTTP/1.1
Host: lan-rest.chip.to
User-Agent: Chip; iOS; 4.1.0(288); iPhone; 13.4.1;
Accept: /*
Accept-Encoding: gzip, deflate, br
Postman-Token: 5b88b558-5a2b-45d3-abd4-70668170fc03
X-Forwarded-For: 3.10.82.218
X-Forwarded-Proto: https
X-Original-Host: lcawleschip.eu.ngrok.io
X-Wsse: UsernameToken Username="84fc114fb5432d1f0a0ae3cb57eeccb68185b51a1", PasswordDigest="0TQ20GQ20Tk5NjgxZjZjNDU4ZDZiMzY3NTM5MGU5YWEw0WE2YTk5NTU3YWFlMmU1Y2ZjY2I1ZD
```

State

[How to persist data in a mock?]

The screenshot shows the Postman application interface. On the left, the sidebar lists various workspaces, APIs, environments, and other tools. The main area displays a collection named "Mock Checkout API / Mock State / Mock State". A GET request is selected, with the URL being `GET {{ckoApiUrl}}/mock/state`. The Headers tab shows six entries. The Body tab is active, displaying the response body in JSON format:

```
1  "tokens": [],
2  "cards": [
3    "4658584090000001": {
4      "scheme": "Mastercard",
5      "bin": "500018",
6      "card_category": "Consumer",
7      "issuer_country": "GB",
8      "product_id": "CIR",
9      "product_type": "Cirrus®"
10     }
11   },
12   "delaySettings": {},
13   "payments": {},
14   "paymentLocks": {},
15   "customers": {},
16   "sources": {},
17   "charges": {}
```

The status bar at the bottom indicates a 200 OK status with a time of 55 ms and a size of 912 B.

State

[How to persist data in a mock?]

The screenshot shows the Postman application interface. On the left, the sidebar lists various workspaces, APIs, environments, and other tools. The main area displays a collection named "Mock Checkout API / Mock State / Patch Mock State". A PATCH request is selected, with the URL being `{ckoApiUrl}/mock/state`. The "Body" tab is active, showing a JSON payload:

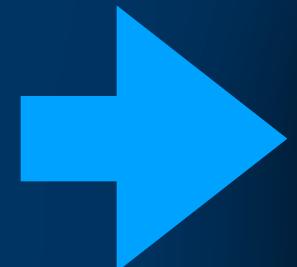
```
1 {
2     "cards": [
3         "424212344242425678": {
4             "scheme": "Visa",
5             "bin": "123456",
6             "card_category": "Consumer",
7             "issuer_country": "GB",
8             "product_id": "VISA",
9             "product_type": "Vida Electron"
10        }
11    }
12 }
```

Below the body, the response status is shown as 200 OK with a time of 248 ms and a size of 1.35 KB. The response body is also displayed in pretty JSON format:

```
1 {
2     "tokens": [],
3     "cards": [
4         "4658584090000001": {
5             "scheme": "Mastercard",
6             "bin": "500018",
7             "card_category": "Consumer",
8             "issuer_country": "GB",
9             "product_id": "VISA",
10            "product_type": "Vida Electron"
11        }
12    }
13 }
```

State

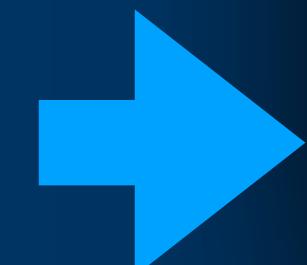
[How to persist data in a mock?]



```
● ● ●  
  
exports.resetState = function() {  
    state.tokens = {};  
    state.cards = {  
        "4658584090000001": defaultCardDetails  
    };  
    state.delaySettings = {};  
    state.payments = {};  
    state.paymentLocks = {};  
    state.customers = {};  
    state.sources = {};  
    state.payout = {  
        statement_id: '190110B107654',  
        id: 'CK0_MOCK_PAYOUT',  
        status: 'Remitted',  
        currency_payout: 'GBP',  
        carried_forward_amount: 0,  
        current_period_amount: 0,  
        net_amount: 0,  
        payout_fee: '-5',  
        channel: 'save'  
    };  
    state.channels = {};  
};  
exports.getChannel = function(req) {  
    return state.channels[req.headers.Authorization] ?? DEFAULT_CHANNEL;  
}
```

State

[How to persist data in a mock?]



```
● ● ●

exports.resetState = function() {
  state.tokens = {};
  state.cards = {
    "4658584090000001": defaultCardDetails
  };
  state.delaySettings = {};
  state.payments = {};
  state.paymentLocks = {};
  state.customers = {};
  state.sources = {};
  state.payout = {
    statement_id: '190110B107654',
    id: 'CK0_MOCK_PAYOUT',
    status: 'Remitted',
    currency_payout: 'GBP',
    carried_forward_amount: 0,
    current_period_amount: 0,
    net_amount: 0,
    payout_fee: '-5',
    channel: 'save'
  };
  state.channels = {};
};
exports.getChannel = function(req) {
  return state.channels[req.headers.Authorization] ?? DEFAULT_CHANNEL;
}
```

Timeouts

[Testing third-party without stubbing your code]



Common Code

```
const getDelaySettings = function() {
  return state.delaySettings || {};
}
exports.getDelaySettings = getDelaySettings;
exports.delayForRequest = function(req, res) {
  const reqSanitized = `${req.method} ${req.path}`;
  const delayTime = getDelaySettings()[reqSanitized] ?? 0;
  console.log(`REQUEST_DELAY ${reqSanitized}: ${delayTime}`);
  if (delayTime > 0) {
    res.setResponseDelay(delayTime);
  }
}
```

Timeouts

[Testing third-party without stubbing your code]



```
const getDelaySetti  
    return state.  
}  
exports.getDelaySe  
exports.delayForRe  
const reqSanitiz...  
const delayTime...  
console.log(`  
  
if (delayTime...  
    res.setRes...  
}  
}  
}
```



```
var config = require("./utilities/config"); Import common code  
  
// rest of file  
  
/**  
 * Identification  
 */  
Sandbox.define("/", "GET", function(req, res) {  
    config.delayForRequest(req, res); One-line per handler  
    res.set('Content-Type', 'text/html');  
    res.status(200).send("<h1>Checkout.com mock</h1>");  
});
```

Timeouts

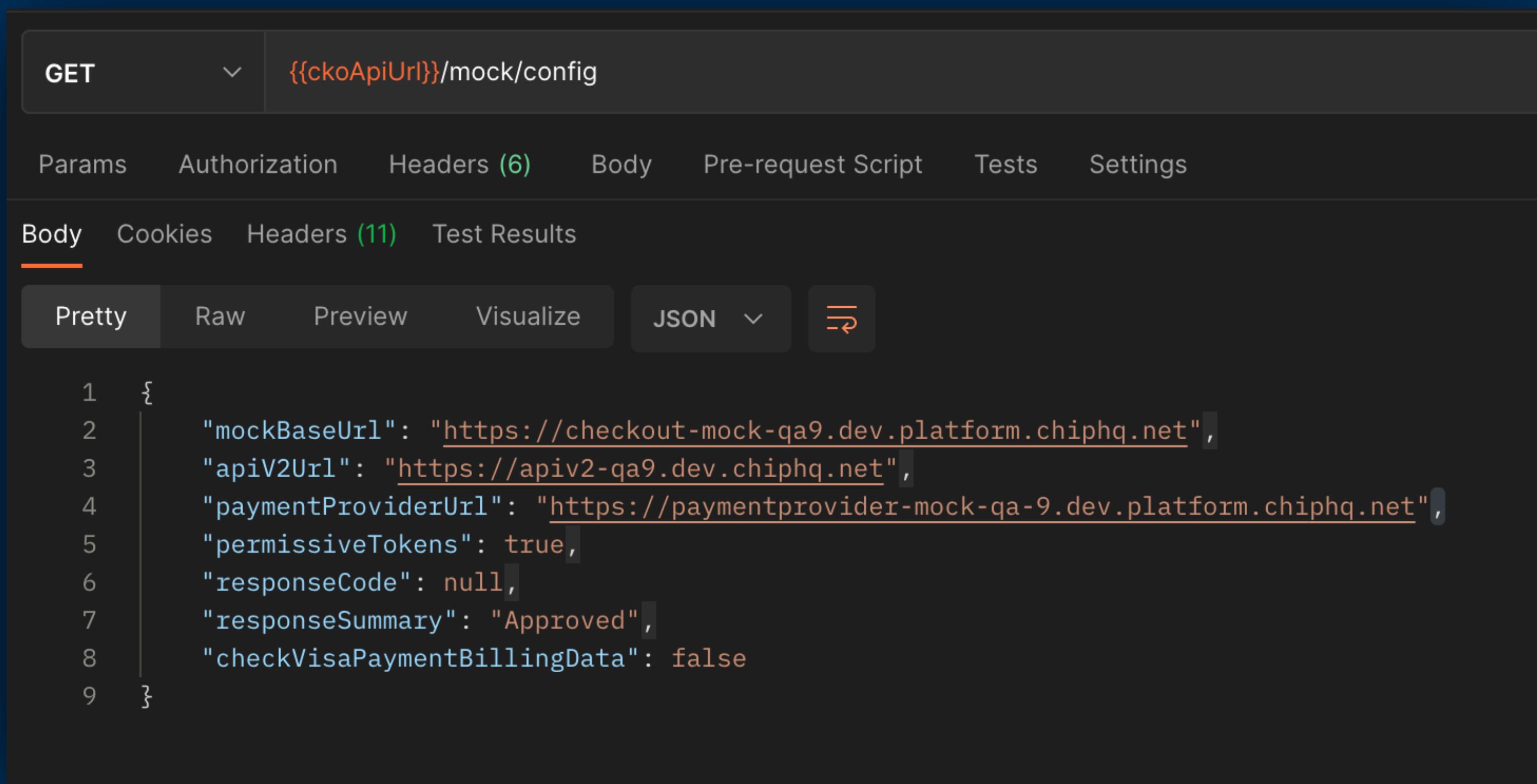
[Testing third-party without stubbing your code]

The image shows a developer's workspace with two main components:

- Code Editor:** On the left, a code editor displays a portion of a file named `delaySettings.js`. The code includes logic for getting delay settings from configuration and applying them to responses.
- API Testing Tool:** On the right, a browser-based tool for testing APIs. It shows a `PATCH` request to `{{ckoApiUrl}}/mock/state`. The **Body** tab is selected, showing a JSON payload:

```
1 {  
2   "delaySettings": {  
3     "GET /": 1000  
4   }  
5 }
```

Adding controls



The screenshot shows the Postman application interface. At the top, there is a header bar with a dropdown menu set to "GET" and a URL field containing "{{ckoApiUrl}}/mock/config". Below the header, there are several tabs: "Params", "Authorization", "Headers (6)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is currently selected, indicated by an orange underline. Under the "Body" tab, there are four options: "Pretty" (selected), "Raw", "Preview", and "Visualize". To the right of these options is a "JSON" dropdown menu and a copy icon. The main content area displays a JSON response with line numbers on the left:

```
1  {
2    "mockBaseUrl": "https://checkout-mock-qa9.dev.platform.chiphq.net",
3    "apiV2Url": "https://apiv2-qa9.dev.chiphq.net",
4    "paymentProviderUrl": "https://paymentprovider-mock-qa-9.dev.platform.chiphq.net",
5    "permissiveTokens": true,
6    "responseCode": null,
7    "responseSummary": "Approved",
8    "checkVisaPaymentBillingData": false
9 }
```

Coordinating web hooks [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays various sections: Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A collection named "Chip API V2 Monolith" is selected. The main workspace shows a request for "WebHook-Capture Payment" with a GET method and a URL template {{ckoApiUrl}}/payments/:checkoutPaymentId. Below the request, the "Tests" tab is active, showing a JavaScript test script. A red oval highlights the "WebHook-Capture P..." part of the script. The right side of the interface includes a sidebar with snippets and environment variable options.

```
pm.test('gets payment info for manual save', () => {
    pm.response.to.have.status(200);
};

var paymentData = pm.response.json();
pm.environment.set('save-id', paymentData?.metadata?.chip_user_id);
pm.environment.set('payment-id', paymentData?.metadata?.payment_id);
pm.environment.set('receipt-id', paymentData?.metadata?.id);

// Send Webhook
const paymentProviderUrl = pm.environment.get('paymentProviderUrl');
var uuid = require('uuid');
var newUuidv4 = uuid.v4();
const webHookRawBody = JSON.stringify({
    id: `evt_${newUuidv4}`,
    type: "payment_captured",
    created_on: paymentData.created_on,
    data: paymentData
});
const ckoSecret = pm.globals.get('CHECKOUT_SAVE_CHANNEL_SECRET_KEY');
pm.sendRequest({
    url: `${paymentProviderUrl}/webhook/checkout/channel/save`,
    method: 'POST',
    header: {
        'Content-Type': 'application/json',
        'CKO-Signature': CryptoJS.HmacSHA256(webHookRawBody, ckoSecret).toString(CryptoJS.enc.Hex)
    },
});
```

Coordinating web hooks [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays the 'Chip API V2 Monolith' workspace with various collections, APIs, environments, mock servers, monitors, flows, and history. The main area shows a collection named 'CPO Deposit Flow / Manual Save (3DS) / WebHook-Capture Payment'. A specific GET request is selected, with its URL path {{ckoApiUrl}}/payments/:checkoutPaymentId highlighted by a red oval. Below the request, the 'Tests' tab is active, displaying a JavaScript test script. The right side of the interface includes a sidebar with snippets and environment variable options.

```
pm.test('gets payment info for manual save', () => {
    pm.response.to.have.status(200);
});

var paymentData = pm.response.json();
pm.environment.set('save-id', paymentData?.metadata?.chip_user_id);
pm.environment.set('payment-id', paymentData?.metadata?.payment_id);
pm.environment.set('receipt-id', paymentData?.metadata?.id);

// Send Webhook
const paymentProviderUrl = pm.environment.get('paymentProviderUrl');
var uuid = require('uuid');
var newUuidv4 = uuid.v4();
const webHookRawBody = JSON.stringify({
    id: `evt_${newUuidv4}`,
    type: "payment_captured",
    created_on: paymentData.created_on,
    data: paymentData
});
const ckoSecret = pm.globals.get('CHECKOUT_SAVE_CHANNEL_SECRET_KEY');
pm.sendRequest({
    url: `${paymentProviderUrl}/webhook/checkout/channel/save`,
    method: 'POST',
    header: {
        'Content-Type': 'application/json',
        'CKO-Signature': CryptoJS.HmacSHA256(webHookRawBody, ckoSecret).toString(CryptoJS.enc.Hex)
    },
});
```

Coordinating web hooks [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays the 'Chip API V2 Monolith' workspace with various collections and environments. The main area shows a collection named 'CPO Deposit Flow / Manual Save (3DS) / WebHook-Capture Payment'. A specific GET request is selected, which has the URL {{ckoApiUrl}}/payments/:checkoutPaymentId. The 'Tests' tab is active, showing a block of JavaScript code. A red oval highlights the first eight lines of the test script, which set environment variables for save-id, payment-id, and receipt-id based on the response JSON. The rest of the script handles sending a webhook to the payment provider.

```
1 pm.test('gets payment info for manual save', () => {
2   pm.response.to.have.status(200);
3 });
4
5 var paymentData = pm.response.json();
6 pm.environment.set('save-id', paymentData?.metadata?.chip_user_id);
7 pm.environment.set('payment-id', paymentData?.metadata?.payment_id);
8 pm.environment.set('receipt-id', paymentData?.metadata?.id);
9
10 // Send Webhook
11 const paymentProviderUrl = pm.environment.get('paymentProviderUrl');
12 var uuid = require('uuid');
13 var newUuidv4 = uuid.v4();
14 const webHookRawBody = JSON.stringify({
15   id: `evt_${newUuidv4}`,
16   type: "payment_captured",
17   created_on: paymentData.created_on,
18   data: paymentData
19 });
20 const ckoSecret = pm.globals.get('CHECKOUT_SAVE_CHANNEL_SECRET_KEY');
21 pm.sendRequest({
22   url: `${paymentProviderUrl}/webhook/checkout/channel/save`,
23   method: 'POST',
24   header: {
25     'Content-Type': 'application/json',
26     'CKO-Signature': CryptoJS.HmacSHA256(webHookRawBody, ckoSecret).toString(CryptoJS.enc.Hex)
27   },
28 });
29
30 pm.response.to.have.status(200);
31 pm.environment.set('receipt-id', pm.response.json().id);
32 pm.environment.set('payment-id', pm.response.json().id);
33 pm.environment.set('save-id', pm.response.json().id);
34
35 pm.info(`Webhook sent successfully with ID: ${pm.environment.get('receipt-id')}`);
36
37 pm.done();
```

Coordinating web hooks [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays various sections: Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A red oval highlights a specific area in the code editor.

The main workspace shows a collection named "Chip API V2 Monolith". The "WebHook-Capture Payment" request is selected. The request details show a GET method with the URL {{ckoApiUrl}}/payments/:checkoutPaymentId. The "Tests" tab is active, displaying the following JavaScript test script:

```
1 pm.test('gets payment info for manual save', () => {
2   |   pm.response.to.have.status(200);
3 });
4
5 var paymentData = pm.response.json();
6 pm.environment.set('save-id', paymentData?.metadata?.chip_user_id);
7 pm.environment.set('payment-id', paymentData?.metadata?.payment_id);
8 pm.environment.set('receipt-id', paymentData?.metadata?.id);
9
10 // Send Webhook
11 const paymentProviderUrl = pm.environment.get('paymentProviderUrl');
12 var uuid = require('uuid');
13 var newUuidv4 = uuid.v4();
14 const webHookRawBody = JSON.stringify({
15   id: `evt_${newUuidv4}`,
16   type: "payment_captured",
17   created_on: paymentData.created_on,
18   data: paymentData
19 });
20 const ckoSecret = pm.globals.get('CHECKOUT_SAVE_CHANNEL_SECRET_KEY');
21 pm.sendRequest({
22   url: `${paymentProviderUrl}/webhook/checkout/channel/save`,
23   method: 'POST',
24   header: {
25     'Content-Type': 'application/json',
26     'CKO-Signature': CryptoJS.HmacSHA256(webHookRawBody, ckoSecret).toString(CryptoJS.enc.Hex)
27   },
28 });
29
30 pm.expect(pm.response.error).to.be.false;
31 pm.expect(pm.response.status).to.be.equal(200);
32 pm.expect(pm.response.json().status).to.be.equal("success");
33 pm.expect(pm.response.json().message).to.be.equal("Webhook sent successfully");
34 pm.expect(pm.response.json().data).to.be.an('object');
35 pm.expect(pm.response.json().data.id).to.be.a('string');
36 pm.expect(pm.response.json().data.type).to.be.a('string');
37 pm.expect(pm.response.json().data.created_on).to.be.a('string');
38 pm.expect(pm.response.json().data.data).to.be.an('object');
39 pm.expect(pm.response.json().data.data.id).to.be.a('string');
40 pm.expect(pm.response.json().data.data.type).to.be.a('string');
41 pm.expect(pm.response.json().data.data.created_on).to.be.a('string');
42 pm.expect(pm.response.json().data.data.data).to.be.an('object');
43 pm.expect(pm.response.json().data.data.data.id).to.be.a('string');
44 pm.expect(pm.response.json().data.data.data.type).to.be.a('string');
45 pm.expect(pm.response.json().data.data.data.created_on).to.be.a('string');
```

The right side of the interface contains a sidebar with various snippets and environment variable options.

Coordinating web hooks [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays the 'Chip API V2 Monolith' workspace with various collections and environments. The main area shows a 'WebHook-Capture Payment' test script for a 'GET' request. A red oval highlights the bottom portion of the code editor, which contains logic for sending a webhook.

```
pm.test('gets payment info for manual save', () => {
    pm.response.to.have.status(200);
};

var paymentData = pm.response.json();
pm.environment.set('save-id', paymentData?.metadata?.chip_user_id);
pm.environment.set('payment-id', paymentData?.metadata?.payment_id);
pm.environment.set('receipt-id', paymentData?.metadata?.id);

// Send Webhook
const paymentProviderUrl = pm.environment.get('paymentProviderUrl');
var uuid = require('uuid');
var newUuidv4 = uuid.v4();
const webHookRawBody = JSON.stringify({
    id: `evt_${newUuidv4}`,
    type: "payment_captured",
    created_on: paymentData.created_on,
    data: paymentData
});

const ckoSecret = pm.globals.get('CHECKOUT_SAVE_CHANNEL_SECRET_KEY');
pm.sendRequest({
    url: `${paymentProviderUrl}/webhook/checkout/channel/save`,
    method: 'POST',
    header: {
        'Content-Type': 'application/json',
        'CKO-Signature': CryptoJS.HmacSHA256(webHookRawBody, ckoSecret).toString(CryptoJS.enc.Hex)
    }
});
```

Managing SQS [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays the 'Chip API V2 Monolith' workspace with various collections, APIs, environments, mock servers, monitors, flows, and history. The 'Collections' section is expanded, showing 'Chip E2E Journeys' with sub-folders like 'Mock Health', 'Chip Health', 'Chip Onboarding Flow', 'Chip Onboarding Logs', 'Chip CB' (with 'CB Save Flow', 'CB Google Pay Deposit', 'CB Apple Pay Deposit', 'CB Complete Save', 'CB First Save Finish', 'CB First Save Logs', 'CB Withdraw Flow', 'CB Withdraw Logs', 'CB CancelAutoSave Flow'), and 'CB Refund Flow' (with 'Refund' and 'Start' requests). A red circle highlights the 'POST Fund Refund' request under the 'Refund' folder. The main panel shows the 'Fund Refund' request details: Method: POST, URL: {{sqS-URL}}/{{aws-account-ID}}/{{sqS-queue-prefix-pp}}-paymentProviderFinishSaveRefund_v2-queue, Body tab selected, Body type: x-www-form-urlencoded, Body content: Action: SendMessage, MessageBody: {{payload}}, and a table for Key and Value. The 'Send' button is visible at the top right.

Managing SQS [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays the project structure under 'Chip API V2 Monolith':

- Collections: Chip E2E Journeys (master)
 - Mock Health
 - Chip Health
 - Chip Onboarding Flow
 - Chip Onboarding Logs
- Environments
- Mock Servers
- Monitors
- Flows
- History

In the main workspace, a collection named 'CB Refund Flow / Refund / Fund Refund' is selected. A POST request named 'Fund Refund' is highlighted. The URL field contains the placeholder: `{{sqs-url}}/{{aws-account-id}}/{{sqs-queue-prefix-pp}}-paymentProviderFinishSaveRefund_v2-queue`. This URL is circled in red.

The request details panel shows the following configuration:

- Method: POST
- URL: `{{sqs-url}}/{{aws-account-id}}/{{sqs-queue-prefix-pp}}-paymentProviderFinishSaveRefund_v2-queue`
- Params
- Authorization
- Headers (12)
- Body (selected)
- Pre-request Script
- Tests
- Settings

Body type: x-www-form-urlencoded

| KEY | VALUE | DESCRIPTION |
|-------------|--------------------------|-------------|
| Action | SendMessage | |
| MessageBody | <code>{{payload}}</code> | |
| Key | Value | Description |

Managing SQS [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays collections, APIs, environments, mock servers, monitors, flows, and history. The main workspace shows a collection named "Chip API V2 Monolith" with a sub-collection "Chip E2E Journeys". Under "Chip E2E Journeys", there are several items including "Mock Health", "Chip Health", "Chip Onboarding Flow", "Chip Onboarding Logs", "Chip CB", "CB Save Flow", "CB Google Pay Deposit", "CB Apple Pay Deposit", "CB Complete Save", "CB First Save Finish", "CB First Save Logs", "CB Withdraw Flow", "CB Withdraw Logs", "CB CancelAutoSave Flow", "CB Refund Flow", "Refund", "Start", "Refund Save", "Fund Refund", and "Refund Money".

The central area shows a POST request for "Fund Refund" with the following details:

- Method:** POST
- URL:** {{sqs-url}}/{{aws-account-id}}/{{sqs-queue-prefix-pp}}-paymentProviderFinishSaveRefund_v2-queue
- Body:** (highlighted by a red oval)
- Params:** none
- Authorization:** (green dot)
- Headers:** (12 green dots)
- Body:** (green dot, highlighted by a red oval)
- Pre-request Script:** (green dot)
- Tests:** (green dot)
- Settings:** (green dot)

The "Body" tab is selected, showing the following parameters:

| KEY | VALUE | DESCRIPTION | Bulk Edit |
|-------------|-------------|-------------|-----------|
| Action | SendMessage | | |
| MessageBody | {{payload}} | | |

A red oval highlights the "Body" tab and the "MessageBody" parameter.

Managing SQS [E2E,SSI]

The screenshot shows the Postman application interface. The left sidebar displays a workspace named "Chip API V2 Monolith" containing various collections and environments. The main area shows a POST request for "Fund Refund" under the path "/CB Refund Flow / Refund / Fund Refund". The "Authorization" tab is selected, showing configuration for AWS Signature. The "Type" dropdown is set to "AWS Signature". The "AccessKey" field contains "{{DEV_AWS_ACCESS_KEY_ID}}". The "SecretKey" field contains "{{DEV_AWS_SECRET_ACCESS_KEY}}". Advanced options include "AWS Region" set to "eu-west-2", "Service Name" set to "sns", and "Session Token" set to "Session Token".

Postman

Home Workspaces API Network Reports Explore

Search Postman

Chip API V2 Monolith

Overview GET Configure mock POST Fund Refund

Build-Tooling [QA-9]

Save Send

POST {{sqS-URL}}/{{aws-account-ID}}/{{sqS-queue-prefix-pp}}-paymentProviderFinishSaveRefund_v2-queue

Params Authorization Headers (12) Body Pre-request Script Tests Settings Cookies

Type AWS Signature

AccessKey {{DEV_AWS_ACCESS_KEY_ID}}

SecretKey {{DEV_AWS_SECRET_ACCESS_KEY}}

ADVANCED

AWS Region eu-west-2

Service Name sns

Session Token Session Token

Collection: Chip E2E Journeys (master)

- Mock Health
- Chip Health
- Chip Onboarding Flow
- Chip Onboarding Logs
- Chip CB
 - CB Save Flow
 - CB Google Pay Deposit
 - CB Apple Pay Deposit
 - CB Complete Save
 - CB First Save Finish
 - CB First Save Logs
 - CB Withdraw Flow
 - CB Withdraw Logs
 - CB CancelAutoSave Flow
- CB Refund Flow
 - Refund
 - Start (GET)
 - Refund Save (POST)
 - Fund Refund (POST)
 - Refund Money (POST)

APIs Environments Mock Servers Monitors Flows History

Managing SQS [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays the 'Chip API V2 Monolith' workspace with various collections, environments, mock servers, monitors, flows, and history. The 'Chip E2E Journeys' collection is currently selected. Within this collection, the 'Refund' folder contains a 'POST Fund Refund' request. The main panel shows the request configuration for a POST method to a URL template: `{{sqS-url}}/{{aws-account-id}}/{{sqS-queue-prefix-pp}}-paymentProviderFinishSaveRefund_v2-queue`. The 'Pre-request Script' tab is active, displaying the following JavaScript code:

```
1 var sourceId = pm.variables.get("save-id")
2 var payload = {
3     "data": {
4         "paymentId": "ba1d8fad-3215-4abe-9efe-21e00265e064",
5         "receiptId": `pay_${sourceId}`,
6         "acquirerTransactionId": sourceId,
7         "sourceId": sourceId
8     }
9 }
10
11 pm.variables.set("payload", JSON.stringify(payload));
```

The right side of the interface includes a 'Send' button, a 'Save' dropdown, and a 'Copy' button. A sidebar on the far right provides links to various documentation and tools related to pre-request scripts.

Coordinating sequences of interactions [E2E,SSI]

The screenshot shows the Postman application interface. On the left, the sidebar displays collections, APIs, environments, mock servers, monitors, flows, and history. The main workspace shows a sequence of requests under the collection 'Chip API V2 Monolith'. A specific request, 'POST [Error] Save', is selected. The request details panel shows a POST method with the URL {{baseUrl}}/save/:productId. The 'Tests' tab is active, displaying a JavaScript test script:

```
1 pm.test("Status code is 418", function () {  
2     pm.response.to.have.status(418);  
3 } );  
4  
5 pm.test("Response structure", () => {  
6     pm.expect(pm.response.json().error).to.be.a('object');  
7     pm.expect(pm.response.json().error.code).to.be.a('number');  
8     pm.expect(pm.response.json().error.message).to.be.a('string');  
9 } );  
10  
11 pm.test("Response message", () => {  
12     pm.expect(pm.response.json().error.message).to.eql("There's not enough money in your bank to make this deposit. Please check your payment  
method.");  
13 } );
```

The right side of the interface includes a sidebar with various tools and documentation links.

Monitoring / Logs

The screenshot shows the Datadog Log Explorer interface. The top navigation bar includes the title "Log Explorer | Datadog" and a search bar with the query "app.datadoghq.eu/logs?query=kube_namespace%3Acheckout-mock-qa9&cols=host%2Cservice&index=&message...". The main interface has tabs for "Views" and "Logs", with "Logs" selected. A date range selector shows "15m Mar 3, 11:00 am - Mar 3, 11:15 am". Below this, a search bar is set to "Kubernetes Namespace:checkout-mock-qa9". The visualization mode is set to "List". A timeline at the bottom shows log entries from 11:01 to 11:15, with a vertical bar at 11:11 indicating the current time.

Search facets on the left include "CORE" (Index, Source, Host, Service), "Status" (mock-sandbox, Error, Warn), and "Error Outliers" (0).

The main table displays 12 logs found, with columns: DATE, HOST, SERVICE, and CONTENT. The first few log entries are:

| DATE | HOST | SERVICE | CONTENT |
|---------------------|---------------------|--------------|--|
| Mar 03 11:11:26.476 | i-05a25cdf0f53cf102 | mock-sandbox | >< Body: '{"mockBaseUrl": "https://checkout-mock-qa9.dev.platform.chiphq.n...} |
| Mar 03 11:11:26.476 | i-05a25cdf0f53cf102 | mock-sandbox | >< Headers: {Access-Control-Allow-Origin=*, Access-Control-Allow-Methods=...} |
| Mar 03 11:11:26.476 | i-05a25cdf0f53cf102 | mock-sandbox | >< Status: 200 (processing 1ms wallclock 1ms) |
| Mar 03 11:11:26.476 | i-05a25cdf0f53cf102 | mock-sandbox | >> No body found |
| Mar 03 11:11:26.476 | i-05a25cdf0f53cf102 | mock-sandbox | >> Headers: {Accept= */*, Accept-Encoding=gzip, deflate, br, content-length=...} |
| Mar 03 11:11:26.476 | i-05a25cdf0f53cf102 | mock-sandbox | >> HTTP GET /mock/config (Matched route '/mock/config') |
| Mar 03 11:11:18.475 | i-05a25cdf0f53cf102 | mock-sandbox | >< Body: '{"mockBaseUrl": "https://checkout-mock-qa9.dev.platform.chiphq.n...} |
| Mar 03 11:11:18.475 | i-05a25cdf0f53cf102 | mock-sandbox | >< Headers: {Access-Control-Allow-Origin=*, Access-Control-Allow-Methods=...} |
| Mar 03 11:11:18.475 | i-05a25cdf0f53cf102 | mock-sandbox | >< Status: 200 (processing 1459ms wallclock 1459ms) |
| Mar 03 11:11:18.475 | i-05a25cdf0f53cf102 | mock-sandbox | >> No body found |
| Mar 03 11:11:18.475 | i-05a25cdf0f53cf102 | mock-sandbox | >> Headers: {Accept= */*, Accept-Encoding=gzip, deflate, br, content-length=...} |

Future work

- Future without postman?
- Proxy in-front of mock for SQS / WebHooks?
- Templating
- Deployment
- Workshop on sandbox mocks?
- Ideas?

Questions?