



Technical Documentation

Version 1.2 – Jan 2024

Table of Contents

01_Overview	5
01.1_Platform Components	6
01.1.1_Processing	7
01.1.2_Dynamic Routing	8
01.1.3_Reconciliation	8
01.1.4_Analytics	9
01.2_Core Concepts	9
01.2.1_Providers	10
01.2.2_Payment Methods	11
01.2.3_Channels	13
01.2.4_Transactions	14
01.2.5_Elements	16
01.2.6_Lifecycle Events	17
01.3_Security	18
01.3.1_PCI DSS and PCI PIN	19
01.3.2_TLS AND HTTPS	22
01.3.3_Two Factor-Authentication	22
01.3.4_API Keys	23
01.4_Fraud Prevention	23
01.5_Stability and Scalability	23
02_Account	25
02.1_Authentication	27
02.2_API Keys and Environment	28
02.3_API key ID	32
02.4_Languages	35

02.5_Navigation and Pagination	35
02.6_Merchant and Sub-Merchants Accounts	37
02.6.1_Merchants	38
02.6.2_Merchant ID	40
02.6.3_Sub-merchants	43
02.7_Users and Authorization Groups	45
02.7.1_Users	45
02.7.2_User ID	49
02.7.3_Authorization Groups	52
02.7.4_Authorization Group ID	55
03_Permissions	58
04_Integrating	65
04.1_Dynamic Forms	73
04.1.1_Display Form	74
04.1.2_Create a Charge	77
04.1.3_Recurring Payments	83
04.1.3_Client Library Reference	84
04.2_Hosted Checkout (not allowed for cards payment method)	92
04.3_REST Integration (not allowed for cards payment method)	95
04.3.1_Display Form	96
04.3.2_Generating a Charge	98
04.3.3_Create Instrument	102
04.3.4_Alternative Flows	107
04.4_Transaction Flows	108
04.4.1_Capturing funds at a later time	111
04.4.2_Refunding Transactions	112
04.5_Event Handling	113
04.5.1_Receive Events	113

04.5.2_Event Details	114
04.5.3_Event Types and Parameters	117
04.6_Error Codes	139
04.6.1_Testing	145
05_Processing	149
05.1_Charge	150
05.2_Instrument	162
05.3_Payment	176
05.4_Reversal	180
05.5_Refund	184
06_Dynamic Routing	188
06.1_Routing Rules	189
06.2_Strategy	207
07_Analytics	209
08_Reporting	216
08.1_Reports	217
08.2_Templates	226
08.3_Reporting Schedules	232
09_Dashboard	236
09.1_Dashboard Settings	240
10_Annex	242
10.1_Payshop Channels	242
10.1.1_Card	242
10.1.2_MBWay	243
10.1.3_Multibanco	244
10.1.4_Payshop Reference	245
10.2_Asynchronous payments	247

01_Overview

The Payshop Online Payments platform has been designed from the beginning to provide flexibility and reduce the maintenance efforts of your payment operation.

Through a Single Integration Flow, a microservices architecture composed of platform Components and an abstracted and standardized means of processing transactions, we allow you to decouple your payments infrastructure. This grants you the opportunity to expand globally without struggles or delays.

1. Avoid the development and maintenance of custom connections to Payment Stakeholders like Acquirers, PSPs, Gateways, Processors, and Fraud Vendors.
2. Avoid building your own Payment Applications.
3. Reduce your PCI DSS scope, by keeping sensitive data off your servers while maintaining a seamless checkout experience across every channel.



Core Concepts

Core Concepts lays the foundations for understanding how we process any Payment Method through any Provider whilst only requiring a Single Integration Flow.

Core Concepts is all about getting on the same page.



Security

It takes a lot of effort to make sure payments are secure. Get to know our security principles and how we enforce them.

01.1_Platform Components

The Payshop Online Payments platform enables real-time transactions on an any-to-any basis: multi-channel, multi-network, multi-method, and multi-currency. This abstraction feeds a shared, event-based database on top of which run multiple internal applications and external value-added services. These make up our platform Components.

The Payshop Online Payments is composed of several cloud-agnostic microservices that can be deployed independently on cloud or on-premise security-certified servers. Microservices include Processing, Dynamic Routing, Reconciliation, Risk, Analytics, Dashboard, Merchant, Reporting, Event-Sourcing, Auth, Load-Balancers, Service Discovery, Monitoring, Logging, and Security Services.

1. Complete Control

The Payshop Online Payments platform's infrastructure provisioning and deployment are fully automated, enabling horizontal scalability, zero-downtime deployments, fault-tolerance, and disaster recovery.

2. Complete Security

The Payshop Online Payments platform is also PCI Level 1 certified and follows the strictest industry security standards.

The Payshop Online Payments platform is completely modular, and each Platform Component can consume both internal services or third party's. This system architecture allows for maximum flexibility on custom setups.

1. The Single Integration Flow can support custom transaction collection parameterization without any changes in your code for multiple payment channels;
2. The Payshop Online Payments platform feeds its own and 3rd party's Dashboard through REST APIs;
3. All Processing integrations, both international card schemes and any alternative Payment Methods, can be added in the transaction flow without any changes to your code

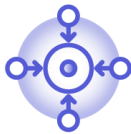
01.1.1_Processing

The Processing application uses the same abstractions for all transactions, regardless of them being synchronous or asynchronous, payins, payouts or marketplaces, redirection-based, pre-payments or post-payments, one-time or recurring. This way, you are able to add new payment channels without any changes in your code, while keeping your checkout experience fully customizable.



Future Proof

Our client-side library will check all Payment Channels enabled on any specific Merchant Account, returning a Form Schema for each of them. A Form Schema is a description of all the payment data fields required to process a specific payment method. It will then be possible to dynamically build any form based on that list. This means that if more payment channels are added in the future, the existing code will be ready for them.



Single Integration

The Single Integration Flow can adapt to any transaction flow, such as specific data collection, custom payment configurations, technical requirements, and UX specifications.



Tailor-Made

The abstraction created on our backend to support the multitude of payment flows can also enable authorization for custom setups. These can include not only financial transactions such as POS and Private Label schemes but also virtual value such as Loyalty and Coupons, making it one the most comprehensive and flexible payments platforms available on the market.

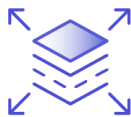
01.1.2_Dynamic Routing

Our Dynamic Routing component provides real-time switching capabilities that allow us to select the best Provider for a given transaction.



Industry Cognizant

Besides enabling connections industry-wide, the Payshop Online Payments platform adds a rules-based layer that intelligently routes transactions between a wide network of payment channels in real-time to maximize payment performance.



Flexible

A Transaction profiling system is processed to set default transaction metadata, which is used to define custom rules for automatic switching in real-time at Transaction level.

01.1.3_Reconciliation

Reconciliation of incoming settlements against bank accounts is a complex process. This activity becomes even more challenging when a business relies upon multiple Providers, with multiple reporting structures. Reconciliation helps oversee and uniformize transaction data towards thoughtful monitoring of your payment operations.



Source Standardization

The Reconciliation Engine processes Transaction Statements from different Sources and formats into a single data structure.



Time-Saving

Reconciliation produces Settlement events that automate and simplify the monitoring process.

01.1.4_Analytics

The Payshop Online Payments platform uses Big Data technologies to capture, store, analyze, search, share and visualize voluminous and complex payment datasets. Our platform transforms raw data into consumable information, generating valuable insights that empower payment managers to make better decisions.



Instantaneous Feedback

Every transaction parameter is indexed in the Lifecycle Database and becomes readily available for processing through the Analytics API.



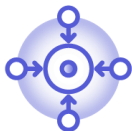
Diversity of Insights

You are able to perform queries to the database using multiple parameters to get operations insights, monitor your processing commissions, or control bookkeeping.

01.2_Core Concepts

Digital payments have been growing at a fast pace to support the ever-evolving global commerce market. New means of payment, new regions, new regulation, new security guidelines, and new customer segments ensure the payments landscape is constantly shifting.

Adjusting to these new realities can be challenging and complex. Our platform simplifies the process and puts an end to the growing pains that come with expanding your business. It was designed with flexibility in mind to support the growth of your payment operation. In order to achieve this, we defined core concepts in our architecture from the outset. These allowed us to maintain a single integration, no matter the payment method being processed, whilst giving you access to multiple Platform Components.



Single Integration Flow

First time's the charm. With this platform you get access to every present and future payment method, as well as all the tools required to run a professional, and comprehensive payments operation through a single integration.

01.2.1_Providers

In the Payshop Online Payments platform, a provider is a payments' value chain stakeholder to which the Processing application is connected to.

Acquirers

An acquiring bank, also known simply as an acquirer, is a bank or financial institution that processes credit or debit card payments on behalf of a merchant. The acquirer allows merchants to accept credit card payments from the card-issuing banks within an association.

Acquirer Processors

Acquirer processors connect directly with merchants, card networks, and the acquirer, to exchange transaction information between all stakeholders. They provide the technical capabilities to communicate authorization and settlement messages between the acquirer and the card networks.

Acquirer processors handle the technical side of the acquiring business, they do not assume financial liability for the process, as they are not involved in fund management. This liability and risks lie on acquirers. Nonetheless, it is important to note that In some cases, the acquiring bank and acquirer processor are a single entity (e.g. Adyen, Elavon).

Payment Service Providers (PSPs)

A payment service provider (PSP) is a payment institution with the right to manage third-party funds. It offers merchants multiple payment methods, namely credit card, direct debit, and bank transfers.

Typically, a PSP can connect to multiple acquiring banks, card networks, and payment networks, which makes the merchant less dependent on financial institutions by eliminating

the burden of establishing these interactions directly. This can be especially relevant when operating internationally.

Gateways

A payment gateway represents a technical layer that collects payment instrument credentials in the client-side and securely forwards them to the relevant payment service provider (PSP) or acquirer. The majority of gateway companies also offer other services.

Fraud Vendors

Fraud vendors are capable of predicting user intent and preventing fraudulent activity in real-time. They fundamentally manage risk and hinder financial crime.

Tokenization Service Providers

These institutions tokenize card data, which is the pseudonymization of credit cards sensitive data to facilitate compliance for merchants, PSPs, and acquirers. Tokenization service providers exist to grant a secure environment in which to store the card data sent by merchants.

01.2.2_Payment Methods

In its broader definition, a payment method is any set of infrastructure and regulation that supports value exchange between a network of participants.

Cards are not the only way to pay for online transactions. Between bank transfers, direct debits, eWallets, mobile payments, local card schemes, pre-pay, post-pay, and e-invoices, there are over 200 different types of alternative payment methods.

Alternative Payment Methods (APMs)

The designation of alternative payment method applies to any form of payment that is not cash or a card issued by a major bank. APMs are commonly used and increasingly adopted across sectors and territories to respond to consumer's trends. Accessing them is essential to any business.

Our platform allows you to process transactions using any payment method, from traditional credit/debit cards to alternative payment methods or custom payment methods like private label, credit-to-consumer, and loyalty schemes.



Some of the payment methods supported by the Payshop Online Payments platform

In the complex and diverse world of payments, there are multiple payment methods available in multiple markets.

Card

A physical or digital card issued by a financial entity that can be used for purchases. There are different types of cards available in the market, including debit, credit, and prepaid cards.

e-Wallet

Digital wallet that can store funds and perform electronic payments. It can be connected to a bank account and used to store card details.

Bank Transfer

Electronic payment directly performed from one bank account to another, between two individuals or an individual and an entity.

Direct Debit

Gives banking customers the ability to authorize third-party creditors to debit funds from their checking account.

Cryptocurrency

Digital asset designed to work as a medium of exchange. Individual coin ownership records are stored in a digital ledger or computerized database using strong cryptography to secure transaction record entries, to control the creation of additional digital coin records, and to verify the transfer of coin ownership.

Local Card Scheme

Local card schemes—specific to certain markets—often operate much like traditional cards, but some schemes will be more sophisticated, for instance, offering card and bank transfer options.

Carrier Billing

Online payment method that allows customers to use their phone carrier bill to pay mostly for digital goods.

Cash

Physical form of currency represented by banknotes and coins that can be exchanged for goods, typically during product delivery.

01.2.3_Channels

The concept of payment method and provider are interconnected in the sense that a given payment method may be available through several providers within the same market, the same way that a given provider may support multiple payment methods. They are also selected by two different entities, the method being chosen by the customer through the Dashboard, and the provider defined by the Dynamic Routing module.

The combination of a payment method and a provider represents a channel in the Payshop Online Payments platform. It is an actual medium that can be used to perform a financial transaction (e.g. Credit Card through Elavon, or Alipay through Worldpay).

A channel is configured through the Dashboard using the API keys for the merchant account that you have opened with the provider. By setting up multiple channels for the same payment method, and leveraging our Dynamic Routing component, you can define specific rules for real-time transaction routing between multiple providers.



Higher Acceptance Rate

Providers have different acceptance rates. These depend on multiple variables, such as card brand, issuing country/bank, MCC and amount. Failed transactions result in lost sales and increased customer support costs. Our platform can retry transactions in real-time across multiple Providers to maximise acceptance rates.



Lower Processing Costs

A rise in cross-border transactions is pushing processing costs up, especially due to inter-regional interchange fees and card scheme fees. The Payshop Online Payments platform calculates expected commissions based on multiple processing variables to choose the optimal transaction route.

01.2.4_Transactions

The diversity of the payments ecosystem poses a great challenge to global companies, as each payment channel requires its own specific integration, which is costly to build and maintain. As payment channels start to add up, companies become unable to optimize their payment operations in a timely manner.

With this challenge in mind, we devised a way to abstract every transaction flow in a single one, independent from the payment method, allowing companies to invest one time into integrating with our platform and gain access to the entire value chain without further development investments.

To this effect, we divide a financial Transaction into multiple correlated steps, which we either call Elements, or Lifecycle Events. Each of these steps represents a different phase in the transaction lifecycle within the Payshop Online Payments platform, and therefore, requires and stores different data.

Supporting different Payment Methods under the same platform in a standardized manner brings additional challenges when it comes to transaction execution. Each Payment Method is designed with its own philosophy when it comes to processing Transactions, and therefore requires different steps, and different execution flows.

Accommodating this diversity under the same platform required us to support four transaction flows.

Transaction Flow	Payment executed automatically?	Recurring Payment?
1. Capture on Creation	Yes	No
2. Auth-Capture	No	No
3. Recurring on Auth	Yes	Yes
4. Recurring on a Capture	No	Yes

Capture on Creation

This flow targets transactions that only need to take place once, and where the funds do not need to be captured at a later time, therefore not requiring an authorization to take place.

From a platform perspective, this flow does not require you to create the Payment element to execute the funds transfer. The Processing Application will automatically create the Payment element once you create the instrument, given that the funds will be captured right away.

Example usage: payment of digital goods.

Auth-Capture

An Auth-Capture flow is intended for transactions that take place once, and have the funds captured at a later time, thus requiring two steps: Authorization and Funds Capture.

The Authorization is executed when the Instrument element is created in our platform, either automatically using our Dynamic Forms or manually leveraging our REST API. At this stage the money has been reserved from the customer accounting platform but has not yet been transferred. The actual funds transfer only happens when

Recurring on Auth

Destined for payments that need to happen on a frequent basis, this flow allows to create a reusable `instrument` element that can be leveraged to make multiple Payments.

This particular recurring flow automatically executes the `Payment` element when the `Instrument` is created. It means that for the first payment, you only need to create the `Instrument`, as the platform will automatically create the first `payment`. For subsequent transactions you only need to create the `payment` element, referencing the same specific `instrument`.

Example usage: subscriptions.

the `payment` element is created in the Platform.

Example usage: hotel or rental car bookings

Recurring on Capture

Designed for transactions that need to take place regularly, enables you to create a reusable `Instrument` element that can be used to perform multiple Payments.

This particular recurring flow does not automatically execute the `Payment` element when the `Instrument` is created. In short, it requires you to create a `Payment` element every time you want to execute a transaction.

Example usage: subscriptions.

01.2.5_Elements

With the current and future variety of payment methods, ensuring they can conform to standardized operation is paramount. To be truly effective, the utilization experience of a payments platform should not require different integrations or execution flows.

The Payshop Online Payments platform defined elements to this effect, to allow a global way of handling payments in a single integration. By utilizing the elements described below we ensure our platform is future-proof, by handling any current and future payment methods in a uniform manner.

- 01 Charge**
Signals the Merchant's intent to do a fund transfer. It is at this stage that the Payment Method and Provider are selected as configured.

- 02 Instrument**
Element providing the information required to authorize one or more transactions for the target payment method (e.g. credit card details).

- 03 Payment**
Stage where transaction captures actually take place. May be initiated for certain transaction flows.
- 04 Reversal**
Reversals serve two purposes. They are used for reversing a capture or Refund that has not yet been cleared, or for voiding an authorization.
- 05 Refund**
Allows the complete or partial return of the funds to the originating customer. Can only be done for successful transactions and is initiated by the merchant.
- 06 Dispute**
Initiated by the provider, this element represents a customer or issuing bank questioning the validity of the original transaction and may result in funds' reversibility if the claim is successful.

01.2.6_Lifecycle Events

The Payshop Online Payments platform relies on a microservices' architecture to provide Platform Components besides the actual transaction processing. In order to achieve this, it uses an event-driven philosophy for cooperation and routing between services.

One added benefit of doing so is that it allows you to subscribe to notifications regarding the lifecycle of transactions. Examples of such notifications would include events like `charge.created`, `instrument.authorized` or `payment.success`. These can be leveraged to orchestrate payment-related activities on your side.

Learn more about lifecycle events.

Lifecycle events are the best way to keep track of transactions' statuses on the server-side, and update your databases and systems accordingly.



01.3_Security

It takes a lot of effort to make sure payments are secure. Fraud can be costly for merchants and financial institutions, so it is important to ensure all types of threats, attacks, and suspicious activity are closely monitored. There are six main security principles we follow when it comes to making sure your transactions are safe.



Secure network and systems

Software updates, firewalls, monitoring, logging.



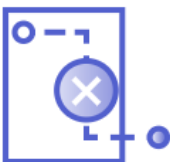
Safety of cardholder data

Hashing, encryption, restricted access to servers, storage, and networks.



Vulnerability management program

Intrusion Detection System (IDS), File Integrity Monitoring (FIM), monitoring and logging.

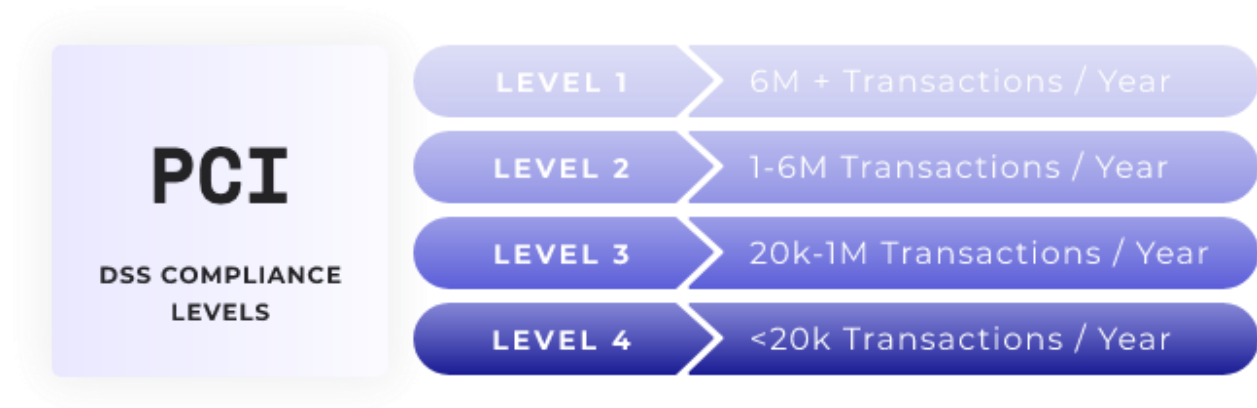


Access control

Restrict users access, track users access, restrict physical access.

01.3.1_PCI DSS and PCI PIN

Payment Card Industry Data Security Standard (PCI DSS) is a set of security standards formed in 2004 by Visa, MasterCard, Discover Financial Services, JCB International, and American Express that aims to secure credit and debit card data transactions against data theft and fraud. The PCI Security Standards Council defines a set of requirements intended to ensure that all companies that process, store, or transmit credit card information maintain a secure environment.



PCI DSS Compliance Levels

Level 1

On-site assessment conducted yearly by a [Qualified Security Assessor](#).

Level 2

Quarterly internal and external scans performed by an [Approved Scanning Vendor](#).

Level 3

Yearly internal penetration tests by an [Approved Scanning Vendor](#).

Level 4

Yearly revision of security and quality procedures.

In short, PCI DSS is a set of regulations that apply to all entities that store, process, and/or transmit cardholder data. It covers technical and operational practices for system components

included in or connected to environments with cardholder data. If you accept or process payment cards, PCI DSS applies to you. The criteria for Level 1 PCI DSS compliance depends on which card brands the merchant accepts, as follows:

- Visa, Mastercard, and Discover define Level 1 merchants as those processing more than 6 million credit card transactions annually;
- American Express's minimum for Level 1 is 2.5 million transactions per year;
- JCB's Level 1 starts at 1 million credit card transactions per year.

To meet PCI DSS Level 1 Standard there are five main required processes, which every compliant company must go through every year.

- Annual Report on Compliance (ROC) by a [Qualified Security Assessor \(QSA\)](#) or Internal Security Assessor;
- Quarterly network scan by [Approved Scan Vendor \(ASV\)](#);
- Penetration test;
- Internal Scan;
- Submission of completed Attestation of Compliance form.

The Payshop Online Payments platform is PCI DSS Level 1 compliant. But what does that mean exactly? And why should you care about it?

PCI DSS is a challenging, complex, and considerably expensive certification to achieve. By integrating with this platform you save your business the effort of going through with it.



Around 250 requirements comprise the [Payment Card Industry \(PCI\) Data Security Standard](#). You can find a simplified version of the requirements necessary to meet the PCI Compliance Security Standard Council's goals in the table below.

Goals	PCI DSS Requirements
Build and Maintain a Secure Network	<ol style="list-style-type: none"> 1. Install and maintain a firewall configuration to protect cardholder data 2. Do not use vendor-supplied defaults for system passwords and other security parameters
Protect Cardholder Data	<ol style="list-style-type: none"> 3. Protect stored cardholder data 4. Encrypt transmission of cardholder data across open, public networks
Maintain a Vulnerability Management Program	<ol style="list-style-type: none"> 5. Use and regularly update anti-virus software or programs 6. Develop and maintain secure systems and applications
Implement Strong Access Control Measures	<ol style="list-style-type: none"> 7. Restrict access to cardholder data by business need-to-know 8. Assign a unique ID to each person with computer access 9. Restrict physical access to cardholder data
Regularly Monitor and Test Networks	<ol style="list-style-type: none"> 10. Track and monitor all access to network resources and cardholder data 11. Regularly test security systems and processes
Maintain an Information Security Policy	<ol style="list-style-type: none"> 12. Maintain a policy that addresses information security for employees and contractors

01.3.2_TLS AND HTTPS

The Hypertext Transfer Protocol (HTTP) is the basic communication protocol that both clients and servers must implement in order to be able to communicate. It transfers information between the browser and the server in clear text, allowing the network, through which the information passes, to see the information transmitted. HTTP Secure (HTTPS) was introduced to allow the client and the server to first establish an encrypted communication channel, and then pass the clear text HTTP messages through it.

It is crucial for payment security that merchants implement SSL protocol on their websites. Transport Layer Security (TLS), and its now-deprecated predecessor, Secure Sockets Layer (SSL) allows for the encryption of the information that goes through the website, such as the payment card details that customers share during the checkout process.

HTTPS TLS provides three important security measures.

- **Confidentiality.** It protects communication between two parties from others within a public medium such as the internet.
- **Integrity.** It ensures information reaches its destined party in full and unaltered.
- **Authentication.** It grants that the website is actually what it claims to be by also checking its legal identity.

01.3.3_Two Factor-Authentication

Two-factor authentication is an extra layer of security designed to ensure that only the rightful users can access their respective accounts on the Payshop Online Payments platform, namely by logging into the Dashboard.

Two-Factor Authentication is enabled under settings for every user with access to the Dashboard.

01.3.4_API Keys

Different combinations between the Merchant API Keys or Processing API Keys and your account ID grant authentication in your requests. It is important to be well aware of when to use each of the keys and how to gain access to them.

- **Public Key.** Used in requests coming from the client-side, to create dynamic forms and instruments.
- **Account ID and Private Key.** Used in requests coming from the merchant server, to create charges, payments, reversals, and refunds, as well as accessing management APIs. This key SHOULD NEVER BE SHARED in client-side code.

Looking for more details on the Dashboard?

The Dashboard offers a comprehensive and user-friendly interface to support your payment strategy. Check out our Dashboard features.



01.4_Fraud Prevention

For every transaction, there are multiple security steps that can be taken. These can be either automatically applied by the Payshop Online Payments platform or are optional and can be enabled through our applications.

01.5_Stability and Scalability

We understand that stability and scalability are big concerns for the businesses that work with us. There are multiple efforts we make towards ensuring our services and applications are reliable.

Goals	Procedures
<p>Systems resilience High-Availability Stability</p>	<ul style="list-style-type: none"> - 24/7 alerting and monitoring: All infrastructure is monitored aboutsystem telemetry. - Application monitoring: all apps communicate to a logging system that centralizes infrastructure logs. -Monitoring of replica databases. - Critical failures are communicated through SMS/email, both internally and to our customers.
<p>High Availability SLAs</p>	<ul style="list-style-type: none"> - Historically, our system's availability surpasses 99.9%. - We are deployed in multiple cloud services and DNSs.
<p>Automatic Backup Disaster Recovery</p>	<ul style="list-style-type: none"> - Daily database backups. - Multiple and geographically dispersed database locations - Automatic disaster recovery: server provisioning, backup import, build, and run app services. - How fast? We annually test all disaster recovery procedures and it takes on average 2h.
<p>Zero Downtime Updates</p>	<ul style="list-style-type: none"> - Use of load balancers to route requests to machines running upgraded versions. - Use of load balancers to rollback requests to machines running previous versions.
<p>Scalability</p>	<ul style="list-style-type: none"> - Full horizontal scalability. - Microservices architecture, based on Docker containerization. - Databases and applications run in different and multiple machines. - Script for automatic microservices instances deployment.

- Service discovery: infrastructure orchestration system to which applications communicate to find the right servers.

- Data warehousing: different data representations to optimize data queries.

- Data denormalization.

- Event sourcing.

- Stream processing: event-based database to be consumed by different applications.

- Transaction flow for client-side tokenization.

Big Data Infrastructure

- Fully PCI Level I compliant: servers, data, network, and employees have implemented security measures to guarantee bank-grade security.

- GDPR compliant: the personal data privacy is fully encrypted and managed by a data controller, so it can be deleted on request.

02_Account

Get started with the Payshop Online Payments Platform.

Businesses are at the forefront of the Payshop Online Payments platform. When starting out with the platform, properly setting up your business account can be crucial to your integration efforts. Learn more about API keys and environments as well as how to proceed regarding authentication, whitelisting, languages, and navigation.



Authentication

All requests to any API use HTTPS basic auth. Get a quick overview of how to handle these authentication specifications.



Merchant Keys and Environment

Find more about Processing API Keys and Merchant API Keys and how to use them when setting up requests through the TEST or LIVE Environments.



Languages

Our APIs offer multiple language options. Learn how to enable them.



Navigation and Pagination

Understand how to filter and navigate responses.

If you are a merchant, you can use our Merchant API to create sub-merchants. The main merchant is responsible for onboarding each individual sub-merchant and it can also label accounts, set the hierarchy and customize white label features for the corresponding sub-merchants.

Associated to your main business account you can create multiple users' accounts. Each user account should correspond to a staff member. Individual users can have different sets of permissions associated with the authorization group they belong to.



Merchants and Sub-merchant Accounts
Create your merchant account and onboard sub-merchants seamlessly.



Navigation and Pagination
Make things easier on your team and distribute permissions that fit your operations.

02.1_Authentication

All requests to the APIs must use `HTTPS`. In our examples, we use `curl`, but any language-specific `HTTPS` request method or library will work. All requests to the API use `HTTPS Basic Auth`. If your library doesn't support it as a function, you can add the following header to your requests.

Basic Authorization

```
Authorization: Basic {credentials}
```

For requests from the server-side, `{credentials}` should be replaced with the base64 encoded string `accountId:privateKey`. For requests from the client-side, `{credentials}` should be replaced with the base64 encoded string `publicKey`.



Never expose your private key on the client-side!

Keep your transactions secure and your clients' information safe.

If you are not using the correct authentication, you'll receive a 401 or 403 HTTP error. Merchant server webhooks should use `HTTPS`, and self-signed certificates are not accepted. During

development, it might be helpful using a service such as [ngrok](#) to expose an HTTPS tunnel to your local environment.

02.2_API Keys and Environment

Merchant Keys and the Environment you set are essential for communicating with the Payshop Online Payments platform throughout your payment operation.

Different combinations between the Merchant API Keys or Processing API Keys and your Account ID grant authentication in your requests. It is important to be well aware of when to use each of the keys and how to gain access to them.

Environments can be set to `TEST` or `LIVE` depending on where you are at with your operations. If you are looking to test new features or still going through onboarding, use `TEST`. When your operations are up and running, `LIVE` should be the standard. You can also find these account details on the Dashboard Settings screen if you are a Dashboard user, which we recommend for ease of access and a user-friendly work environment.

- **Public** **Key**
Used in requests coming from the client-side, to create `Dynamic Forms` and `Instruments`.
- **Account ID + Private Key**
Used in requests coming from the merchant server, to create `Charges`, `Payments`, `Reversals`, and `Refunds`, as well as accessing management APIs. **This key SHOULD NEVER BE SHARED in client-side code.**
- **Environment**
`TEST` or `LIVE`, corresponding to `https://switch-processing.teya.com/v2/` and `https://switch-processing.teya.com/v2/`, respectively. The v2 part of the URL corresponds to API versioning.

GET `/v1/merchants/{merchantId}/keys`

Get API keys corresponding to a specific merchant.

Result Parameter

collection

Array

A list contains all the API key objects.

id String

Unique identifier for the API key.

active Boolean

Whether the key is currently active.

authorization_group JSON Object

The authorization group associated with the key.

description String

The key's description.

key String

The API key.

REQUEST

```
$ curl GET https://switch-gateway.teya.com/v1/merchants/{merchantId}/keys -u accountId:apiKey
```

RESULT: HTTP 200

```
{
  "collection": [
    "...",
    {
      "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
      "active": true,
      "authorization_group": null,
      "description": "Default Keys",
      "key": "3eva1vGdMdg0GNLiArtNcXL4WgQtL4sJLCplXzffCyKRZJCQYSIPUewelaJJdQ4"
    },
    "...",
  ]
}
```

POST /v1/merchants/{merchantId}/keys

Create API Keys.

Request Parameter

active Boolean

Whether the key is currently active or not.

description String

The key's description.

authorization_group String

The authorization group associated with the key.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants/{merchantId}/keys -u accountId:apiKey -d '{
  "active": true,
  "description": "Default Keys",
  "authorization_group": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954" }'
```

Result Parameter

id String

Unique identifier for the API key.

active Boolean

Whether the key is currently active or not.

authorization_group JSON Object

The authorization group associated with the key.

description String

The key's description.

key String

The API key.

RESULT: HTTP 201

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "active": true,
  "authorization_group": null,
  "description": "Default Keys",
  "key": "3eva1vGdMdg0GNLiArtNcXL4WgQtL4sJLCplXzfFCyKRZJCQYSIPUewelaJJdQ4"
}
```

Create **API** **key** **without** **required** **fields**

It is not possible to create API Keys without registering all the required fields. Tending to this error case, a list with all the invalid fields and the respective errors is returned.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants/{merchantId}/keys -u accountId:apiKey -d '{ "active": true
}'
```

Result Parameter

message String

String with the error details, in this case: "Invalid parameters".

parameters JSON Object

A list with all the invalid fields and the respective errors.

RESULT: HTTP 201

```
{
  "message": "Invalid parameters",
  "parameters": {
    "description": [
      "This field is required."
    ],
    "authorization_group": [
      "This field is required."
    ]
  }
}
```

02.3_API key ID

It is possible to search API Keys through their respective id's. The API key ID is also a resource to consider when you are looking to make changes to existing API Keys and the ways in which you identify them.

GET `/v1/merchants/{merchantId}/keys/{apiKeyId}`

Get details on the Merchant API key using its id.

Result Parameter

id	String
-----------	--------

Unique identifier for the API key.

active	Boolean
---------------	---------

Whether the key is currently active or not.

authorization_group	JSON Object
----------------------------	-------------

The authorization group associated with the key.

description	String
--------------------	--------

The key's description.

key	String
------------	--------

The API key.

REQUEST

```
$ curl -vX GET https://switch-gateway.teya.com/v1/merchants/{merchantId}/keys/{apiKeyId} -u accountId:apiKey
```

RESULT: HTTP 200

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "active": true,
  "authorization_group": null,
  "description": "Default Keys",
  "key": "3eva1vGdMdg0GNLiArtNcXL4WgQtL4sJLCplXzfFCyKRZJCQYSIPUewelaJJdQ4"
}
```

PATCH /v1/merchants/{merchantId}/keys/{apiKeyId}

Get details on the Merchant API key using its id.

Request Parameter

active Boolean

Whether the key is currently active or not.

description String

The key's description.

authorization_group JSON Object

The authorization group associated with the key.

REQUEST

```
$ curl -vX GET https://switch-gateway.teya.com/v1/merchants/{merchantId}/keys/{apiKeyId} -u accountId:apiKey
```

Result Parameter

id String

Unique identifier for the API key.

active Boolean

Whether the key is currently active or not.

authorization_group JSON Object

The authorization group associated with the key.

description String

The key's description.

key String

The API key.

RESULT: HTTP 200

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "active": true,
  "authorization_group": null,
  "description": "Default Keys",
  "key": "3eva1vGdMdg0GNLiArtNcXL4WgQtL4sJLCpIXzfFCyKRZJCQYSIPUewelaJJdQ4"
}
```

DELETE /v1/merchants/{merchantId}/keys/{apiKeyId}

Delete a Merchant API key using its id.

REQUEST

```
$ curl -vX DELETE https://switch-gateway.teya.com/v1/merchants/{merchantId}/keys/{apiKeyId} -u  
accountid:apiKey
```

02.4_Languages

The Payshop Online Payments platform APIs support calls in multiple languages. This means that you can set up the desired language to be applied in the responses. To do so, you only need to include the respective language code in the HTTP header `Accept-Language`.

In case the target language is not available, the response will be generated in the default en-US. The following table includes our currently available language options.

Language	Header
English (USA)	<code>Accept-Language: 'en-US'</code>
Portuguese (Portugal)	<code>Accept-Language: 'pt-PT'</code>
Portuguese (Brazil)	<code>Accept-Language: 'pt-BR'</code>

02.5_Navigation and Pagination

When making requests to APIs that might generate multiple pages of results, only the first page is presented by default. This is to ensure there is no decline in performance when handling substantial volumes of information.

If you intend to access responses from other pages, mind the following procedure using as an example the `GET v1/merchants/{id}/children` endpoint which fetches the list of sub-merchants associated with a specific merchant.

GET `/v1/merchants/{id}/children`

Method	Path	Description
GET	Sandbox https://switch-gateway.teya.com/v1/merchants/{id}/children Production https://switch-gateway.teya.com/v1/merchants/{id}/children	List available sub-merchants.

Request Parameter

page Number **Required**

The page you want to consider for your list of results.

REQUEST EXAMPLE

```
$ curl -vX GET https://switch-gateway.teya.com/v1/merchants/{id}/children?page=3?merchant_id=accountId -u accountId:APIKey
```

RESPONSE EXAMPLE

```
{
  "collection": [
    {
      "account_id": "OkJ4tL0eJultLfDzGjwnDxVr6xE5HJcXw8oUbvaUYMracs",
      "account_type": "test",
      "name": "Candy shop",
      "metadata": {
        "address": "Braga"
      },
      "approved": true,
      "created_at": "2020-10-23T15:00:02.090528+00:00",
      "updated_at": "2020-10-23T17:45:52.942176+00:00",
    }
  ]
}
```

```

    "whitelabel_settings": "63f6d61deacfe8579e6499d3b62b92b959ad335b5f931679"
  },
  {
    "account_id": "ofAB1Ec0LBEv8tXtKyZrBCVhMAMtiFIV9fO6SNAzNgDK51fLnov",
    "account_type": "test",
    "name": "test dispute",
    "metadata": {
      "address": "place"
    },
    "approved": true,
    "created_at": "2020-10-08T09:56:09.071632+00:00",
    "updated_at": "2020-10-08T09:56:09.071658+00:00"
  },
  {
    "account_id": "cvG0nPy1QYpZx6VyNCrZ5DaxZVwqOxi2PS3RedeVuVrVzrrbE",
    "account_type": "test",
    "name": "Demo",
    "metadata": {
      "address": "Porto"
    },
    "approved": true,
    "created_at": "2020-09-23T11:00:12.459745+00:00",
    "updated_at": "2020-09-23T11:00:12.459787+00:00"
  }
],
"filters": {},
"pagination": {
  "page": 3,
  "per_page": 30,
  "total_pages": 3,
  "total_items": 3
}
}

```

02.6_Merchant and Sub-Merchants Accounts

As we have seen, the business account holds information that is fundamental to your communications with the Payshop Online Payments platform and the success of your transactions. Setting up a business account with extensive information that mirrors your company is important to keep your payment operation up and running.

The main merchant is responsible for onboarding each individual sub-merchant. The onboarding process consists of creating a business account on behalf of the sub-merchant. The main Merchant needs only to be aware of the parent account and register its id.

Merchants accounts can be set to TEST or LIVE. You can choose how to label the accounts and the hierarchy set between merchant and sub-merchants. When creating a merchant account, you can also register White Label settings, which grant for the customization of your workspace in case you are using the Dashboard.

02.6.1_Merchants

The available operations in regards to Merchants include searching for Merchants by id and creating Merchants from scratch. It is also possible to make changes to existing Merchant accounts, as well as delete them.

POST /v1/merchants

Create Merchant.

Request Parameters

account_type	String
The type of the account ("live" or "test").	

name	String
The Merchant's name.	

address	String
Physical location for the Merchant or sub-merchant account being created.	

parent	String (Optional)
If we are creating a sub-merchant, set the parent's account ID here.	

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants -u accountId:apiKey -d '{
  "account_type": "test",
  "name": "ACME Corp.",
  "parent": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954"
}'
```

Result Parameters

account_id	String
-------------------	--------

Unique identifier for the Merchant.

account_id	String
-------------------	--------

The type of the account ("live" or "test").

name	String
-------------	--------

The Merchant's name.

children	Array
-----------------	-------

sub-merchants of this account.

RESPONSE: HTTP 201

```
{
  "account_id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "account_type": "test",
  "name": "ACME Corp.",
  "children": []
}
```

Create Merchant without required fields

It is not possible to create Merchants without registering all the required fields. Tending to this error case, a list with all the invalid fields and the respective errors is returned.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants -u accountId:apiKey -d '{
  "name": "ACME Corp."
}'
```

Result Parameters

message String

String with the error details, in this case "Invalid Parameters".

parameters JSON Object

A list with all the invalid fields and the respective errors.

RESPONSE: HTTP 400

```
{
  "message": "Invalid parameters",
  "parameters": {
    "account_type": [
      "This field is required."
    ]
  }
}
```

02.6.2_Merchant ID

It is possible to get details on Merchants using their respective `id`. The Merchant `id` is also a resource to consider when you are looking to make changes to existing Merchant accounts.

GET `/v1/merchants/{id}`

Get merchant details using this respective id.

Result Parameters

account_id String

Unique identifier for the Merchant.

account_id String

The type of the account ("live" or "test").

name String

The Merchant's name.

children Array

sub-merchants of this account.

REQUEST

```
$ curl GET https://switch-gateway.teya.com/v1/merchants/{id} -u accountId:apiKey
```

RESPONSE: HTTP 200

```
{
  "account_id": "<account_id>",
  "account_type": "test",
  "name": "ACME Corp.",
  "archived": false,
  "children": []
}
```

PATCH /v1/merchants/{id}

Make changes to merchant details using its id

Result Parameters

account_id String
Unique identifier for the Merchant.

account_id String
The type of the account ("live" or "test").

name String
The Merchant's name.

children Array
sub-merchants of this account.

REQUEST

```
$ curl -vX PATCH https://switch-gateway.teya.com/v1/merchants/{id} -u accountId:apiKey -d '{
  "name": "ACME"
}'
```

RESPONSE: HTTP 200

```
{
  "account_id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "account_type": "test",
  "name": "ACME Corp.",
  "children": []
}
```

DELETE /v1/merchants/{id}

Delete merchant

REQUEST

```
$ curl -vX DELETE https://switch-gateway.teya.com/v1/merchants/{id} -u accountId:apiKey
```

02.6.3_Sub-merchants

The available operations in regard to sub-merchants include searching sub-merchants using their `id`, making changes to existing sub-merchant Accounts and deleting sub-merchants. sub-merchants are children to Merchant accounts. Therefore it is important to keep in mind that a sub-merchant always has a `parent` Merchant Account.

GET /v1/merchants/{id}/children

Get merchant details using this respective `id`.

Result Parameter

collection Array

A list that contains all the merchant objects.

account_id String

Unique identifier for the merchant.

account_type String

The type of account (“live” or “test”)

name String

The Merchant’s name.

children Array

Sub-merchants of this account.

REQUEST

```
$ curl GET https://switch-gateway.teya.com/v1/merchants/{id}/children -u accountId:apiKey
```

RESPONSE: HTTP 200

```
{
  "collection": [
    "...",
    {
      "account_id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
      "account_type": "test",
      "name": "ACME Corp.",
      "children": []
    },
    "..."
  ]
}
```

POST /v1/merchants/{id}/children

Get merchant details using this respective `id`.

Result Parameter

collection Array

A list that contains all the merchant objects.

account_id String

Unique identifier for the merchant.

account_type String

The type of account ("live" or "test")

name String

The Merchant's name.

children Array

Sub-merchants of this account.

02.7_Users and Authorization Groups

The way you set up authorizations groups and users defines the dynamics of your team when using the Payshop Online Payments platform's APIs. Make sure you pay close attention to your users, keep the permissions up to date.

02.7.1_Users

The available operations for the Users resource are `GET`, and `POST`.

GET /v1/merchants/{merchantId}/users

Get merchant details using this respective id.

Result Parameter

collection Array
A list that contains all the User objects.

id String
Unique identifier for the user.

email String
User's email address.

last_login String
Date of last login.

global_permissions JSON Object

Global permissions for the user.

merchant_permissions JSON OBJECT

Merchant's permissions for the user.

own_merchants_permissions JSON OBJECT

Merchant's permissions only for authorization groups linked to the User.

tfa_via_app_enabled Boolean

Whether the user has two factor authentication enabled for this account.

merchants Array

Merchants the user has access to..

own_merchants_permissions Array

Authorization groups linked with this user.

REQUEST

```
$ curl GET https://switch-gateway.teya.com/v1/merchants/{merchantId}/users -u accountId:apiKey
```

RESPONSE: HTTP 200

```
{
  "collection": [
    "...",
    {
      "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
      "email": "john@acme.com",
      "last_login": "2019-11-07 11:52:11",
      "name": "John Doe",
      "global_permissions": {},
      "merchants_permissions": {
        "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954": {
          "com.switchpayments.dashboard.transactions_read": true,
          "com.switchpayments.dashboard.refund": true
        }
      }
    }
  ],
}
```

```

"own_merchants_permissions": {
  "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954": {
    "com.switchpayments.dashboard.transactions_read": true,
    "com.switchpayments.dashboard.refund": true
  }
},
"tfa_via_app_enabled": false,
"merchants": [],
"authorization_groups": null
},
"..."
]
}

```

POST /v1/merchants/{merchantId}/users

Request Parameter

email	Array
The user's email address.	

name String
The user's name.

Result Parameter

id	String
Unique identifier for the user.	

email String
User's email address.

last_login String
Date of last login.

name String
User's name.

global_permissions JSON Object

Global permissions for the user.

merchant_permissions JSON OBJECT

Merchant's permissions for the user.

own_merchants_permissions JSON OBJECT

Merchant's permissions only for authorization groups linked to the User.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants/{merchantId}/users -u accountId:apiKey -d '{
  "email": "john@acme.com",
  "name": "John Doe"
}'
```

RESPONSE: HTTP 200

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "email": "john@acme.com",
  "last_login": null,
  "name": "John Doe",
  "global_permissions": {},
  "merchants_permissions": {},
  "own_merchants_permissions": {},
  "tfa_via_app_enabled": false
}
```

Create User without required fields

It is not possible to create Users without registering all the required fields. Tending to this error case, a list with all the invalid fields and the respective errors is returned.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants/{merchantId}/users -u accountId:apiKey -d '{
  "name": "John Doe"
}'
```

Response Body Parameters

message string

String with the error details, in this case "Invalid Parameters".

parameters JSON object

A list with all the invalid fields and the respective errors.

RESULT: HTTP 400

```
{
  "message": "Invalid parameters",
  "parameters": {
    "email": [
      "This field is required."
    ]
  }
}
```

02.7.2_User ID

The available operations for the User ID resource are PATCH and DELETE.

POST /v1/merchants/{merchantId}/users/{userId}

Request Parameter

permissions JSON object

Key value object of the permissions the User should or should not have.

authorization_groups Array (Optional)

List of Authorization Groups `id` the User should be associated with. This `PATCH` will remove any Groups not present in this list, in case the User is associated with them.

REQUEST

```
$ curl -vX PATCH https://switch-gateway.teya.com/v1/merchants/{merchantId}/users/{userId} -u accountId:apiKey -d '{
  "permissions": {
    "refund": true,
    "routing_management": true,
    "read_processing_api_keys": false
  },
  "authorization_groups": [
    "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
    "674ae7bhn83927h5cf6hu87d8hfoiuhudspwef8429jlsdf"
  ]
}'
```

Result Parameter

id	string
Unique identifier for the User.	

email	string
The User's email address.	

last_login	date
Date of last login.	

name	string
User's name.	

global_permissions	JSON object
Global permissions for the User.	

merchants_permissions	JSON object
Merchant permissions for the User.	

own_merchants_permissions JSON object

Merchant's permissions only for Authorization Groups linked to the User.

tfa_via_app_enabled boolean

Whether the user has Two Factor Authentication enabled for this account or not.

merchants array

Merchants the User has access to.

authorization_groups array

Authorization Groups linked to this User.

RESULT: HTTP 200

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "email": "john@acme.com",
  "last_login": "2019-11-07 11:52:11",
  "name": "John Doe",
  "global_permissions": {},
  "merchants_permissions": {},
  "own_merchants_permissions": {},
  "tfa_via_app_enabled": false,
  "merchants": [],
  "authorization_groups": null
}
```

DELETE /v1/merchants/{merchantId}/users/{userId}

REQUEST

```
$ curl -vX DELETE https://switch-gateway.teya.com/v1/merchants/{merchantId}/users/{userId} -u accountId:apiKey -d '{
  "name": "ACME"
}'
```

02.7.3_Authorization Groups

The available operations for the Authorization Groups resource are [GET](#), and [POST](#).

GET /v1/merchants/{merchantId}/authorization-groups

REQUEST

```
$ curl GET https://switch-gateway.teya.com/v1/merchants/{merchantId}/authorization-groups -u accountId:apiKey
```

Result Parameter

collection Array

A list that contains all the Authorization Group objects.

id String

Unique identifier for the Authorization Group.

type String

The Authorization Group's type.

name String

The Authorization Group's name.

permissions JSON object

The Authorization Group's permissions.

RESULT: HTTP 200

```
{
  "collection": [
    "...",
    {
      "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
      "type": "merchant",

```

```
"name": "Analysts Permissions",
"permissions": {
  "refund": true,
  "routing_management": true,
  "read_processing_api_keys": false
}
},
"..."
]
```

POST /v1/merchants/{merchantId}/authorization-groups

Request Parameter

name String

The name given to the Authorization Group.

permissions JSON Object The Authorization Group's permissions.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants/{merchantId}/authorization-groups -u accountId:apiKey -d '{
  "name": "Analysts Permissions",
  "permissions": {
    "refund": true,
    "routing_management": true,
    "read_processing_api_keys": false
  }
}'
```

Result Parameter

id String

Unique identifier for the Authorization Group.

type String

The Authorization Group's type.

name String

The Authorization Group's name.

permissions JSON Object

The Authorization Group's permissions.

RESULT: HTTP 201

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "type": "merchant",
  "name": "Analysts Permissions",
  "permissions": {
    "refund": true,
    "routing_management": true,
    "read_processing_api_keys": false
  }
}
```

EXAMPLES

Create Authorization Group without required fields

Create Authorization Group without required fields.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/merchants/{merchantId}/authorization-groups -u accountId:apiKey -d '{
  "name": "Analysts Permissions"
}'
```

Result Parameter

message

String

String with the error details, in this case "Invalid Parameters".

parameters JSON Object

A list with all the invalid fields and the respective errors.

RESULT: HTTP 400

```
{
  "message": "Invalid parameters",
  "parameters": {
    "permissions": [
      "This field is required."
    ]
  }
}
```

02.7.4_Authorization Group ID

The available operations for the Authorization Group ID resource are [GET](#), [PATCH](#), and [DELETE](#).

GET /v1/merchants/{merchantId}/authorization-groups/{authorizationGroupId}

REQUEST

```
$ curl GET https://switch-gateway.teya.com/v1/merchants/{merchantId}/authorization-groups -u accountId:apiKey
```

Result Parameter

id String

Unique identifier for the Authorization Group.

type String

The Authorization Group's type.

name String

The Authorization Group's name.

permissions JSON Object

The Authorization Group's permissions.

RESULT: HTTP 200

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "type": "merchant",
  "name": "Analysts Permissions",
  "permissions": {
    "refund": true,
    "routing_management": true,
    "read_processing_api_keys": false
  }
}
```

PATCH /v1/merchants/{merchantId}/authorization-groups/{authorizationGroupId}

REQUEST

```
$ curl -vX PATCH https://switch-gateway.teya.com/v1/merchants/{merchantId}/authorization-
groups/{authorizationGroupId} -u accountId:apiKey -d '{
  "name": "Analysts Permissions",
  "permissions": {
    "refund": true,
    "routing_management": true,
    "read_processing_api_keys": false
  }
}'
```

Request Parameter

name

String

The Authorization Group's name.

permissions JSON Object

The Authorization Group's permissions.

RESULT: HTTP 200

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "type": "merchant",
  "name": "Analysts Permissions",
  "permissions": {
    "refund": true,
    "routing_management": true,
    "read_processing_api_keys": false
  }
}
```

Result Body Parameters

id String

Unique identifier for the Authorization Group.

type String

The Authorization Group's type.

name String

The Authorization Group's name.

permissions JSON Object

The Authorization Group's permissions.

DELETE

`/v1/merchants/{merchantId}/authorization-groups/{authorizationGroupId}`

REQUEST

```
$ curl -vX DELETE https://switch-gateway.teya.com/v1/merchants/{merchantId}/authorization-
groups/{authorizationGroupId} -u accountId:apiKey
```

Next

Steps

Now that you have gone through the Merchant starting kit, head over to the Integrating with the Payshop Online Payments Platform to learn more about the integration patterns available and choose the one that best suits your needs.

03_Permissions

Permissions in the Payshop Online Payments platform are normally associated with an authorization group. You can add new permissions or edit existing ones by tapping into Settings > Security and Users in the Dashboard. You should be able to find the permissions in question under Authorization Groups.

Each authorization group consists of a list of specific users selected from the registered users in your team. Therefore, it's important to consider which authorizations are necessary for each staff member when you set up your teams.

- Can these permissions allow for more than what the user needs?
- Can these permissions allow for more than what the user needs?
- Can the user share these permissions with someone that might not be aware of the consequences of bad usage?
- Does the permission expose the user to unwanted information?
- Can another permission provide the same benefit with less added risk?

Analytics

Having permissions to Analytics, the user will be able to access the Analytics tab, which is available in the Dashboard navigation bar.

Transactions

Enabling Transactions for an authorization group or specific users grants access to the list of transactions in the Transactions tab of the Dashboard. To be able to perform actions on said transactions, users need further permissions, as described below.

Permission	Description
Create Charge	The user can create charges. This action is performed via API and also using the Dashboard. With this permission the user is able to initiate checkouts using the Create Checkout button in the Dashboard.
Create Instrument	The user can create instruments. This action is performed via API.
Create Refund	The user is able to refund payments. It allows for whole and partial refunds. Refunds can never exceed the total amount paid. Refunds can be performed via API or through the Dashboard by hitting the Refund button.
Create Reversal	The user is able to void transactions. It can be performed via API and also by hitting Void on the Dashboard.

Channels

Enabling Channels for an authorization group allows the respective users to access the Channels tab in the Dashboard. This grants them access to the list of available channels and routing rules.

Permission	Description
Activate Channels	The user can enable and disable channels.
Configure Channels	The user is able to edit channel details and keys.
Manage Routing Rules	The user is able to create routing rules and also edit and delete existing ones.

Events

Enabling Events for an authorization group allows the respective users to access the Events tab in the Dashboard. This allows them to explore listed events and all of their details.

Permission	Description
Resend	The user is able to resend event communications to businesses.

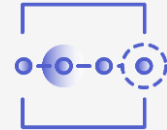
Manage Destinations

The user can configure events' delivery.

Audit log?

The audit log registers changes made to elements in the Payshop Online Payments platform.

Every time a given resource is created, updated, or deleted, an audit log event is created. Audit logs include the information being altered and also when and by whom the changes were made.



Event Schema

Description

Authorization
Audit Log

Group

Describes changes made to authorization groups.

Charge Audit Log

Describes changes made to a charge element.

Destination Audit Log

Describes changes made to destinations.

Events Checker Report

Contains all the failed dispatches of the last 24 hours.

Lifecycle Event	Describes each transaction.
Business Audit Log	Contains changes made to a business account.
Report Status	Contains the status of a report. Every time the state of a report changes an event with details on that report and its status is generated.
Settlements Batch	Describes settlement batches.
Settlements Source Group	Contains changes made to source groups.
Terminal Audit Log	Registers changes made to terminals
Terminal Session	Registers the end of a terminal session and it includes details on said session.

Reporting

Enabling Reporting for an authorization group allows the respective users to access the Reporting tab in the Dashboard.

Permission	Description
Manage Reporting	The user is able to create, read, edit, and delete reports, schedules, and reporting templates.

Settings

Under Settings you can find permissions to make changes to the configurations of user and business accounts. Note that some of these permissions are only available for staff members.

Permission	Description
Manage Users	The user is able to create and manage user accounts in the current business account, it does not include sub-business accounts. Please note that a user with this permission is able to edit the permissions of any other user including their own.
Read Processing API Keys	Ability to read Processing API keys.
Manage Processing API Keys	Ability to generate, edit, and delete Processing API keys.

Read Authorization Groups

The user can read authorization groups on that account only.

Manage Authorization
Groups

The user is able to generate and make changes to authorization groups in the current business account, it does not include sub-business accounts. This permission is also required when cloning authorization groups.

User and Business

Under User and Businesses you can find permissions to make changes to the configurations of user accounts and business account environments. Note that User permissions are only made available to staff members.

Permission		Description
Manage Test Accounts	Business	Ability to create and edit test sub-accounts, create and edit users from test sub-accounts, and create and edit authorization groups from test sub-accounts.
Manage Live Accounts	Business	Ability to create and edit live sub-accounts, create and edit users from live sub-accounts, and create and edit authorization groups from live sub-accounts.

04_Integrating

Every stakeholder, one integration.

With the constant evolutions in the payments landscape, ensuring that your payment operation supports current and future Payment Methods is paramount. By integrating the Payshop Online Payments platform, you get access to every present and future Payment Method, as well as all the tools required to run a professional and comprehensive payments operation.

Warning: For online payments regarding Debit/Credit Cards, the only integration pattern Payshop allows is *Dynamic Forms with iFRAME=true*. In this payment method, the other two integration patterns (*Hosted Checkout* and *REST*) are not allowed to be implemented by the development Team integrating with Payshop Online Payments. Obviously, this rule does not apply to the *Plugins* that Payshop provides.



Dynamic Forms

Add new Payment Channels without any changes to your code.



Hosted Checkout

Streamline your checkout process.



REST Integration

Enable custom Transaction Flows.

The Payshop Online Payments platform's architecture was designed with a single integration flow in mind. Businesses are expected to integrate with the platform once and be able to process any payment methods, without any changes to their code. This reduces development effort and maintenance costs, whilst also improving your time to market. To better understand how to conduct integration access the following resources.



Transaction Flows

Understand the execution flows of your Transactions.



Event Handling

Deconstruct your Transactions and keep track of their statuses.



Error Codes

Check out the codes and descriptions of all errors that may occur when you communicate with the Payshop Online Payments platform APIs.

There are three possible integration patterns, which offer differing degrees of liberty to businesses. These are Dynamic Forms, Hosted Checkout, and REST Integration.

What is the right integration for me?

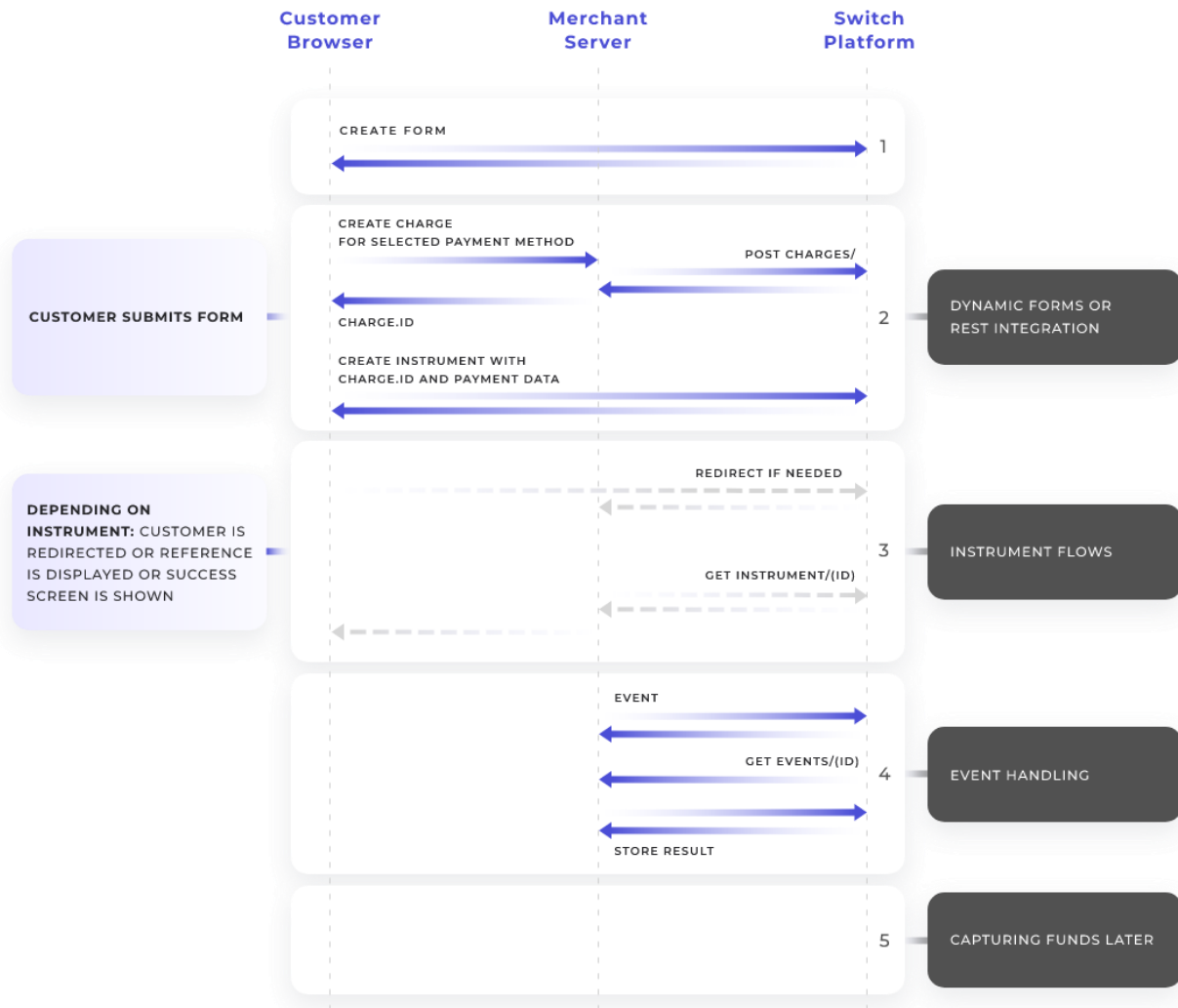
We recommend choosing Dynamic Forms, as this option provides an efficient, out-of-the-box solution that requires low integration effort and supports a wide variety of use-cases. Our REST API integration is recommended for singular scenarios where Dynamic Forms are unable to comply with your business requirements.

In order to better understand which integration pattern would be better suited to you, please have a look at the table below, which compares these two patterns from a requirements perspective.

	Dynamic Forms	Hosted Checkout	REST Integration
Development Effort	Low	Low	High
Maintenance Effort	Low	Low	High
UI Library	Yes	Yes	No
Customizable Look & Feel	Yes	Yes	Yes

Customizable Transaction Flow	No	No	Yes
Web & Mobile	Yes	Yes	Yes
Native Apps	No	No	Yes
PCI-DSS	Filled in and signed SAQ A	Filled in and signed SAQ A	Filled in and signed SAQ A-EP along with quarterly scans by an ISV

There are several steps to take full advantage of the Payshop Online Payments platform's processing capabilities. Not all the steps are required to process payments, but it is good to have the full picture before making your integration options. First, it is important to go through the steps involved in how we help our clients handle transactions.



1. **Display** **Form**
 The form through which the customer chooses a payment method can be created by you. But keep in mind that if you use Dynamic Forms to do it, your form will adapt to future payment methods without requiring additional development.
2. **Create** **Charge** **and** **Instrument**
 These are the minimum steps required to process a transaction using the Payshop Online Payments platform.
3. **Instrument** **Flows**
 Handle methods that require redirection or displaying references.
4. **Event** **Handling**
 How your server keeps track of transactions.
5. **Capturing** **Funds** **Later**
 It applies to recurring or auth-capture methods.

Elements of a Transaction

Every Transaction, regardless of the payment method, has the same Elements.

Parameter	Description
Charge	The Merchant's intent to initiate a Payment Flow. In this step, the Payment Method and Provider are chosen and configured by the Merchant.
Instrument	The Customer authorizes the Transaction by sending Payment details.
Payment	The fund transfer per se.
Reversal	Element used for reversing a capture or Refund that has not yet been cleared, or for voiding an authorization.
Refund	Returning the funds to the customer. This only applies to refundable Payment Methods

Instrument Properties

Instruments from different Payment Methods have specific properties that define how they can be used.

Parameter	Description
Captured on Creation	A Payment is created automatically when, on the client-side, an Instrument is created.
Recurring	Multiple Payments can be created from a Recurring Instrument.
Refundable	Refunds can be created.
Redirect	The customer must be redirected to an external page to complete the Transaction. This process is handled automatically by the Dynamic Forms and JS lib.

Reference

The customer will be shown a payment reference to push the funds to.

Dynamic Forms

The main aim of our Dynamic Forms solution is to provide a flexible UI for web, mobile and POS, that is able to process any current and future Payment Methods without requiring any changes to the code.

Dynamic Forms is composed of a Javascript Client-Side Library that can be easily inserted into your checkout page. It will be responsible for rendering the UI and handling the required communication with the Payshop Online Payments platform to execute any type of Transaction.



Tailor-Made

With Dynamic Forms, the customization of your checkout process is merely one click away. This integration grants you an array of different Payment Methods that can be made available to your Customers immediately and without fuss.



Future Proof

Using Dynamic Forms, the work required to stay up to date is offloaded from your team. This integration option dynamically supports new payment options via communication with the Payshop Online Payments platform, without requiring modifications.



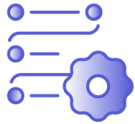
Flexible

Dynamic Forms automatically prepares your website to collect different data points and support different user interaction flows, whilst allowing you to customize the look and feel.



Fast

All the technical complexity is handled by a single client-side library that is responsible for user interaction and communication with the Payshop Online Payments platform.



Easy to Maintain

By leveraging server-side communication to dynamically adapt the user interface and associated experience, Dynamic Forms support any current or future Payment Methods, thus reducing the maintenance required to stay up to date.



Simplified Compliance

Uses by default an iframe configuration, which ensures the customer Payment data never touches your website. From a PCI-DSS compliance perspective, this reduces the burden required to process payments.

REST Integration (not allowed for cards payment method)

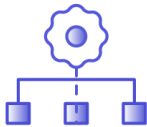
Whilst having a turnkey solution to process payments on your website can be very useful, there might be scenarios where the customizations needed require a bespoke solution. If you need to support different platforms, namely web, native mobile apps, and IoT devices, or are looking for a customized user experience with additional steps, REST Integration might be the answer for you.

Our REST Integration leverages the Payshop Online Payments platform API endpoints to orchestrate Transaction execution using a host-to-host communication. You become responsible for rendering the forms and handling user interaction, allowing you to tailor the components to your design and user experience guidelines.

What about new Payment Methods?

The REST API has an endpoint that specifies which components the UI should have, given the Payment Methods active at the moment.





Custom Transaction Flows

The number of steps and the input collected from the user to execute a Transaction can be personalized, as long as the three required API calls are executed in between.



Interoperability

Given that this pattern does not rely on a client-side library to collect data, it has a broader reach by being able to support any frontend language, application, or platform where it is possible to execute a REST API call. One such example would be a native mobile application for Android or iOS.



Bespoke User Experience

This integration option allows you to set up the UI according to your specifications, ensuring a consistent design language throughout your application and following your preferred user interaction patterns.




What about Compliance

REST Integration requires PCI DSS compliance, which means having servers scanned on a quarterly basis by an Approved Scanning Vendor (ASV), and at least a filled and signed SAQ A-EP.



Next Steps

Now that we have outlined the existing integration patterns with the Payshop Online Payments platform, feel free to jump over to your integration of choice.

 <p>Dynamic Forms Add new Payment Channels without any changes to your code.</p>	 <p>Hosted Checkout Streamline your checkout process.</p>	 <p>REST Integration Enable custom Transaction Flows.</p>
--	---	---

04.1_Dynamic Forms

Integrating with the Payshop Online Payments Platform

Dynamic Forms is composed of a Javascript client-side library. The library can be easily included in your checkout page and will be responsible for rendering the UI and handling the required communication with the Payshop Online Payments platform to execute any type of transaction.

The Payshop Online Payments platform enables real-time transactions on an any-to-any basis, be it multi-channel, multi-network, multi-method, or multi-currency. This abstraction feeds a shared, event-based database on top of which runs multiple internal applications and external value-added services that make up the Platform Components.

Integrating with the Payshop Online Payments platform requires code both on:

- The client-side, to create the form that collects customer payment data.
- The server-side, to create Charges. This step is required to authenticate and avoid fraudulent requests.

For Dynamic Forms you should consider the following order of instructions:

1. Displaying a form
2. Creating a Charge
3. Client Library Reference

After the Dynamic Forms integration, you will be able to add new Payment Channels without any changes to your code.

04.1.1_Display Form

Each Payment Method requires specific input fields and actions from the customer, and Dynamic Forms let you unify all these use cases in a single integration. Subsequently, enabling additional Payment Channels and configuring Dynamic Routing and Risk management rules is possible without having to change the integration.

To create a Dynamic Form, instantiate the client library with the environment and your public key, and call the dynamicForms over a container in your page.

Dynamic Forms can be added to your application through a quick and simple process that requires three steps from an integration perspective:

1. Adding an HTML container;
2. Loading our client-side library; and
3. Initializing the Dynamic Forms.

The following example shows how you can accomplish these three steps in your checkout page.

Checkout Page

```
<html>
<body>

  <!-- Step 1 -->
  <!-- DIV container that will be used to render the Dynamic Forms -->
  <!-- The UI will adapt to the space available -->
  <div id="dynamic-forms-container"></div>
</body>
```

```

<!-- Step 2          -->
<!-- Import Client library -->
<script src="switch.js"></script>

<!-- Step 3          -->
<!-- Instatiante the client library -->
<script>
let formContainer = document.getElementById('dynamic-forms-container');
let formOptions = {
  chargesUrl: 'https://your.url.com/charges/',
  merchantTransactionId: '123456',
};

let switchJs = new SwitchJs(SwitchJs.environments.SANDBOX, 'ACCOUNT_PUBLIC_KEY');
switchJs.dynamicForms(formContainer);
</script>
</html>
<html>

```

Add HTML container

Our library requires a DIV container to be present in the page that will be used to render the UI. This HTML container should have a dedicated ID so you can reference it when initializing the Dynamic Forms. In the previous example this is done by the following HTML element: `<div id="dynamic-forms-container"> </div>`.

Load the client-side library

This integration mechanism uses a client-side javascript library to coordinate the execution process. As such, it's loaded in a standard manner as any javascript library: `<script src="switch.js"></script>`



Looking for more information about our Client library?

Head over to the Client Library Reference down below and review all the functions and options available.

Initialize the Dynamic Forms

Our client library provides the functions necessary to load and initialize the UI that will process payments. This is done essentially by performing two function calls, one to instantiate the library itself and another to render the Dynamic Forms

Initializing the Dynamic Forms

```
let formContainer = document.getElementById('dynamic-forms-container');
let formOptions = {
  chargesUrl: 'https://your.url.com/charges/',
  merchantTransactionId: '123456',
};

// 1. Instantiate
let switchJs = new SwitchJs(SwitchJs.environments.SANDBOX, 'ACCOUNT_PUBLIC_KEY');

// 2. Render
switchJs.dynamicForms(formContainer, formOptions);
```

The first function call will initialize the client library by indicating the environment and account that will be used to process transactions: `SwitchJs(SwitchJs.environments.SANDBOX, 'ACCOUNT_PUBLIC_KEY');`

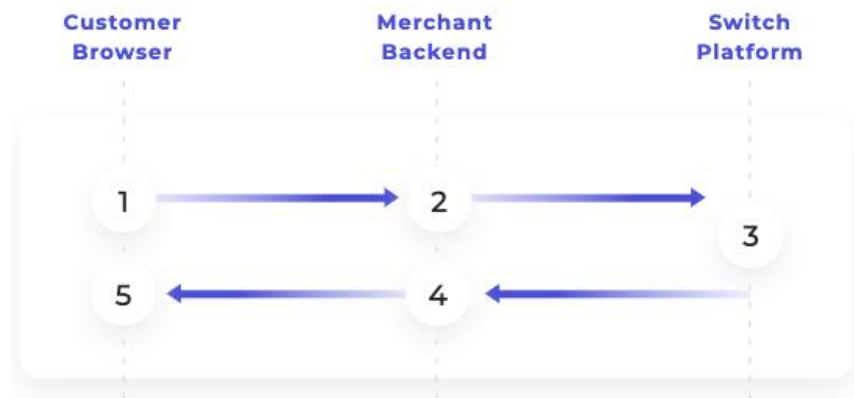
Afterward, it's time to render the Dynamic Forms on your checkout page. To achieve this we call the following function with the reference to the HTML element where the form will be drawn as well as the rendering options: `switchJs.dynamicForms(formContainer, formOptions);`

04.1.2_Create a Charge

After performing the previous steps, the Dynamic Forms have been successfully rendered in your checkout page. The user can now choose the preferred Payment Method towards acquiring the intended goods.

As stated in the Core Concepts section, our platform divides a transaction into three execution-related elements: charge, instrument, and payment. The client-side library will coordinate the creation of these elements, where required, and the associated communications with the Payshop Online Payments platform.

One of such communication steps will be with your backend platform, in order to authenticate the transaction being attempted and effectively create the `charge`. This authentication will be done using your private API key and will ensure that both you and the customer intended to execute the transaction. To this effect, the following steps will be executed.



1. After the customer has selected the intended payment method, the browser calls the merchant backend through the `chargesURL`.
2. The merchant backend system receives a `POST` request containing the `merchantTransactionId` to correlate the transaction being attempted and using your `Private API Key` authenticates the transaction.
3. The merchant backend calls the Payshop Online Payments platform to proceed with creating the `charge`.

4. The Payshop Online Payments platform creates a charge and replies back with the associated `charge` information, including ID.
5. The merchant backend replies back to the original call made in step 1 with the provided `charge_id`.

ChargeURL Call

To execute steps 2 and 3 as previously outlined, which will lead to the creation of the Charge by your backend system, our platform expects your API to comply with the following signature.

Path	Method	Description
Configured by the chargeURL when initializing the Dynamic Forms	POST	This API call is made by the Payshop Online Payments platform and aims to authenticate the Charge being attempted.

Result Parameter

chargeType String	Indicates the Payment Method that has been selected by the customer.
merchantTransactionId String	Parameter that univocally identifies the transaction being attempted. The Payshop Online Payments platform will use the merchantTransactionId that is passed when initializing the Dynamic Form.

REQUEST EXAMPLE

```
$ curl -vX POST https://charge.url
-d '{
  "chargeType": "card_onetime",
  "merchantTransactionId": "506785940"
}'
```

Response Body Params

id String	The ID returned by the Payshop Online Payments platform that uniquely identifies the Charge element for this transaction.
--------------	---

RESPONSE EXAMPLE

```
{  
  "id": "06d7c2e4145f3be209e9ab5c6ed24da8b786f"  
}
```

API Call

As documented previously in steps 4 and 5, in order to generate a Charge element your backend system must contact our API using the private key. Below you can find the endpoint that should be used to execute this step as well as all the required details to perform the call.

POST v2/charges

Path	Method	Description
https://switch-processing.teya.com/v2/charges	POST	Creates a Charge element by using the information passed in the body parameters.

Request Body Params

charge_type String (mandatory)	Indicates the Payment Method that has been selected by the customer, should contain the value passed by our API call.
currency String (mandatory)	ISO 4217 code that indicates the currency that will be used by the transaction.

amount Number	The amount that should be charged to the customer.
(mandatory)	
events_url String	Allows you to configure the URL that will be called by the Payshop Online Payments platform to notify about lifecycle events related to this transaction. Please note that the URL must use HTTPS.
redirect_url String	Specifies the HTTPS URL to where the users should be sent after they authenticated the transaction on the Payment Method page.
(conditional)	This parameter is only required for Payment Methods that require user redirection for authentication, e.g.: Paypal.
instrument_params JSON Object	Specifies configuration parameters that are passed to the Provider when creating the Instrument element and which allow you to configure how the fund transfer is executed.
metadata JSON Object	This object allows you to pass any transaction-related data points that may be useful to be displayed in the Dashboard for analysis purposes.
channels Array	Gives the ability to specify which Channel should be used to process the transaction. Using this parameter you can leverage your business logic to select the Channel rather than relying on Dynamic Routing.
	If multiple values are present a fallback mechanism will be used, giving precedence to the first entries.

REQUEST EXAMPLE

```
-u <merchant id>:<private key>"
-d '{
  "charge_type": "card_onetime",
  "amount": 42,
  "currency": "EUR",
  "events_url": "https://your.url/v1/notificationHandler"
}'
```


Response Body Params

id String	The ID that uniquely identifies the created Charge element for this transaction.
charge_type String	Indicates the Payment Method that will be used to process the transaction.
charge_type_label String	Human-friendly description of the Payment Method that will be used to process the transaction.
currency String	ISO 4217 code that indicates the currency that will be used by the transaction.
amount Number	The amount that will be charged to the customer.
confirmed Boolean	Indicates whether the current Charge element has been confirmed by the merchant using their private key: <ul style="list-style-type: none">• true: the Charge has been confirmed by the merchant.• false: the Charge has not been confirmed by the merchant.
external Boolean	Indicates whether the current Charge element was not created by the Payshop Online Payments platform: <ul style="list-style-type: none">• true: the Charge has not been processed by the Payshop Online Payments platform.• false: the Charge has been created using the Payshop Online Payments platform.
instrument_params JSON Object	The configuration parameters that were specified when creating the Charge element, which allow you to configure how the funds' transfer is executed by the Provider.
events_url String	Contains the URL that will be called by the Payshop Online Payments platform to notify about lifecycle events related to this transaction.
redirect_url String	The URL to where the users should be sent after they authenticated the transaction on the Payment Method page.
channels Array	Documents the Channels that will be used to process the transaction.
metadata JSON Object	The transaction-related data points that were passed when creating the Charge, which may be useful to be displayed in the Dashboard for analysis purposes.
external_ids JSON Object	Documents the Provider ID for the current charge, if available.

request_log JSON Object	Stores information about the device that was used to create the current Charge element, such as country, ip_address, user_agent, and library_version.
created_at String	Indicates the date and time when the current Charge element was created.
updated_at String	Documents the date and time when the last update was performed to the current Charge.

RESPONSE EXAMPLE

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "charge_type": "card_onetime",
  "amount": 10,
  "currency": "EUR",
  "confirmed": false,
  "external": false,
  "instrument_params": {
    "enable3ds": true
  },
  "events_url": "https://merchant.com/events",
  "channels": [{
    "processor": "checkout",
    "id": "85a557e4fdb6c8806f413bc75fabab162828e4f95b8e6390",
    "label": "card_onetime_checkout"
  },
  {
    "processor": "acapture",
    "id": "9fb7b1e253f1c592210b7c37b40b18e576ff30995b8e40de",
    "label": "card_onetime_acapture"
  }
  ],
  "charge_type_label": "Card One-Time",
  "redirect_url": "https://merchant.com/redirect",
  "metadata": {
    "orderId": "1337"
  },
  "external_ids": null,
  "request_log": {
    "country": "PT",
    "ip_address": "100.10.10.10",
```

```
"user_agent": "curl/7.54.0",
"library_version": null
},
"created_at": "2018-06-18T10:28:36.358233+00:00",
"updated_at": "2018-06-18T10:28:36.358258+00:00"
}
```

04.1.3_Recurring Payments

An example of the use given to a recurring payment is the sale of periodic baskets of fruits and vegetables, where the end consumer authorizes the recurring payment for the product on the website. For the merchant to activate the recurring payment, they must instruct the charge at creation that there will be multiple funds captures:

```
{ "charge_type": "card", "amount": 20, "currency": "EUR", "channels": [ "card_checkout_payfac" ],
"instrument_params": { "recurring": true, "capture_on_creation": true } }
```

We emphasize the necessary changes to distinguish this charge from a single capture.

With such a charge created, as soon as the respective instrument is in place (when the client enters their card details in the DynamicForm), sequential payments can be created.

When the merchant sends a new periodic basket of fruits and vegetables to the consumer, they must create a new payment based on the instrument they have saved, and thus a new charge will be made on the card, without the need to bother the consumer.

As required by card processor regulations, the following information must be displayed to cardholders, both on the web page where payment details are requested and entered, and on the checkout screen:

Your name as the merchant
The amount of the transaction (even if it is zero) and the amount and timing of any recurring transactions that will occur
The length of any trial period, introductory offer, or promotional period
How the cardholder will be charged (including showing the last 4 digits of the card)
How to cancel the agreement and/or any subsequent recurring transactions

This information must be shown throughout the payment process, including displaying the information when the cardholder's payment credentials are requested and entered.

04.1.3_Client Library Reference

Our Client library provides the required functionality to initialize and operate Dynamic Forms on your checkout page. Here you can find out more about the available functions and what each can achieve.

SwitchJs(environment, accountPublicKey)

Constructor function to initialize the Client-Side Library

USAGE EXAMPLE

```
let switchJs = new SwitchJs(SwitchJs.environments.TEST, 'ACCOUNT_PUBLIC_KEY');
```

Parameters

environment

String

The platform environment to which the Dynamic Forms Library will connect to process the payments. For ease of use, our library has constants available with the URLs. `SwitchJs.environments.TEST`: constant containing the test environment URL. `SwitchJs.environments.LIVE`: constant containing the production URL.

accountPublicKey String

The Public Key attributed to the account. Typically provided by our Sales Department during the integration process.

The Authorization Group's permissions.

Return Value

Object

The Public Key attributed to the account. Typically provided by our Sales Department during the integration process.

.dynamicForms(formContainer, formOptions)

Renders the Dynamic Form in the UI. Requires the Client-Side Library to be previously initiated.

USAGE EXAMPLE

```
let formContainer = document.getElementById('dynamic-forms-container');
let formOptions = {
  chargesUrl: 'https://your.url.com/charges/',
  merchantTransactionId: '123456',
};

let switchJs = new SwitchJs(SwitchJs.environments.TEST, 'ACCOUNT_PUBLIC_KEY');
switchJs.dynamicForms(formContainer, formOptions);
```

Parameters

formContainer

Object

The object containing the HTML element where the form will be rendered.

formOptions Object

An object containing the initialization options for the Form. Please check below for further details regarding the available parameters.

name String

The Authorization Group's name.

permissions JSON Object

The Authorization Group's permissions.

..dynamicForms(formContainer, formOptions)

These parameters are passed to the `..dynamicForms()` function call and allow you to configure the UI/UX of the form.

USAGE EXAMPLE

```
let formStyle = {
  formHeader: {'display': 'none'},
  formField: {'margin': '10px 0 10px 0'}
};

let formOptions = {
  merchantTransactionId: '123456',
  chargesUrl: 'https://your.url.com/charges/',
  chargeTypes: ['card_onetime', 'multibanco'],
  selectedChargeType: 'card_onetime',
  iframe: true,
  autoRedirect: true,
  showReference: true,
  language: 'en',
  resetStyle: true,
  style: formStyle
};
```

Parameters

autoRedirect

Boolean

Whether customers will be automatically redirected to the provider's payment page, when a payment method requires redirection. Defaults to `true`.

chargeId String

ID of a previously created Charge. Useful when a single payment method is supposed to be displayed, instead of a list of all the available ones. Sending this parameter makes `chargesUrl` and `merchantTransactionId` optional, and not used.

showOptionalFields Boolean

For payment methods that include a reference (information that should be presented to the customers, for them to proceed with the payment, e.g.: multibanco, boleto, pix, ...), this parameter defines whether the Dynamic Forms should handle the display of it or not. Defaults to `true`.

chargeTypes Array

List of charge types that will be available to the customers. By default, all the payment channels enabled on your Merchant account are available. This list works as a filter over that, if you want to display only certain payment methods to a given customer.

language String

Language in which the forms should be displayed. Defaults to the browser set language.

merchantTransactionId String

An ID that identifies the transaction on your end. It will be sent in `chargeUrl` requests. It is required by default, but optional if `chargeId` is specified.

payouts Boolean

An ID that identifies the transaction on your end. It will be sent in `chargeUrl` requests. It is required by default, but optional if `chargeId` is specified.

theme String

Set the theme that will be used. Most customizations should be made using the `customStyles` parameter, but the theme can help with some more radical changes to some parts of the layout. Currently the only available themes are `base` and `material`. The one single difference between them are the form inputs, which have a material-design-like look and feel on the `material` theme. Defaults to `base`.

customStyles Object

This is where you customize the Dynamic Forms to match your website's style. The possibilities are endless.

validateFormOnChange Boolean

Whether form data should be validated immediately when input by the user, or only at the form submission stage. Defaults to `true`.

selectedChargeType String

Charge type that will be initially selected on forms render. Charge types list screen will be skipped, and the payment form for this charge type will be immediately displayed. Other charge types are still available by navigating back to the list.

.on(event, callbackFunction)

Registers a callback function that will be triggered every time a given behavior is performed by the Dynamic Forms. Using these event listeners you can monitor which actions are being performed by the forms and trigger any relevant actions on your side.

USAGE EXAMPLE

```
dynamicForms.on('charge-type-selected', (chargeType) => {
  if (chargeType === 'card_onetime') {
    console.log('Customer chose Card one-time');
  }
});
```


Parameters

event

String

Whether customers will be automatically redirected to the provider's payment page, when a payment method requires redirection. Defaults to true.

callbackFunction

 Function

The callback function that will be executed by the Client Library when the subscribed event is triggered. Depending on the event in question, it will contain at most one variable with relevant information.

Event Example	Description
forms-loaded	The Form has completed rendering on the webpage.
form-data-changed	The user has changed the data inputted in the Dynamic Forms.
form-data-error	There was an error parsing the information submitted in the Form.
charge-success	Object with the created Charge ID. Triggered when Dynamic Forms receive a successful response from chargesUrl.

charge-error	Object with the unsuccessful response returned by chargesUrl.
charge-types-loaded	The list of available charge types has been rendered in the UI.
charge-type-selected	Charge type selected by the customer.
charge-type-canceled	The charge type that was previously selected has been canceled by the user. Triggered when the user goes back to charge type selection list.
instrument-authorized	Instrument details of the successfully created authorized Instrument.
instrument-pending	<p>When an Instrument was successfully created, but it's not authorized yet, you may need to wait for some action from the user.</p> <p>If the instrument has a reference to be displayed, and showReference is disabled, you should display it to the customer. If the instrument requires a redirect, and</p>

autoRedirect is disabled, you should redirect the customer now.

instrument-invalid

This event is triggered when the instrument fails, either because the user input data is wrong, or a provider is down.

submit

Triggered every time the user submits the form.

That's it!

By following these steps, your product is already enabled for many Payment Methods and Providers, and your integration is ready to be tested.

Please contact us if you have any feedback or need more help integrating. We want to make this integration as simple as possible, and depending on your server-side language or CMS, we might even be able to provide you with code samples.

Next Steps

You should create your events_url and handle Transaction Lifecycle Events, to mark transactions as paid in your database.

There are also few additional Transaction Flows you should handle, to truly support every payment method, provider and flow in the world.

04.2_Hosted Checkout (not allowed for cards payment method)

Integrating with the Payshop Online Payments Platform

The Hosted Checkout allows you to redirect your customers from your cart to our checkout when it is time to pay. Hosted Checkout uses Dynamic Forms to make it easier on you to change around Payment Methods as you see fit as well as streamline your checkout process.



Pay by Link

Transfer the Transaction conundrum to us. Redirect Customers to the Hosted Checkout and avoid the Payment setup.



PCI Compliance

With Hosted Checkout your Customer's Payment information never touches your website, you can share your compliance concerns with us.



Alternative Payment Methods

Let your Customers ditch the cards. Allow your clients to use alternative forms of Payment in your terminals through QR codes.

Path	Method	Description
https://checkout.switchpayments.com	POST	Create a hosted checkout page.

HOSTED CHECKOUT EXAMPLE

```
{
```

```

"checkoutParams": {
  "currency": "EUR",
  "showReference": false,
  "cancelUrl": "https://checkout-demo.switchpayments.com/cart",
  "chargesUrl": "https://checkout-demo.switchpayments.com/create-charge",
  "redirectUrl": "https://checkout-demo.switchpayments.com/checkout-complete?",
  "chargeTypes": [
    "afterpay",
    "alipay",
    "boleto",
    "card_onetime",
    "giropay",
    "ideal",
    "klarna_invoice",
    "lotericas",
    "multibanco",
    "mbway",
    "neosurf",
    "ninja_wallet",
    "offline_bank_transfer",
    "p24",
    "paypal",
    "paysafecard",
    "sofort",
    "trustly",
    "yandex"
  ],
  "products": [
    {
      "title": "Sample product 1",
      "reference": "595326123",
      "quantity": 1,
      "price": 14.99,
      "thumbnail": "https://checkout-demo.switchpayments.com/img/sample-desktop.jpg"
    },
    {
      "title": "Sample product",
      "reference": "5955200026",
      "quantity": 1,
      "price": 4.19,
      "thumbnail": "https://checkout-demo.switchpayments.com/img/sample-camera.jpg"
    }
  ]
}

```

```

    },
    {
      "title": "Sample product 3",
      "reference": "595326124",
      "quantity": 1,
      "price": 2.5
    },
    {
      "title": "Sample product 4",
      "reference": "595326456",
      "quantity": 1,
      "price": 1.5
    },
    {
      "title": "Sample product 5",
      "reference": "595365930",
      "quantity": 1,
      "price": 10,
      "thumbnail": "https://checkout-demo.switchpayments.com/img/sample-headphones.jpeg"
    }
  ],
  "totals": {
    "vat": 0,
    "shippingCosts": 5,
    "subtotal": 33.18
  },
  "merchant": {
    "name": "Sample Merchant",
    "logo": "https://switchpayments.com/homepage/imgs/base/switch-logo_normal.png",
    "publicKey": "publicKey",
    "environment": "SANDBOX"
  },
  "amount": 38.18,
  "merchantTransactionId": 25
}
}

```

Enabling Hosted Checkout in your payment operations is a four-step process.

1. Redirect Customer

You must redirect the Customer from your website to Hosted Checkout, you can do this immediately from the pay option in your cart or generate a URL which can be accessed by the Customer at a later time.

Your Hosted Checkout Transaction should include the same data as it is required by Dynamic Forms, this includes parameters such as amount, currency and chargesURL. Additionally it can also figure customization parameters exclusive to the Hosted Checkout feature, namely merchant_name and logo. Last but not least, it is relevant to input the order information. This refers to the selected products, shipping costs, and VAT, for example.

2. Create Charge

When creating a Charge, Merchants are expected to supply their own URL to the Hosted Checkout. This URL is called when the Customer submits the Payment details and should return the respective charge_id.

3. Implement Webhook Endpoint

It is essential to implement an endpoint which will receive the Webhooks generated from the Hosted Checkout procedures. This process is similar to what we implement for Dynamic Forms or REST Integration.

4. Redirect Customer Back

You should include a redirecting URL that guides the users back to your website once the Payment procedures have been completed.

04.3_REST Integration (not allowed for cards payment method)

Integrating with the Payshop Online Payments Platform

In case it is not possible to implement a specific flow using Dynamic Forms, the data collection can be performed by a merchant customized form and sent to the Payshop Online Payments platform using the REST Processing API.

This integration should be used for scenarios where Dynamic Forms are not practical. This can include custom transaction flows, for instance, authentication parameter collection split into multiple steps and webpages, or host-to-host communication that demands different compliance dynamics.

For Dynamic Forms you should consider the following order of instructions:

- 1. Displaying a form**
- 2. Generating a Charge**
- 3. Creating an Instrument**
- 4. Alternative flows: reference and redirection**

04.3.1_Display Form

Although this integration pattern does not provide a client-side library to render the UI, it still allows you to dynamically support new Payment Methods.

In order to do so, our API provides an endpoint that will describe which Payment Methods are currently active for your account and the corresponding fields that should be captured from the user (form schema). This information allows you to construct the UI in a dynamic manner, ensuring it can accommodate future changes and allowing you to truly leverage our Dashboard management capabilities to activate Payment Channels at will.

Path	Method	Description
https://switch-processing.teya.com/v2/charges/types	GET	Lists the Charge types currently active for this account.

REQUEST EXAMPLE

```
$ curl https://switch-processing.teya.com/v2/charges/types
-u publicKey
```

RESPONSE EXAMPLE

```
[
  "...",
  {
    "id": "card",
    "title": "Credit/Debit Card",
    "schema": {
      "required": ["name", "number", "expiration_month", "expiration_year", "cvv"],
      "properties": {
        "name": {
          "title": "Cardholder Name",
          "type": "string",
          "minLength": "3",
          "maxLength": "255"
        },
        "expiration_month": {
          "title": "Expiration Month",
          "type": "integer",
          "minimum": "1",
          "maximum": "12"
        }
      }
    },
    "ui_schema": {
      "name": {"ui:placeholder": "Full Name"},
      "number": {"ui:placeholder": "1111 2222 3333 4444"},
      "cvc": {
        "ui:placeholder": "123",
        "ui:help": "Last 3 digits on the back of the card"
      },
      "..."
    }
  },
  "..."
]
```

The payload returned by this endpoint can be leveraged by your client-side library to construct the UI with the active Payment Methods, gather the required data from the user, and validate it accordingly. This will grant a higher deal of flexibility to your front-end implementation, allowing you to seamlessly support future changes.

04.3.2_Generating a Charge

As stated in the Core Concepts section, our platform divides a Transaction into three executable Elements: Charge, Instrument, and Payment. The API allows you to coordinate the creation of these Elements, where required, through multiple available endpoints.

In order to generate a Charge, the following endpoint should be called by your backend platform using your Private API Key.

Path	Method	Description
https://[redacted]switch-processing.teya.com/v2/charges	POST	Creates a new Charge for a given transaction.

REQUEST EXAMPLE

```
-u accountId:privateKey
-d '{
  "charge_type": "card_onetime",
  "amount": 42,
  "currency": "EUR",
  "events_url": "https://your.url/v1/notificationHandler"
}'
```

Request Body Parameter

charge_type String **Required**

Indicates the Payment Method that has been selected by the Customer, should contain the value passed by our API call.

currency String **Required**

ISO 4217 code that indicates the currency that will be used by the Transaction.

amount Number **Required**

The amount that should be charged to the Customer.

events_url String

Allows you to configure the URL that will be called by the Payshop Online Payments platform to notify about Lifecycle Events related to this Transaction. Please note that the URL must use HTTPS.

redirect_url String

Specifies the HTTPS URL to where the users should be sent after they authenticated the Transaction on the Payment Method page. This parameter is only required for Payment Methods that require user redirection for authentication (e.g. PayPal).

instrument_params JSON Object

Specifies configuration parameters that are passed to the Provider when creating the Instrument element and which allow you to configure how the fund transfer is executed.

metadata JSON Object

This object allows you to pass any Transaction related data points that may be useful to be displayed in the Dashboard for analysis purposes.

channels Array

Gives the ability to specify which Channel should be used to process the transaction. Using this parameter you can leverage your business logic to select the Channel rather than relying on Dynamic Routing. If multiple values are present, a fallback mechanism will be used, giving precedence to the first entries

RESPONSE EXAMPLE

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "charge_type": "card_onetime",
  "amount": 10,
  "currency": "EUR",
  "confirmed": false,
  "external": false,
  "instrument_params": {
    "enable3ds": true
  },
  "events_url": "https://merchant.com/events",
  "channels": [{
    "processor": "checkout",
    "id": "85a557e4fdb6c8806f413bc75fabab162828e4f95b8e6390",
    "label": "card_onetime_checkout"
  },
  {
    "processor": "acapture",
    "id": "9fb7b1e253f1c592210b7c37b40b18e576ff30995b8e40de",
    "label": "card_onetime_acapture"
  }
],
  "charge_type_label": "Card One-Time",
  "redirect_url": "https://merchant.com/redirect",
  "metadata": {
    "orderId": "1337"
  },
  "external_ids": null,
  "request_log": {
    "country": "PT",
    "ip_address": "100.10.10.10",
    "user_agent": "curl/7.54.0",
    "library_version": null
  },
  "created_at": "2018-06-18T10:28:36.358233+00:00",
  "updated_at": "2018-06-18T10:28:36.358258+00:00"
}
```

Response Body Parameter

id String

The id that uniquely identifies the created Charge element for this Transaction.

charge_type String

Indicates the Payment Method that will be used to process the Transaction.

charge_type_level String

User-friendly description of the Payment Method that will be used to process the Transaction.

currency String

ISO 4217 code that indicates the currency that will be used by the Transaction.

confirmed Boolean

Indicates whether the current Charge element has been confirmed by the Merchant using their private key.

external Boolean

Indicates whether the current Charge element was not created by the Payshop Online Payments Platform.

instrument_params JSON Object

The configuration parameters that were specified when creating the Charge element, which allow you to configure how the funds transfer is executed by the Provider.

events_url String

Contains the URL that will be called by the Payshop Online Payments Platform to notify about Lifecycle Events related to this transaction.

redirect_url JSON Object

The URL to where the users should be sent after they authenticated the transaction on the Payment Method page.

channels Array

Documents the Channels that will be used to process the Transaction.

metadata JSON Object

The Transaction-related data points that were passed when creating the Charge, which may be useful to be displayed in the Dashboard for analysis purposes.

external_ids JSON Object

Documents the Provider id for the current Charge, if available.

request_log JSON Object

Stores information about the device that was used to create the current Charge element, such as `country`, `ip_address`, `user_agent`, and `library_version`.

created_at String

Indicates the date and time when the current Charge element was created.

Updated_at JSON Object

Documents the date and time when the last update was performed to the current Charge.

04.3.3_Create Instrument

At this stage in the payment processing process, we need to collect the required information from the user to execute the payment. This will be executed by your checkout page, leveraging your frontend framework, which should collect the parameters required depending on the Payment Method selected by the customer.

Once this is completed, your UI component should create the Instrument element using the following endpoint.

Path	Method	Description
https://switch-processing.teya.com/v2/instruments	POST	Creates a new Instrument for a given transaction.

REQUEST EXAMPLE

```
$ curl -vX POST https://switch-processing.teya.com/v2/instruments
-u publickey:
-d '{
  "charge": "35ed95bfb772b94c4e59f91fcbef0f5618d46e3d5b2b7da5",
  "name": "John doe",
  "number": "4111111111111111",
  "expiration_month": 12,
  "expiration_year": 2018,
  "cvc": "007"
}'
```

Request Body Parameter

charge String **Required**

The unique identifier for the Charge element that was previously created for this Transaction.

currency String **Required**

The remaining fields are dependent on the Payment Method that was previously selected. Below you can find an example for card_onetime.

name String

The name of the Cardholder, as printed on the card.

number Integer

The number of the card that will be used to process the Payment. It should contain 14 to 19 digits, without any separators.

expiration_month Integer

Zero padded, two-digit representation of the expiration month for the card.

expiration_year Integer

Zero padded, two-digit representation of the expiration year for the card.

cvc String

The card verification code should contain 3 to 4 digits according to the card brand.

RESPONSE EXAMPLE

```
{
  "id": "0d0e51462ef62787dcb711f3c7ec42d086a172f85b2b8ddc",
  "status": "pending",
  "customer": null,
  "used": false,
  "last_payment": null,
  "redirect": {
    "url": "https://switch-
processing.teya.com/v2/instruments/0d0e51462ef62787dcb711f3c7ec42d086a172f85b2b8ddc/redirect",
    "method": "GET",
    "parameters": null
  },
  "reference": null,
  "response": {
    "eci_code": "05"
  },
  "params": {
    "bin": "411111",
    "name": "John doe",
    "expiration_year": 2018,
    "brand": "VISA",
    "expiration_month": 12,
    "fingerprint":
"a1ccb846ceda84e80b91bee9025437b73784de262233a1bde39490bcb597fb69ef51d9dd3f67a2cb4ee85fb674104
60f138f9bb1ee5d7de2f27291e7e895f7a8",
    "enable3ds": true,
    "last_4_digits": "1111"
  },
  "external": false,
  "success": true,
  "failure_description": null,
  "channel": {
    "processor": "checkout",
    "id": "85a557e4fdb6c8806f413bc75fabab162828e4f95b8e6390",
    "label": "card_onetime_checkout"
  },
}
```



```
"charge": {
  "charge_type": "card_onetime",
  "currency": "EUR",
  "created_at": "2018-06-21T10:27:49.241769+00:00",
  "charge_type_label": "Card One-Time",
  "amount": 10,
  "id": "35ed95bfb772b94c4e59f91fcbef0f5618d46e3d5b2b7da5"
},
"external_ids": {
  "processor": "8a8294496421b8d3016422227f17687"
},
"request_log": {
  "country": null,
  "ip_address": "100.10.10.10",
  "user_agent": "curl/7.54.0",
  "library_version": null
},
"created_at": "2018-06-21T11:37:01.694149+00:00",
"updated_at": "2018-06-21T11:37:01.694168+00:00"
}
```

Response Body Parameter

id String

The **id** that uniquely identifies the created Instrument element for this transaction.

fingerprint String

A unique, one-way hash fingerprint of the current this Instrument, which can be leveraged for Risk prevention purposes.

success Boolean

This boolean flag indicates if the current Instrument has been successfully executed.
true: the Instrument has been successfully executed. **false**: the Instrument has failed to be executed.

status String

Documents the execution status for the Instrument. **authorized**: the Instrument was successfully authorized by the Provider. **invalid**: the Instrument was considered invalid by the Payshop Online Payments Platform or the Provider. **pending**: used for asynchronous Payment Methods, indicates that the Customer is yet to provide additional information to proceed forward.

response JSON Object

Contains the technical information returned by the Provider when processing the current Instrument, such as the card ECI code.

failure_description String

Used for scenarios where the Instrument fails to be created, this parameter will contain a textual description of the error.

used Boolean

Indicates if the current Instrument has been previously used. **true**: the Instrument has been used to perform a previous transaction. **false**: the Instrument is yet to be used to perform a transaction.

redirect String

Used for Payment Methods that require redirection, documents the information required to direct the user to the page where he will complete the Transaction: **url**, **method**, **params**.

reference JSON Object

Only applicable to Payment Methods that require the Customer to complete the Transaction asynchronously outside of the Payshop Online Payments platform. Documents the information required by the Customer to be able to do so, the specific schema of this object will be dependent on the Payment Method.

external Boolean

Indicates whether the current Instrument element was not created by the Payshop Online Payments platform. **true**: the Instrument has not been processed by the platform. **false**: the Instrument has been created using the platform.

channels JSON Object

Documents the Channel that was used to process the Instrument.

charge JSON Object

Basic Charge information,

external_ids JSON Object

Documents the Provider id for the current Instrument, if available.

request_log String

Stores information about the device that was used to create the current Instrument element, such as: country, ip_address, user_agent, and library_version.

created_at_at String

Indicates the date and time when the current Instrument element was created.

created_at_at String

updated_at String

Documents the date and time when the last update was performed to the current Charge.

04.3.4_Alternative Flows

Reference

For payment methods that require the customer to push funds using an account reference (e.g. Multibanco, by going to an ATM), the reference fields should be displayed to the customer.

Redirection

For payment methods that require redirecting the user to a provider page (e.g. Paypal), after creating the instrument, the app should redirect the user to

'<https://switch-processing.teya.com/v2/instruments/>' + `instrument.id` + `'/redirect'`. After that, the user will be redirected to the `redirectUrl` defined in the charge.

That's it!

By following these steps, your product is already enabled for many payment methods and providers, and your integration is ready to be tested.

Next Steps

You should create your `events_url` and handle Transaction Events, to mark transactions as paid in your database.

There are also a few more Transaction Flows you should handle, to truly support every payment method, provider, and flow in the world.

04.4_Transaction Flows

Integrating with the Payshop Online Payments Platform

Supporting different Payment Methods under the same platform in a standardized manner brings additional challenges when it comes to transaction execution. Each Payment Method is designed with its own philosophy when it comes to processing transactions, and therefore requires different steps, and different execution flows.

Accommodating this diversity under the same platform required us to support four transaction flows: Capture on Creation, Auth-Capture, Recurring on Auth, and Recurring on Capture.

Transaction Flow	Payment executed automatically?	Recurring Payments
1. Capture on Creation	Yes	No

2. Auth-Capture	No	No
3. Recurring on Auth	No	No
4. Recurring on Capture	No	Yes

Capture on Creation

This flow targets transactions that only need to take place once, and where the funds do not need to be captured at a later time, therefore not requiring an authorization to take place.

From a platform perspective, this flow does not require you to create the Payment element to execute the funds' transfer. The Processing Application will automatically create the Payment element once you create the Instrument, given that the funds will be captured right away.

Auth-Capture

An Auth-Capture flow is intended for transactions that take place once, and have the funds captured at a later time, thus requiring two steps: authorization and funds capture.

The authorization is executed when the Instrument element is created in our platform, either automatically using our Dynamic Forms or manually leveraging our REST API. At this stage, the money has been reserved from the customer accounting platform but has not yet been transferred. The actual funds' transfer only happens when the Payment element is created in the platform.

Recurring on Auth

Destined for payments that need to happen on a frequent basis, this flow allows you to create a reusable Instrument element that can be leveraged to make multiple Payments.

This particular recurring flow automatically executes the Payment element when the Instrument is created. It means that for the

Recurring on Capture

Designed for transactions that need to take place regularly, enables you to create a reusable Instrument element that can be used to perform multiple Payments.

This particular recurring flow does not automatically execute the Payment element when the Instrument is created. In short, it requires you to create a Payment element every time you want to execute a transaction.

first payment, you only need to create the Instrument, as the platform will automatically create the first Payment. For subsequent transactions you only need to create the Payment element, referencing the same specific Instrument.

Instrument flows

Depending on the payment method, after creating an Instrument on the client-side, one of three things might happen:

1. If the payment is synchronous, the flow will end on success, and the customer can be shown the success screen (e.g. Credit Card).
2. If the payment requires the customer to push funds using an account reference this reference should be displayed to the customer or the dynamic forms can show it for you (e.g. Multibanco, by going to an ATM).
3. If the payment requires redirection, the Dynamic Forms will redirect the customer to the payment provider page for authentication. Afterward, it will redirect back to your server's redirectUrl with an Instrument id parameter with which you can confirm the Instrument status (e.g. Paypal),.

To support redirection, you should add a redirectUrl on the charge creation. For convenience, you simply can add it for all transactions, even if it is not necessary.

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/charges
-u accountId:privateKey
-d '{
  "charge_type": "card_onetime",
  "amount":42,
  "currency":"EUR",
  "redirectUrl":"https://www.merchant.com/redirectUrl"
}'
```

04.4.1_Capturing funds at a later time

Depending on the Payment Method, there are specific server-side operations that can be enabled after redirecting.

1. If the Instrument is not captured on creation, it means that the funds have been authorized/reserved and can be captured/transferred in a future time.
2. If the Instrument is recurring, it means that the Customer has authorized multiple Payments to be created.

In both these cases, the instrument id should be saved on your database during the instrument.authorized Event and used to create a Payment at any time, by calling the Payments endpoint using the Instrument.

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/payments
-u accountId:privateKey
-d '{
  "instrument": "4145f3be2094da8b786fe9ab5c6ed206d7c2e",
  "amount":42,
  "currency":"EUR"
}
```

Result Parameters

id String

Payment id, to be used for refunds.

success String

Payment success status.

refundable Boolean

Whether refunds can be created for this payment

RESULT HTTP 201

```
{
  "id": "206d7c2e4145f3be2b786fe9ab5c6ed094da8",
  "success": true,
  "refundable": true
}
```

04.4.2_Refunding Transactions

If a payment is refundable, you can call its refund method with the private keys.

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/payments/{id}/refund
-u accountId:privateKey
-d '{
  "amount":42,
  "currency":"EUR",
}'
```

RESULT HTTP 201

```
{
  "success": true
}
```

Result Parameter

success Boolean

Boolean that marks whether transactions are successful or unsuccessful.

04.5_Event Handling

Integrating with the Payshop Online Payments Platform

Search: Trouble finding a payment? Need to dive deeper into a specific transaction? Customized searches for one or multiple Elements take into account specified criteria, allowing you to consult technical details;

Events: When analyzing your payment operation, every moment counts. Each of the previously mentioned Elements come with their event sets. This means, for instance, you can decompose the various events related to a transaction. Consulting the moment when it became successful, it failed or it was settled.

04.5.1_Receive Events

To start receiving events on your server, add an HTTPS events_url during the Charge creation.

Path	Method	Description
https://switch-processing.teya.com/v2/charges	POST	Add events_url to Charge creation.

ADD EVENTS URL

```
$ curl -vX POST https://switch-processing.teya.com/v2/charges
-u accountId:privateKey
-d '{
  "charge_type": "card_onetime",
  "amount": 42,
  "currency": "EUR",
  "events_url": "https://merchant.com/events"
}
```

Parameter Description

event String

Lifecycle Event id.

event_type String

Lifecycle Event type.

As a Transaction progresses its lifecycle, the Payshop Online Payments platform will send an HTTPS POST request to your `events_url` with two parameters. The most important Event Type to listen to is the `instrument.authorized` Event, which happens when the Customer Payment data has been confirmed.

04.5.2_Event Details

GET /v2/events/{id}

To get the event details, use the event parameter as the `{id}` in the following request

REQUEST

```
$ curl https://switch-processing.teya.com/v2/events/{id}
-u accountId:privateKey
```

Result Parameter

id String

Unique event identifier

created_at String

The date when the event was created

object_id String

The unique identifier of the event related object (e.g. refund_id)

result_status Integer

HTTP status of the event

retries_log Array

A list with all the retries

type String

The event type (e.g. charge.created, instrument.created)

type_label String

Pretty name for the event type

charge JSON Object

The charge object (see Charges)

instrument JSON Object

The instrument object (see Instruments)

payment JSON Object

The payment object (see Payments)

refund JSON Object

The refund object (see Refunds)

dismissed_at String

The date when the event was marked as dismissed

settlement JSON Object

The settlement object

RESPONSE HTTP 200

```
HTTP 200
{
  "id": "f3be209e9ab06d7c2e4145f3be209e9fggab5c",
  "type": "instrument.authorized",
  "charge": {
    "id": "06d7c2e4145f3be209e9ab5c6ed24da8b786f",
    "type": "card",
    "amount": 42,
    "currency": "EUR",
    "metadata": {"orderId": "1337"}
  },
  "instrument": {
    "id": "4145f3be2094da8b786fe9ab5c6ed206d7c2e",
    "failure_description": null,
    "recurring": false,
    "captured_on_creation": true
  }
}
```

Events have many parameters, corresponding to every detail available on the Transaction at that moment. In the previous example you can find some of the most important ones. On an `instrument.authorized` Event, the transaction corresponding to the `charge.metadata` can be marked as successfully completed in the database.

04.5.3_Event Types and Parameters

It's possible to configure independent webhooks to handle each event type or to handle all of them in the same endpoint. The different lifecycle events that are communicated to the merchant server and can be browsed on the dashboard or through the Lifecycle API.

Event	Description
Charge created	Charge was initiated by the merchant or customer
Charge confirmed	Charge was confirmed by the merchant
Instrument authorized	An Instrument is ready to be used to create a Payment
Instrument pending	Instrument is awaiting customer input
Instrument invalid	Instrument was not authorized or was invalidated by the merchant

Instrument risk

Instrument was tagged by the Risk Management

Instrument settled

Instrument has been settled by the payment provider

Payment error

Payment attempt was unsuccessful

Payment pending

Payment is pending approval by the payment provider

Payment success

Payment was successfully created

Payment risk

Payment was tagged by the Risk Management

Payment settled

Payment has been settled by the payment provider

Refund error

Refund attempt was unsuccessful

Refund pending

Refund is pending approval by the payment provider

Refund success

Refund was successfully created

Refund settled

Refund has been settled by the payment provider

Reversal success

Authorization was successfully voided

Reversal pending

Authorization reversal is pending approval by the payment provider

Reversal error

Authorization was not possible to reverse

Dispute created

Payment was disputed by the Customer

Dispute settled

Dispute has been settled by the payment provider

EVENT PARAMETER EXAMPLE

```
{
  "id": "f3be209e9ab06d7c2e4145f3be209e9fggab5c",
  "type": "instrument.authorized",
  "charge": {
    "id": "06d7c2e4145f3be209e9ab5c6ed24da8b786f",
    "type": "card",
    "amount": 42,
    "currency": "EUR",
    "metadata": {"orderId": "1337"}
  },
  "instrument": {
```



```

    "id": "4145f3be2094da8b786fe9ab5c6ed206d7c2e",
    "failure_description": null,
    "recurring": false,
    "captured_on_creation": true
  }
}

```

Availability	Result Parameter	Description
	event.id	Lifecycle Event id.
	event.type	Lifecycle Event type.
	charge.id	Charge id.
Result Available on All Events	Parameters charge_type	Payment method chosen by the Customer.
	charge.amount	Charged amount.
	charge.currency	Processing currency.
	charge.metadata	Parameter sent during the creation of the charge, that identifies the Transaction in your system, allowing you to mark it as successfully completed in your database.
	instrument.id	This is the id of the transaction authorization provided by your customer. It acts as a token in recurring Payments.
	instrument.failure_description	If the instrument failed, this field contains the description of the error.
Result Parameters available on Instrument and Payment Events	instrument.request_log	Payment method chosen by the Customer.
	Instrument.recurring	Whether the Instrument allows multiple Payments to be created from this Customer authorization.
	Instrument.captured_on_creation	Whether a Payment has been created automatically or if it needs to be captured in a separate operation..

Result Parameters available on Payment Events payment.id

Payment id used to create Refunds.

Charges

POST /v1/sources/charges

To get the event details, use the event parameter as the {id} in the following request

Path	Method	Description
Sandbox https://switch-gateway.teya.com/v1/sources/charges	POST	Add charge elements using External Sources.
Production https://switch-gateway.teya.com/v1/sources/charges		

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/sources/charges?merchant_id=merchantID -u  
accountId:APIKey -d '{  
  "charge_type": "card_onetime",  
  "amount": 42,  
  "currency": "EUR",  
  "created_at": "2018-10-16T15:15:34.694149+00:00",  
  "operation": {"metadata": {"transaction_id": "123123123"}},  
  "channels": ["card_onetime_checkout"],  
  "metadata": {"orderId": "1337"},  
  "events_url": "https://merchant.com/events",  
  "redirect_url": "https://merchant.com/redirect",  
  "instrument_params": {"descriptor": "D891220"},  
  "request_log": {"country": null, "ip_address": "100.10.10.10", "user_agent": "curl/7.54.0",  
  "library_version": null}  
}'
```

Request Parameter

charge_type String **Required**

Payment method selected by the customer.

amount Float **Required**

Transaction amount.

currency String **Required**

Transaction currency.

created_at String **Required**

The date when the charge was created (ISO 8601).

operation JSON Object **Required**

Include `metadata` keyword (it must be a JSON object) with some external operation metadata from the provider.

channels Array

Indicates which channels you want to use for the transaction.

metadata JSON Object

Any metadata that uniquely identifies this transaction in your system. This field will be used to identify this transaction when handling events, searching for transactions on the dashboard, dynamically routing transactions or managing risk.

events_url String

HTTPS merchant server-side webhook where events will be handled.

redirect_url String

HTTPS URL (or URL schemas for mobile apps) to send users back to, for payment methods that require customer redirection (e.g. Paypal, credit cards with 3D-secure).

instrument_params JSON Object

The merchant can pass parameters which will be used when creating the instrument.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version. Defaults to the one who made the request.

RESULT: HTTP 201

```
{
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "amount": 42,
  "currency": "EUR",
  "charge_type": "card_onetime",
  "charge_type_label": "Card One-Time",
  "confirmed": true,
  "created_at": "2018-10-16T15:15:34.694149+00:00",
  "updated_at": "2018-10-16T15:15:34.694149+00:00",
  "external": true,
  "metadata": {
    "orderId": "1337"
  },
  "events_url": "https://merchant.com/events",
  "redirect_url": "https://merchant.com/redirect",
  "instrument_params": {
    "descriptor": "D891220"
  },
  "external_ids": null,
  "request_log": {
    "country": null,
    "ip_address": "100.10.10.10",
    "user_agent": "curl/7.54.0",
    "library_version": null
  },
  "channels": [{
    "processor": "checkout",
    "id": "85a557e4fdb6c8806f413bc75fabab162828e4f95b8e6390",
    "label": "card_onetime_checkout"
  }
]
```

```
  }  
}
```

Result Parameter

id String

The charge ID that can be used once to create an instrument.

amount Float

Maximum amount set for the charge.

currency String

The currency of the charge.

charge_type String

Identifier of the charge type.

charge_type_label String

The designation attributed to the charge.

confirmed Boolean

Indicates if the created charge was confirmed by the merchant using their private key.

created_at String

The date when the charge was created.

updated_at String

The date when the charge was last updated.

external Boolean

Indicates if the charge was created as an external source.

metadata JSON Object

Provided charge metadata.

external_ids JSON Object

Provider IDs of the charge.

redirect_url String

Redirect url defined.

instrument_params JSON Object

The instrument parameters used to create the charge.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version.

channels Array

Channels to be used in the transaction.

Check out the following use case

It is not possible to create a charge without registering all the required fields. Tending to this error case, a list with all the invalid fields and the respective errors is returned.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/sources/charges?merchant_id=merchantID -u  
accountId:APIKey -d '{
```

```
"charge_type": "card_onetime",
"amount": 42,
"created_at": "2018-10-16T15:15:34.694149+00:00"
}'
```

Result Parameter

message String

String with the error details, in this case: "Invalid parameters"

parameters JSON Object

A list with all the invalid fields and the respective errors

RESULT: HTTP 400

```
{
  "message": "Invalid parameters",
  "parameters": {
    "currency": [
      "This field is required."
    ]
  }
}
```

Instruments

POST /v1/sources/instrument

Path	Method	Description
------	--------	-------------

Sandbox

https://switch-gateway.teya.com/v1/sources/instruments

POST

Add instrument elements using External Sources.

Production

https://switch-gateway.teya.com/v1/sources/instruments

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/sources/instruments?merchant_id=merchantID -u  
accountId:APIKey -d '{  
  "charge": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",  
  "created_at": "2018-10-16T15:15:34.694149+00:00",  
  "operation": {"metadata": {"transaction_id":  
"918dd9382a17be5d060f9a8dd15674b047b7f5d15b3e2f27"}},  
  "request_log": {"country": null, "ip_address": "100.10.10.10", "user_agent": "curl/7.54.0",  
  "library_version": null}  
}'
```

Request Parameter

charge String **Required**

The identifier of the charge that was previously created.

created_at String **Required**

The date when the charge was created (ISO 8601).

operation JSON Object **Required**

Any operation metadata required to handle the transaction on the processor side.

params JSON Object

The sanitized instrument parameters.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version. Defaults to the one who made the request.

RESULT: HTTP 200

```
{
  "id": "0d0e51462ef62787dcb711f3c7ec42d086a172f85b2b8ddc",
  "status": "pending",
  "success": true,
  "used": false,
  "external": true,
  "charge": {
    "charge_type": "card_onetime",
    "currency": "EUR",
    "created_at": "2018-06-18T10:28:36.358233+00:00",
    "charge_type_label": "Card One-Time",
    "amount": 10,
    "id": "35ed95bfb772b94c4e59f91fcbef0f5618d46e3d5b2b7da5"
  },
  "last_payment": null,
  "created_at": "2018-10-16T15:15:34.694149+00:00",
  "updated_at": "2018-06-21T11:37:01.694168+00:00",
  "request_log": {
    "country": null,
    "ip_address": "100.10.10.10",
    "user_agent": "curl/7.54.0",
    "library_version": null
  },
  "external_ids": {
    "processor": "8a8294496421b8d3016422227f17687",
    "transaction_id": "918dd9382a17be5d060f9a8dd15674b047b7f5d15b3e2f27"
  },
  "redirect": {
    "url": "https://test.ppipe.net/connectors/demo/simulator.link",
    "method": null,
    "parameters": [
      {
        "name": "MD",
        "value": "8a8294496421b8d30164222287d768e"
      }
    ]
  }
}
```

```
    "name": "PaReq",
    "value":
      "IT8ubu+5z4YupUCOEHKsbiPep8UzIAcPKJEjpwGlzD8#KioqKioqKioqKioqMTExMSMxMC4wMCBFVVIj"
  },
  {
    "name": "TermUrl",
    "value":
      "https://test.ppipe.net/connectors/asyncresponse_simulator;jsessionid=9DBF6D37DA9B12E4B23A70F24
      52A8021.sbg-vm-
      con02?asyncsource=THREEDSECURE&ndcid=8a82941751c365120151c4a473fa04bf_830642b94a9040a3
      a08463b9414f0396"
  },
  {
    "name": "connector",
    "value": "THREEDSECURE"
  }
]
},
"params": {
  "descriptor": "D891220"
},
"channel": {
  "processor": "checkout",
  "id": "85a557e4fdb6c8806f413bc75fabab162828e4f95b8e6390",
  "label": "card_onetime_checkout"
},
"failure_description": null
}
```

Result Parameter

id String

The instrument ID that can be used to create a payment.

status String

Current status of the instrument. It could be pending, invalid or authorized.

success Boolean

Flag indicating whether the instrument was successfully created.

used Boolean

Flag indicating whether the instrument has already been used.

external Boolean

Indicates if the created charge was created as an external source.

charge JSON Object

Details about the charge.

last_payment JSON Object

A reference for the last payment of the instrument. If it exists, the object has the following keys: id (string) and success (boolean).

created_at String

The date when the instrument was created.

updated_at String

The date when the instrument was last updated.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version.

external_ids JSON Object

Useful external ids (e.g. IDs from the processing channel).

redirect JSON Object

Contains information from the provider on how to redirect the merchant.

params JSON Object

Includes the instrument parameters concatenated with the instrument_params from the corresponding charge.

channel JSON Object

Channel used to create the instrument.

failure_description String

If the instrument was not successful there may be details in this field about the reasons for the error (e.g. from the processor).

Payments

POST /v1/sources/payments

Path	Method	Description
Sandbox https://switch-gateway.teya.com/v1/sources/payments	POST	Add payment elements using External Sources.
Production https://switch-gateway.teya.com/v1/sources/payments		

Request Parameter

instrument String **Required**

The identifier of the instrument that was previously created.

amount Float **Required**

Transaction amount.

currency String **Required**

Transaction currency.

created_at String **Required**

The date when the charge was created (ISO 8601).

operation JSON Object **Required**

Any operation metadata required to handle the transaction on the processor side.

description String

Transaction description.

metadata JSON Object

Any metadata that uniquely identifies this transaction in your system. This field will be used to identify this transaction when handling events, searching for transactions on the dashboard, dynamically routing transactions or managing risk.

params JSON Object

The sanitized payment parameters.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version. Defaults to the one who made the request.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/sources/payments?merchant_id=merchantID -u
accountId:APIKey -d '{
  "instrument": "0d0e51462ef62787dcb711f3c7ec42d086a172f85b2b8ddc",
  "amount": 10,
  "currency": "EUR",
  "created_at": "2018-10-16T15:15:34.694149+00:00",
  "operation": {"metadata": {"transaction_id":
"918dd9382a17be5d060f9a8dd15674b047b7f5d15b3e2f27"}},
  "description": "Payment description",
  "metadata": {"orderId": "123"},
  "request_log": {"country": null, "ip_address": "127.0.0.1", "user_agent": "curl/7.54.0",
"library_version": null}
}'
```

Result Parameter

id String

Unique identifier for the payment.

amount Float

The payment amount.

currency String

The currency of the payment.

description String

Transaction description.

created_at String

The date when the payment was created.

updated_at String

Last date when the payment was last updated.

external_ids JSON Object

Useful external ids (e.g. IDs from the processing channel).

success Boolean

Flag indicating whether the payment was successfully created.

external Boolean

Indicates if the created charge was created as an external source.

charge JSON Object

Contains a trimmed object of the related charge: id, charge_type, charge_type_label, metadata.

instrument JSON Object

Contains a trimmed object of the related instrument: id, channel.

refunds Array

A list that contains all the refunds objects related to the payment.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version.

params JSON Object

The payment parameters.

refundable Boolean

Defines whether or not the payment allows for refunds.

metadata JSON Object

Any metadata that uniquely identifies this transaction in your system. This field will be used to identify this transaction when handling events, searching for transactions on the dashboard, dynamically routing transactions or managing risk.

failure_description String

If the payment is not successful, this field has details about the failure.

RESULT

```
{
  "id": "ed9fdf723c40fada03b6ce783beb182376a85c735b3e2f27",
  "amount": 42,
  "currency": "EUR",
  "description": "Payment description",
  "created_at": "2018-10-16T15:15:34.694149+00:00",
  "updated_at": "2018-10-16T15:15:34.694149+00:00",
  "external_ids": {
    "transaction_id_trunc": "918dd9382a17be5d060f9a8dd15674",
    "processor": "8a8294496421b8d3016422227f17687",
    "transaction_id": "918dd9382a17be5d060f9a8dd15674b047b7f5d15b3e2f27"
  }
}
```

```

},
"success": true,
"external": true,
"charge": {
  "charge_type": "card_onetime",
  "charge_type_label": "Card One-Time",
  "id": "756ae7bdc3390050cf6648fb819ac1c4de02f4d15b278954",
  "metadata": {
    "orderId": "837232"
  }
},
"instrument": {
  "id": "0d0e51462ef62787dcb711f3c7ec42d086a172f85b2b8ddc",
  "channel": {
    "processor": "checkout",
    "id": "85a557e4fdb6c8806f413bc75fabab162828e4f95b8e6390",
    "label": "card_onetime_checkout"
  }
},
"refunds": [],
"request_log": {
  "country": null,
  "ip_address": "127.0.0.1",
  "user_agent": "curl/7.54.0",
  "library_version": null
},
"params": null,
"refundable": false,
"metadata": {"orderId": "123"},
"failure_description": null
}

```

Refunds

POST /v1/sources/refunds

Path	Method	Description
Sandbox https://switch-gateway.teya.com/v1/sources/refunds	POST	Add refunds elements using External Sources.

Production

https://switch-gateway.teya.com/v1/sources/refunds

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/sources/refunds?merchant_id=merchantID -u  
accountId:APIKey -d '{  
  "payment": "ed9fdf723c40fada03b6ce783beb182376a85c735b3e2f27",  
  "amount": 10,  
  "created_at": "2018-10-16T15:15:34.694149+00:00",  
  "operation": {"metadata": {"transaction_id":  
"918dd9382a17be5d060f9a8dd15674b047b7f5d15b3e2f27"}},  
  "description": "Unapplied coupon",  
  "request_log": {"country": null, "ip_address": "127.0.0.1", "user_agent": "curl/7.54.0",  
"library_version": null}  
}'
```

Request Parameter

payment String **Required**

The identifier of the payment that was previously created.

amount Float **Required**

The amount to be refunded. It can be partial and it also cannot exceed the total amount of the payment.

created_at String **Required**

The date when the charge was created (ISO 8601).

operation JSON Object **Required**

Any operation metadata required to handle the transaction on the processor side.

description String

Transaction description.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version. Defaults to the one who made the request.

RESULT

```
{
  "id": "2d017c99745a15ddeda9ba0e35d8e1e26d3b512a5bc5d259",
  "amount": 1.0,
  "success": true,
  "description": "Unapplied coupon",
  "created_at": "2018-10-16T15:15:34.694149+00:00",
  "payment": {
    "id": "ed9fdf723c40fada03b6ce783beb182376a85c735b3e2f27"
  },
  "request_log": {
    "country": null,
    "ip_address": "127.0.0.1",
    "user_agent": "curl/7.54.0",
    "library_version": null
  },
  "external_ids": {
    "transaction_id_trunc": "918dd9382a17be5d060f9a8dd15674",
    "processor": "8a8294496421b8d3016422227f17687",
    "transaction_id": "918dd9382a17be5d060f9a8dd15674b047b7f5d15b3e2f27"
  },
  "failure_description": null
}
```

Result Parameter

id String

Unique identifier of the payment.

amount Float

The refund amount. It must not exceed the payment amount.

success Boolean

Flag indicating whether the refund was successfully created.

external Boolean

Indicates if the created charge was created as an external source.

description String

Transaction description.

created_at String

The date when the payment was created.

payment JSON Object

Contains a trimmed object of the related payment: id.

request_log JSON Object

Contains information on the origin of the request: country, ip_address, user_agent and the library_version.

external_ids JSON Object

Useful external ids (e.g. IDs from the processing channel).

failure_description String

If the refund is not successful, this field has details about the failure.

04.6_Error Codes

Here you can find codes and descriptions of all errors that may occur when you communicate with the Payshop Online Payments Platform APIs.

Error Code Format: XX.YY.ZZZZ (numeric)

- 01.YY.ZZZZ: Validation Errors
The request failed because there are issues with the data being sent.
- 02.YY.ZZZZ: Soft Declines
The request was declined. Yet, subsequent attempts may be successful.

- 03.YY.ZZZZ: Hard Declines
The request was declined. Most hard declines require the Issuer or Customer to rectify issue(s) before a subsequent attempt can be made.
- 04.YY.ZZZZ: Communication
Problems in the communication between the Payshop Online Payments platform and the Processor. If these issues are recurring, it is important to check for the root cause.
- 05.YY.ZZZZ: Implementation
Errors raised when the integration with the Payshop Online Payments Platform was performed incorrectly.
- 06.YY.ZZZZ: Risk
Possible fraudulent activities.
- 99.YY.ZZZZ: Unhandled Errors
The request failed due to unspecified reasons. Further analysis might be needed.

Validation

Errors

Error Code	Designation	Description
01.01.0000	Invalid Parameters	Request containing invalid parameters (e.g. email address).
01.02.0000	Amount Mismatch	Invalid amount. The amount inputted does not match the request.
01.03.0000	Instrument Mismatch	Request containing invalid instrument details. The payment instrument used does not match the criteria set by the processor.

Soft

Declines

Error Code	Designation	Description
02.00.0001	Failure	Failure originated in the processor.
02.01.0000	Too Many Tries	You have exceeded the maximum number of tries.
02.02.0000	Limit Exceeded	The customer exceeds the maximum allowed limit.
02.03.0000	Insufficient Funds	There are not enough funds to complete this payment.
02.04.0000	Temporary Issue	There is a temporary problem with your submission. Retry.

Hard Declines

Error Code	Designation	Description
03.00.0001	Processor Declined	The processor refused the transaction request.

03.00.0002	Instrument Expired	The payment Instrument being used has expired.
03.01.0000	Disabled Account	The customer's account has been disabled.
03.02.0000	Unsupported Value	Unsupported value for this request. Review the content and retry.
03.02.0001	Unsupported Scheme	Your payment instrument does not support this type of purchase.
03.02.0002	Unsupported BIN	Unsupported bank identification number.
03.03.0000	Card is Declined	The transaction is refused by the issuer. Use a different card or contact your bank for further details.
03.03.0001	Card is Flagged as Lost/Stolen/Expired / Restricted	Card flagged as lost, stolen, expired, or restricted.
03.04.0000	Duplicate Operation	A transaction request with the same information has already been submitted.

Communication

Error Code	Designation	Description
04.01.0000	Processor Unavailable	Processor not available to confirm the transaction.

04.02.0000	Timeout Communicating With Processor	The server took too long to respond. Check the status of the transaction. Retry if needed.
------------	--	--

Implementation

Error Code	Designation	Description
05.01.0000	Configuration Error	Channel is not correctly configured.
05.02.0000	Processor Configuration Error	Processor is not correctly configured.
05.01.0002	Merchant Configuration Error	Merchant account is not correctly configured in the processor.
05.02.0000	Invalid Processor Credentials	The credentials supplied do not grant access to the requested resource.
05.03.0000	Authorization Error	Permissions error.
05.03.0001	Processor Authorization Error	Permissions error coming from the processor.
05.03.0002	Merchant Authorization Error	Access to resource denied. Verify your account permissions.

05.04.0000	Feature Available	Not	Feature does not exist.
05.04.0001	Instrument Already Captured		The status of the instrument does not allow for your request (eg. reversing a card authorization that is already captured).
05.04.0002	Instrument Authorized	Not	The instrument verification does not allow for your request (eg. reversing a card authorization with 3DS that is still not authorized).

Unhandled

Errors

Error Code	Designation	Description
99.00.0000	Unspecified Error	Unknown/unspecified errors that do not fit any other category.
99.01.0000	Internal Server Error	The server encountered an internal error or misconfiguration and was unable to complete your request.
99.02.0000	Processor Integration Error	Issue(s) with the processor integration.
99.02.0001	Unexpected Processor Response	Processor response does not match the expected parameters.
99.02.0002	Unable To Read Processor Response	Processor response cannot be interpreted.

99.02.0003	Processor Resource Found	Not	Processor issued resources do not match the needs of your request.
99.02.0004	Processor Unhandled Error		The request failed due to unspecified reasons. Issue not addressed by the processor.

04.6.1_Testing

Error handling is an important step in integration. To aid with error testing we have made triggering these errors a straightforward process. You only need to mind the amount set when you create a transaction.

In order to come up with the right amount to trigger the error you are looking for, mind the following instructions:

- Amount: XZZZZ.YY (numeric);
- The integer part of the amount represents the error code that will be returned. The decimal part of the amount represents the element from which the error originates (e.g. 0.10 for Charge, 0.20 for Instrument, 0.30 for Payment, 0.40 for Refund, 0.50 for Reversal);
- Do not include the "." separators and the leading "0"s in this number.

Examples:

- "amount": `6020000.20`
Results in an instrument error (0.20), error code 06.02.0000.
- "amount": `3030001.30`
Results in a payment error (0.30), error code 03.03.0001.

Let's use the first amount to exemplify the procedures involved in triggering an error.

```
"amount": 6010000.20
```

Results in an instrument error (0.20), error code 06.01.0000.

CREATE CHARGE: REQUEST PARAMETERS

```
$ curl -vX POST https://switch-processing.teya.com/v2/charges -u accountId:privateKey -d '{
  "charge_type": "card_onetime",
  "amount": 6020000.20,
  "currency": "EUR",
  "events_url": "https://merchant.com/events",
  "channels": ["card_onetime_no_processor"]
}
```

```
"amount": 6020000.20
```

All transactions begin with a charge. When creating this charge, you should input the amount that corresponds to the error you intend to trigger.

Additionally, you should note that the amount that defines the error is always the charge amount, even if the current resource allows for a different amount to be set (e.g. payments, refunds).

```
"channels": ["card_onetime_no_processor"]
```

You should use a `no_processor` channel for error testing. Channels result from a concatenation of the `charge_type` applied and the actual channel used for the transaction, hence the `["card_onetime_no_processor"]` in this case.

To gain access to this test channel and proceed with error testing you should contact our Support Department.

CREATE CHARGE: RESULT PARAMETERS

```
{
  "id": "675d32208a91bd3ad70641ba7f810036c65f5bb75ea9897f",
  "external_ids": null,
  "charge_type": "card_onetime",
  "charge_type_label": "Card One-Time",
  "amount": 6020000.2,
  "currency": "EUR",
  "events_url": "https://merchant.com/events",
  "redirect_url": "",
  "metadata": {},
  "instrument_params": null,
  "failure_code": null,
  "failure_description": null,
  "channels": [
    {
      "id": "d33577bc1ed9e106c050ff02bcccc285d57c5c5ea98619",
      "label": "card_onetime_no_processor",
      "processor": "no_processor"
    }
  ],
  "confirmed": true,
  "created_at": "2020-04-29T14:04:47.145730+00:00",
  "expires_at": null,
  "updated_at": "2020-04-29T14:04:47.145770+00:00",
  "request_log": {
    "ip_address": "89.155.14.252",
    "country": "PT",
    "user_agent": "PostmanRuntime/7.24.1",
    "library_version": null
  },
  "external": false
}
```

The error we are triggering is generated in an Instrument. Next, create an instrument deriving from the previous Charge, using the correspondent `charge_id`.

CREATE CHARGE: REQUEST PARAMETERS

```
{
  "charge": "675d32208a91bd3ad70641ba7f810036c65f5bb75ea9897f",
  "name": "John doe",
  "number": "4111111111111111",
  "expiration_month": 12,
  "expiration_year": 2020,
  "cvc": "007"
}
```

You should find the error in the response. In this case, we triggered a 06.02.0000: Unsafe Transaction. The respective Instrument generated is Invalid, as observable in the Switch Dashboard.

CREATE INSTRUMENT: REQUEST PARAMETERS

```
{
  "message": "Transaction Error",
  "metadata": {
    "failure_description": null
  },
  "failure_description": "Unsafe Transaction",
  "failure_code": "06.02.0000"
}
```

05_Processing

Every payment flow, one integration.

A key element of the Payshop Online Payments platform is its ability to process transactions and leverage the resulting data to feed the Reconciliation, Risk, and Analytics Applications.

Transactions represent a central role in the Payshop Online Payments platform, around which all its components have been devised. This led us to create an API solely dedicated to performing operations on transactions - the Processing API. These operations include the mechanisms mentioned below and can be applied to multiple types of elements. In the Processing API we highlight `charge`, `instrument`, `payment`, `reversal` and `refund`.

01 Create charge, instrument and payment
Multiple single events make up the different platform elements. Quickly understanding how to create them and the requirements needed to do so, can speed up your process. Explore how to create charges, instruments and payments.

02 Refund and reversal
Sometimes transactions do not carry their natural course. When it comes to interacting with consumers about reversing a charge or setting up a refund, you can use our Processing API.



Charge

A charge represents the merchant request to either pull funds or push funds into a customer's account.



Instrument

The instrument encompasses the data used to authenticate the customer.



Payment

A payment is a transaction authorization from a provider.



Reversal

Reverse a payment before it officially goes through.



Refund

After the transaction has been completed and before the customer has filled an official dispute, refunds are the right method to reverse a payment.



Channels

Combine payment methods and providers of choice. Each channel opens a world of opportunity for your payment operations.

05.1_Charge

A `charge` represents the merchant request to either pull funds or push funds into a user's account. When dealing with charges, you should be mindful of the payment channel, also known as `charge_type`, the amount, the currency and the necessary metadata for transaction reconciliation, such as `user ID` and `order ID`. The charge is the precursor to the `instrument`.

GET /v2/charges/types

Method	Path	Description
POST	Sandbox https://switch-processing.teya.com/v2/charges/types Production https://switch-processing.teya.com/v2/charges/types	List the available charge types, meaning payment methods. This list includes all the fields that should be displayed on checkout and filled out by the customer.

REQUEST

```
$ curl GET https://switch-processing.teya.com/v2/charges/types -u publicKey:
```

RESPONSE : HTTP 201

```
{
  "collection": [
    "...",
    {
      "id": "card_onetime",
      "payout": false,
      "label": "Card One-Time",
      "capture_on_creation": true,
      "schema_merchant": {
        "type": "object",
        "properties": {
          "enabled3ds": {
            "type": "boolean"
          }
        }
      },
      "schema": {
        "title": "Credit/Debit Card",
        "type": "object",
        "required": ["name", "number", "expiration_month", "expiration_year", "cvv"],
        "properties": {
          "expiration_month": {
            "minimum": 1,
            "type": "integer",
            "maximum": 12,
            "title": "Expiration Month"
          },
          "cvc": {
            "minLength": 3,
            "maxLength": 4,
            "type": "string",
            "title": "CVV"
          },
          "number": {
            "minLength": 14,
            "maxLength": 19,
            "type": "string",
            "title": "Card Number"
          },
          "expiration_year": {
            "type": "integer",
            "title": "Expiration Year"
          },
          "name": {
            "minLength": 3,
```

```

    "maxLength": 255,
    "type": "string",
    "title": "Cardholder Name"
  }
}
},
"ui_schema": {
  "name": {
    "ui:placeholder": "Full Name"
  },
  "number": {
    "ui:placeholder": "1111 2222 3333 4444"
  },
  "cvc": {
    "ui:placeholder": "123",
    "ui:help": "Last 3 digits on the back of the card"
  }
}
},
"..."
]
}

```

Response Parameters

collection

Array

An array of JSON objects that contains all the charge types currently active.

id String

This is a unique identifier for the charge type.

name String

Indicates whether the current charge type is designed to pay or receive funds.

- **true:** the charge type is a payout;
- **false:** the charge type is not a payout.

label String

User-friendly name that describes the charge type.

capture_on_creation string

Indicates whether the current charge type is a capture on creation transaction flow. It defines whether or not a payment should be requested on a successful instrument creation.

- **true:** the charge type captures funds on creation;
- **false:** the charge type does not capture funds on creation.

schema_merchant JSON Schema Object

This field specifies the parameters to be completed on the merchant side. These are the parameters that should be included in `instrument_params` and cannot be overridden by the customer, such as `enable3DS`.

type String

The data type included in this property.

properties JSON Object

A JSON Object documenting each of the properties that should be collected from the user for the charge type in question.

schema JSON Schema Object

Documents the data that should be collected for this charge type. This data can vary between different payment methods. To better understand the different requirements for each provider check out Integration Resources.

title String

The user interface identifier for the charge type, meaning payment method.

type String

The data type included in this property.

required Array

List of mandatory fields the customer should fill out to proceed with the charge.

properties JSON Object

A JSON object documenting the available payment provider specific parameters and data types.

minimum Number

The minimum value supported for this property. It applies when the data type is a number.

maximum Number

The maximum value supported for this property. It applies when the data type is a number.

type String

The data type of the property being collected.

title String

The user interface identifier for the property being collected.

minLength Number

The minimum supported length for this property. It applies when the data type is a string.

maxLength Number

The maximum supported length for this property. It applies when the data type is a string.

ui_schema JSON Schema Object

This JSON object contains the user interface information used to aid in rendering the required input collection forms. It is useful when using Dynamic Forms.

ui:placeholder String

Contains the placeholder text that should be added to the input of the property. This exemplifies to the user the required input.

ui:help Number

Contains helpful indications for the user regarding the property in question.

POST /v2/charges

Method	Path	Description
POST	Sandbox https://switch-processing.teya.com/v2/charges	Creates a new charge for a given transaction.
	Production https://switch-processing.teya.com/v2/charges	

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/charges -u accountId:privateKey -d '{
  "charge_type": "card_onetime",
  "amount": 42,
  "currency": "EUR",
  "metadata": {"orderId": "1337"},
  "events_url": "https://merchant.com/events",
  "redirect_url": "https://merchant.com/redirect",
  "instrument_params": {"descriptor": "D891220"},
  "channels": ["card_onetime_acapture"]
}'
```

Request Parameters

charge_type String **Required**

Payment method selected by the customer.

amount Number **Required**

Amount of the transaction in question.

currency String **Required**

Currency used in this transaction.

metadata JSON Object

Any metadata that uniquely identifies this transaction in your system. This field is used to identify this transaction when handling events, searching for transactions on the Dashboard, dynamically routing transactions or managing Risk.

events_url String

HTTPS merchant server-side webhook where events will be handled.

redirect_url String

HTTPS URL, or URL schemas when considering mobile apps, to send users back to. This field applies to payment methods that require customer redirection, such as Paypal or 3DS enabled credit cards.

instrument_params JSON Object

With this field the merchant can pass parameters which will be used when creating the instrument.

failure_code String

For charge elements that failed to be created, this field will document the associated error code.

failure_description String

Documents a user interface description of why this particular charge element failed to be created.

channels Array

Indicates which channels you use for the transaction. In case it is empty the default channel will be used.

RESPONSE : HTTP 201

```
{
  "id": "ceb69ab2eeb161ee6ed4906bff883dc1c82f3fb95f1859f1",
  "external_ids": null,
  "charge_type": "card_onetime",
  "charge_type_label": "Card One-Time",
  "amount": 42.0,
  "currency": "EUR",
```

```
"events_url": "https://merchant.com/events",
"redirect_url": "https://merchant.com/redirect",
"metadata": {
  "orderId": "1337"
},
"instrument_params": {
  "descriptor": "D891220"
},
"failure_code": null,
"failure_description": null,
"channels": [
  {
    "id": "704d26ef06980411b178d5436294b8d99e443abc5b1ead14",
    "label": "card_onetime_acapture",
    "processor": "acapture"
  }
],
"confirmed": true,
"created_at": "2020-07-22T15:23:29.253599+00:00",
"expires_at": null,
"updated_at": "2020-07-22T15:23:29.253650+00:00",
"request_log": {
  "ip_address": "149.90.219.7",
  "country": "PT",
  "library_version": null
},
"external": false
}
```

Response Parameters

id String

The charge ID that can be used once to create an instrument.

charge_type String

Identifier of the charge type, or payment method.

charge_type_label String

User-friendly identifier of the charge type, or payment method.

amount Float

Amount set for the charge.

Currency String

The currency used in the charge.

schema_merchant JSON Schema Object

This field specifies the parameters to be completed on the merchant side. These are the parameters that should be included in `instrument_params` and cannot be overridden by the customer, such as `enable3DS`.

events_url String

HTTPS merchant server-side webhook where events will be handled.

redirect_url String

HTTPS URL, or URL schemas when considering mobile apps, to send users back to. This field applies to payment methods that require customer redirection, such as Paypal or 3DS enabled credit cards.

metadata JSON Schema Object

Any metadata that uniquely identifies this transaction in your system. This field is used to identify this transaction when handling events, searching for transactions on the Switch Dashboard, dynamically routing transactions or managing Risk.

instrument_params JSON Object

With this field the merchant can pass parameters which will be used when creating the instrument.

channels Array

Indicates which channels you use for the transaction. In case it is empty the default channel will be used.

confirmed Boolean

Indicates if the created charge was confirmed by the merchant using their private key.

- **true:** the charge was confirmed by the merchant using their private key;
 - **false:** the charge was not confirmed by the merchant using their private key.
-

events_url String

The instrument parameters used to create the charge

created_at String

The date when the charge was created.

expires_at Date

The date when the charge was last updated

request_log JSON Object

Contains information on the origin of the request, such as `country`, `ip_address`, `user_agent` and the `library_version`.

title String

The user interface identifier for the property being collected.

external Boolean

This boolean flag indicates if the current charge has been created in the Payshop Online Payments platform or if it was incorporated via external sources.

- **true:** the charge has been created by the platform;
 - **false:** the charge was created outside of the platform.
-

> Check out the following use cases

It is not possible to create a charge without registering all the required fields. Tending to this error case, a list with all the invalid fields and the respective errors is returned.

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/charges -u accountId:privateKey -d '{
  "charge_type": "card_onetime",
  "amount": 42
}'
```

Response Parameters

message String

String with the error details. In this case: "Invalid parameters".

parameters JSON Object

A list with all the invalid fields and the respective errors.

RESPONSE : HTTP 400

```
{
  "message": "Invalid parameters",
  "parameters": {
    "currency": [
      "This field is required."
    ],
    "events_url": [
      "This field is required."
    ]
  }
}
```

It is possible to pass instrument parameters when creating a charge. For example, we can use the `instrument_params` in the charge to enable 3DS, as follows.

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/charges -u accountId:privateKey -d '{
  "charge_type": "card_onetime",
  "currency": "EUR",
  "amount": 10,
  "metadata": {"orderId": "837232"},
  "events_url": "https://merchant.com/events",
  "instrument_params": {"enable3ds": true}
}'
```


RESPONSE : HTTP 200

```
{
  "id": "35ed95bfb772b94c4e59f91fcbef0f5618d46e3d5b2b7da5",
  "charge_type": "card_onetime",
  "amount": 10,
  "currency": "EUR",
  "confirmed": true,
  "instrument_params": {
    "enable3ds": true
  },
  "events_url": "https://merchant.com/events",
  "expires_at": "2018-06-21T10:32:49.241268+00:00",
  "channels": null,
  "charge_type_label": "Card One-Time",
  "redirect_url": "",
  "metadata": {
    "orderId": "837232"
  },
  "external_ids": null,
  "request_log": {
    "country": null,
    "ip_address": "100.10.10.10",
    "user_agent": "curl/7.54.0",
    "library_version": null
  },
  "created_at": "2018-06-21T10:27:49.241769+00:00",
  "updated_at": "2018-06-21T10:27:49.241800+00:00"
}
```

It is also possible to choose what channel you want to use for the transaction by making changes to the `channels` field in the charge.

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/charges -u accountId:privateKey -d '{
  "charge_type": "card_onetime",
  "currency": "EUR",
  "amount": 10,
  "metadata": {"orderId": "837232"},
  "events_url": "https://merchant.com/events",
  "channels": ["card_onetime_checkout"]
}
```

```
}'
```

05.2_Instrument

The instrument is the object used to initiate the transfer of funds. Instruments come in many shapes and sizes, they are collections of the authentication parameters for any given payment method. You should watch out for fingerprints and status when evaluating instruments. Authentication fields for a card can include parameters like cardholder_name, PAN, CVV and expiry_date, whereas payment methods like PayPal would only require a redirection_url. Each instrument generates one or more payment objects.

The following request examples describe a card_onetime instrument. Each charge type has its own required fields which you should be mindful of when setting up your requests. For more information on this topic, access Integration Resources.

POST /v2/instruments

Method	Path	Description
POST	Sandbox https://switch-processing.teya.com/v2/instruments Production https://switch-processing.teya.com/v2/instruments	Creates a new instrument for a given transaction.

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/instruments -u publicKey -d '{  
  "charge": "a325e88948799260d9d8319a3ddb79ff2f74bbf35f198b30",  
  "name": "John Doe",  
  "number": "4111111111111111",  
  "expiration_month": 12,  
  "expiration_year": 2030,  
  "cvc": "007"  
}'
```

Request Parameters

charge String **Required**

The identifier of the charge associated with this payment that was previously created.

name String **Required**

The cardholder name.

number Number **Required**

The credit card number.

expiration_month Number **Required**

Expiration month for the card being used.

expiration_year Number **Required**

Expiration year for the card being used. Mind this value in your tests, dates in the past can generate errors.

cvc Integer **Required**

The card verification code.

RESPONSE : HTTP 201

```
{
  "id": "e866b0f517e92bd392183e53450dd6bd87e8cdc35f199c3c",
  "external_ids": {
    "processor": "8ac7a49f737620a201737c0a4bfb669c",
    "descriptor": "3750.0784.9102 Switch CC"
  },
  "success": true,
  "status": "authorized",
  "failure_code": null,
  "failure_description": null,
  "params": {
    "name": "John doe",
    "expiration_month": 12,
    "expiration_year": 2030,
    "descriptor": "D891220",
    "card_bin": "411111",
```

```

"card_last_4_digits": "1111",
"bin": "411111",
"last_4_digits": "1111",
"card_bank": "JPMORGAN CHASE BANK, N.A.",
"card_brand": "VISA",
"card_country": "US",
"card_account_type": "CREDIT",
"bank": "JPMORGAN CHASE BANK, N.A.",
"bank_phone": "1-212-270-6000",
"brand": "VISA",
"country": "UNITED STATES",
"country_isoa2": "US",
"country_isoa3": "USA",
"country_isonumber": "840",
"type": "CREDIT"
},
"fingerprint":
"947cc6e0e908f1e7c2670a4e44a8194397966738d0fae5640f9889f2f1fbc16eef7dd19dc4603a5632d69087e6969ea
5de876bca6db687b922ac999a6fddfb56",
"reference": null,
"response": null,
"redirect": null,
"created_at": "2020-07-23T14:18:36.596701+00:00",
"updated_at": "2020-07-23T14:18:36.596722+00:00",
"request_log": {
  "ip_address": "149.90.219.7",
  "country": "PT",
  "user_agent": "PostmanRuntime/7.26.2",
  "library_version": null
},
"external": false,
"recurring": false,
"capture_on_creation": true,
"channel": {
  "id": "704d26ef06980411b178d5436294b8d99e443abc5b1ead14",
  "label": "card_onetime_acapture",
  "processor": "acapture"
},
"used": true,
"last_payment": {
  "id": "2dfba90fcc304376701a2691677d1e68713bb8075f199c3c",
  "success": true,
  "status": "success"
},
}

```

```
"charge": {
  "id": "a325e88948799260d9d8319a3ddb79ff2f74bbf35f198b30",
  "charge_type": "card_onetime",
  "charge_type_label": "Card One-Time",
  "amount": 42.0,
  "currency": "EUR",
  "created_at": "2020-07-23T13:05:52.777328+00:00"
}
```

Response Parameters

id String

The ID that uniquely identifies the instrument element for this transaction.

external_ids String

A JSON Object documenting any external ID that the payment provider has related to the current instrument, if applicable.

success String

This boolean flag indicates whether the current Instrument has been successfully created.

- **true:** the instrument has been successfully created;
 - **false:** the instrument has failed to be created.
-

status String

Documents the execution status for the instrument.

- **authorized:** the instrument was successfully authorized by the provider;
 - **invalid:** the instrument was considered invalid by the Payshop Online Payments platform or the Provider;
 - **pending:** indicates that additional information is still necessary to proceed forward.
-

failure_code Number

For instrument elements that failed to be created, this field will document the associated error code.

failure_description String

Documents a user interface description of why this particular instrument element failed to be created.

parameters JSON Object

Contains the parameters that identify the customer and/or the payment instrument.

fingerprint String

A unique, one-way hash fingerprint of the Instrument, which can be leveraged for risk prevention purposes.

reference String

Only applicable to payment methods that require the customer to complete the transaction asynchronously outside of the Payshop Online Payments platform. Documents the information required to the customer. The specific schema of this object will depend on the payment method being used.

response JSON Object

Contains the technical information returned by the provider when processing the current instrument, such as the card ECI code.

redirect JSON Object

Used for payment methods that require redirection, documents the information required to direct the user to the page where the transaction can be completed.

- **url:** the URL to where the user should be redirected;
- **method:** the HTTP method that should be used when performing the URL call;
- **params:** the parameters that should be passed along with the URL call.

created_at Date

Contains the date and time when the instrument element was originally created.

updated_at Date

Stores the date and time when the instrument element was last updated.

request_log JSON Object

Stores information about the location and browser used to create the instrument element, such as the country of the IP address or the user agent.

external Boolean

This boolean flag indicates if the current instrument has been created in the Payshop Online Payments platform or if it was injected via external sources.

- **true:** the instrument has been created by the platform;
- **false:** the instrument was created outside of the platform.

recurring Boolean

This boolean flag indicates if the current instrument is classified as recurring.

- **true:** it is a recurring instrument;
- **false:** it is not a recurring instrument.

capture_on_creation Boolean

This boolean flag indicates if the current instrument is classified as captured on creation.

- **true:** capture on creation instrument, the payment is automatically captured on the instrument authorization;
- **false:** not a capture on creation instrument.

channel JSON Object

Documents the properties of the channel that was used to process this given instrument element.

- **id:** identification of the channel being used;
- **label:** user-friendly designation of the channel being used;
- **processor:** provider associated with the transactions in this channel.

used Boolean

Indicates whether a successful payment was already completed with the instrument in question.

- **true:** this instrument was already used in a successful payment;
 - **false:** this instrument was not previously applied to a successful payment.
-

last_payment JSON Object

Contains details on the last payment performed with the instrument in question.

charge JSON Object

Describes the charge element that was used to create this instrument.

> Check out the following use case

For Payment Methods that require a `reference` JSON Object, such as Multibanco, the field will contain the structure below. Please note that this information should be shown to the customer.

REFERENCE

```
{
  "reference": {
    "fields": [
      {"field": "entity", "value": "815412", "label": "Entity"},
      {"field": "reference", "value": "412523632", "label": "Reference"},
      {"field": "value", "value": "10", "label": "Value"}
    ]
  }
}
```

GET `/v2/instruments/{id}`

Merchants are able to make `GET /v2/instruments` requests with both private and public credentials. The response is different for either case, as shown in the following examples.

Method	Path	Description
GET	Sandbox https://switch-processing.teya.com/v2/instruments/{id}	Gets the details of an instrument element using its ID.
	Production https://switch-processing.teya.com/v2/instruments/{id}	

REQUEST ACCOUNTID:PRIVATEKEY

```
$ curl -vX GET https://switch-processing.teya.com/v2/instruments/{id}
-u accountId:privateKey
```

RESPONSE: HTTP 200

```
{
  "id": "ca4cb4177f8f9b18726f3605f25c3fa5412a8f2e5f157233",
  "external_ids": {
    "processor": "8ac7a4a0736acff601736bc61b003fbf",
    "descriptor": "4645.9162.8814 Switch CC"
  },
  "success": true,
  "status": "authorized",
  "failure_code": null,
  "failure_description": null,
  "params": {
    "name": "John Doe",
    "expiration_month": 12,
    "expiration_year": 2020,
    "card_bin": "411111",
    "card_last_4_digits": "1111",
    "bin": "411111",
    "last_4_digits": "1111",
    "card_bank": "JPMORGAN CHASE BANK, N.A.",
    "card_brand": "VISA",
    "card_country": "US",
    "card_account_type": "CREDIT",
    "bank": "JPMORGAN CHASE BANK, N.A.",
    "bank_phone": "1-212-270-6000",
    "brand": "VISA",
    "country": "UNITED STATES",
    "country_isoa2": "US",
```

```

    "country_iso3": "USA",
    "country_isonumber": "840",
    "type": "CREDIT"
  },
  "fingerprint":
"947cc6e0e908f1e7c2670a4e44a8194397966738d0fae5640f9889f2f1fbc16eef7dd19dc4603a5632d69087e6969ea
5de876bca6db687b922ac999a6fddfb56",
  "reference": null,
  "response": null,
  "redirect": null,
  "created_at": "2020-07-20T10:30:12.132238+00:00",
  "updated_at": "2020-07-20T10:30:12.132259+00:00",
  "request_log": {
    "country": "PT",
    "ip_address": "149.90.219.7",
    "user_agent": "PostmanRuntime/7.26.1",
    "library_version": null
  },
  "external": false,
  "recurring": false,
  "capture_on_creation": true,
  "channel": {
    "id": "9a48d37466f9323dc8c305fbf082fe5b80ca75b95af9ba98",
    "label": "wirecard",
    "processor": "acapture"
  },
  "used": true,
  "last_payment": {
    "id": "d1d44207a96f9c10c6ab7a6e43309937045453c65f157234",
    "success": true,
    "status": "success"
  },
  "charge": {
    "id": "3307721a72908e1a7a68d81ea9e08b758b63a87c5f157224",
    "charge_type": "card_onetime",
    "charge_type_label": "Card One-Time",
    "amount": 100.0,
    "currency": "EUR",
    "created_at": "2020-07-20T10:29:56.768942+00:00",
    "metadata": {
      "name": "Maria"
    }
  }
}

```

Response Parameters

id String

The ID that uniquely identifies the instrument element for this transaction.

external_ids String

A JSON Object documenting any external ID that the payment provider has related to the current instrument, if applicable.

success String

This boolean flag indicates whether the current Instrument has been successfully created.

- **true:** the instrument has been successfully created;
 - **false:** the instrument has failed to be created.
-

status String

Documents the execution status for the instrument.

- **authorized:** the instrument was successfully authorized by the provider;
 - **invalid:** the instrument was considered invalid by the Payshop Online Payments platform or the provider;
 - **pending:** indicates that additional information is still necessary to proceed forward.
-

failure_code String

For instrument elements that failed to be created, this field will document the associated error code.

failure_description String

Documents a user interface description of why this particular instrument element failed to be created.

parameters JSON Object

Contains the parameters that identify the customer and/or the payment instrument.

fingerprint String

A unique, one-way hash fingerprint of the Instrument, which can be leveraged for risk prevention purposes.

reference String

Only applicable to payment methods that require the customer to complete the transaction asynchronously outside of the Payshop Online Payments platform. Documents the information required to the customer. The specific schema of this object will depend on the payment method being used.

response JSON Object

Contains the technical information returned by the provider when processing the current instrument, such as the card ECI code.

redirect JSON Object

Used for payment methods that require redirection, documents the information required to direct the user to the page where the transaction can be completed.

- **url:** the URL to where the user should be redirected;
- **method:** the HTTP method that should be used when performing the URL call;
- **params:** the parameters that should be passed along with the URL call.

created_at Date

Contains the date and time when the instrument element was originally created.

updated_at Date

Stores the date and time when the instrument element was last updated.

request_log JSON Object

Stores information about the location and browser used to create the instrument element, such as the country of the IP address or the user agent.

external Boolean

This boolean flag indicates if the current instrument has been created in the Payshop Online Payments platform or if it was injected via external sources.

-
- **true:** the instrument has been created by the platform;
 - **false:** the instrument was created outside of the platform.
-

recurring Boolean

This boolean flag indicates if the current instrument is classified as recurring.

- **true:** it is a recurring instrument;
 - **false:** it is not a recurring instrument.
-

capture_on_creation Boolean

This boolean flag indicates if the current instrument is classified as captured on creation.

- **true:** capture on creation instrument, the payment is automatically captured on the instrument authorization;
 - **false:** not a capture on creation instrument.
-

channel JSON Object

Documents the properties of the Channel that was used to process this given instrument element.

- **id:** identification of the channel being used;
 - **label:** user-friendly designation of the channel being used;
 - **processor:** provider associated with the transactions in this channel.
-

used Boolean

Indicates whether a successful payment was already completed with the instrument in question.

- **true:** this instrument was already used in a successful payment;
 - **false:** this instrument was not previously applied to a successful payment.
-

last_payment JSON Object

Contains details on the last payment performed with the instrument in question.

charge JSON Object

Describes the charge element that was used to create this instrument.

REQUEST PUBLICKEY

```
$ curl -vX GET https://switch-processing.teya.com/v2/instruments/{id}
-u publicKey
```

RESPONSE: HTTP 200

```
{
  "id": "ca4cb4177f8f9b18726f3605f25c3fa5412a8f2e5f157233",
  "success": true,
  "status": "authorized",
  "used": true,
  "last_payment": {
    "id": "d1d44207a96f9c10c6ab7a6e43309937045453c65f157234",
    "success": true,
    "status": "success"
  },
  "created_at": "2020-07-20T10:30:12.132238+00:00",
  "updated_at": "2020-07-20T10:30:12.132259+00:00"
}
```

Response Parameters

id String

The ID that uniquely identifies the instrument element for this transaction.

success String

This boolean flag indicates whether the current Instrument has been successfully created. true: the instrument has been successfully created. false: the instrument has failed to be created.

status String

Documents the execution status for the instrument.

- **authorized:** the instrument was successfully authorized by the provider;
 - **invalid:** the instrument was considered invalid by the Payshop Online Payments platform or the provider;
-

-
- **pending:** indicates that additional information is still necessary to proceed forward.
-

used Boolean

Indicates whether a successful payment was already completed with the instrument in question.

- **true:** this instrument was already used in a successful payment;
 - **false:** this instrument was not previously applied to a successful payment.
-

last_payment JSON Object

Contains the technical information returned by the provider when processing the current instrument, such as the card ECI code.

created_at Date

Contains the date and time when the instrument element was originally created.

updated_at Date

Stores the date and time when the instrument element was last updated.

> Check out the following use case

For Payment Methods that require a **reference** JSON Object, such as Multibanco, the field will contain the structure below. Please note that this information should be shown to the customer.

REFERENCE

```
{
  "reference": {
    "fields": [
      {"field": "entity", "value": "815412", "label": "Entity"},
      {"field": "reference", "value": "412523632", "label": "Reference"},
      {"field": "value", "value": "10", "label": "Value"}
    ]
  }
}
```

05.3_Payment

Verifying the status of payments is essential to any business. A payment is a transaction authorization from a provider. Payments can be synchronous or asynchronous, pay-ins or payouts, redirection-based, pre-payments or post-payments, one-time or recurring. The payment object contains the technical information returned by the provider when processing your payments. Every payment comes with a `charge_id` and `instrument_id`, completing the transaction cycle.

POST /v2/instruments

Method	Path	Description
POST	Sandbox https://switch-processing.teya.com/v2/payments	Creates a new payment for a given transaction.
	Production https://switch-processing.teya.com/v2/payments	

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/payments -u accountId:privateKey -d '{
  "instrument": "e866b0f517e92bd392183e53450dd6bd87e8cdc35f199c3c",
  "currency": "EUR",
  "amount": 42
}'
```

Request Parameters

instrument String **Required**

The unique identifier for the instrument element that was previously created for this transaction.

currency String **Required**

[ISO 4217](#) code that indicates the currency that will be used in the transaction.

amount Number **Required**

The amount that should be captured from the customer.

description String

An optional textual description of the payment to provide further context.

metadata JSON Object

This object allows you to pass any transaction related data points that may be useful to be displayed in the Dashboard for analysis purposes. This field will be used to identify this transaction when handling events, searching for transactions on the Dashboard, dynamically routing transactions or managing Risk.

RESPONSE : HTTP 201

```
{
  "id": "ed9fd723c40fada03b6ce783beb182376a85c735b3e2f27",
  "amount": 42,
  "currency": "EUR",
  "description": "",
  "external": false,
  "metadata": {},
  "success": true,
  "response": {
    "eci_code": "05"
  },
  "params": null,
  "refundable": true,
  "failure_description": null,
  "refunds": [],
  "instrument": {
    "id": "0d0e51462ef62787dcb711f3c7ec42d086a172f85b2b8ddc",
    "channel": {
      "processor": "checkout",
      "id": "85a557e4fdb6c8806f413bc75fabab162828e4f95b8e6390",
      "label": "card_onetime_checkout"
    }
  }
},
```

```
"charge": {
  "charge_type": "card_onetime",
  "charge_type_label": "Card One-Time",
  "id": "ceb69ab2eeb161ee6ed4906bff883dc1c82f3fb95f1859f1",
  "metadata": {
    "orderId": "837232"
  }
},
"external_ids": {
  "transaction_id_trunc": "918dd9382a17be5d060f9a8dd15674",
  "processor": "8a8294496421b8d3016422227f17687",
  "transaction_id": "918dd9382a17be5d060f9a8dd15674b047b7f5d15b3e2f27"
},
"request_log": {
  "country": "PT",
  "ip_address": "100.10.10.10",
  "user_agent": "curl/7.54.0",
  "library_version": null
},
"created_at": "2018-07-05T14:46:00.040018+00:00",
"updated_at": "2018-07-05T14:46:00.040037+00:00"
}
```

Response Parameters

instrument String **Required**

The unique identifier for the instrument element that was previously created for this transaction.

currency String **Required**

[ISO 4217](#) code that indicates the currency that will be used in the transaction.

amount Number **Required**

The amount that should be captured from the customer.

description String

An optional textual description of the payment to provide further context.

external Boolean

Indicates whether the current payment element was not created or not by the platform. **true**: the payment has not been processed by the platform. **false**: the payment has been created using the platform.

metadata String

An optional textual description of the payment to provide further context.

success Boolean

This boolean flag indicates if the current payment has been successfully executed.

- **true**: the payment has been successfully executed;
 - **false**: the payment has failed to be executed.
-

response JSON Object

Contains the technical information returned by the provider when processing the current payment, such as the card ECI code.

refundable Boolean

Indicates whether the current payment can be refunded back to the customer, as some payment methods do not support this operation.

- **true**: the payment can be refunded back to the customer;
 - **false**: the payment cannot be refunded back to the customer.
-

failure_description String

Used for scenarios where the payment fails to be created, this parameter will contain a textual description of the error.

refunds Array

Stores the properties of the refund elements related to this payment.

instrument JSON Object

This JSON Object stores the properties of the instrument element related to this payment, as described in the previous sections

charge JSON Object

This JSON Object stores the properties of the charge element related to this payment, as described in the previous sections.

external_ids JSON Object

Documents the provider ID for the current payment, if available.

request_log JSON Object

Stores information about the device that was used to create the current payment element, such as `country`, `ip_address`, `user_agent`, and `library_version`.

created_at Date

Indicates the date and time when the current payment element was created.

updated_at Date

Documents the date and time when the last update was performed to the current payment.

05.4_Reversal

A reversal is a transaction cancellation previous to clearing. The `reversal` object translates both reversal and void processes. A void refers to a pre-authorization cancellation.

Reversals stop the payment process in an earlier stage, before refunds or disputes become necessary. This translates into less hassle for your customer and smaller fees for your business. Reversals can be applied to `instruments`, `payments`, or `refunds`.

**Keep in mind!**

The correct way of performing this operation is by creating a reversal and not by deleting a payment, instrument, or refund.

POST /v2/reversals

Method	Path	Description
POST	Sandbox https://switch-processing.teya.com/v2/reversals	Applies a reversal on the instrument, payment, or refund element, thus canceling the transaction authorization. Used in auth-capture or recurring transaction flows.
	Production https://switch-processing.teya.com/v2/reversals	

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/reversals -u accountId:privateKey -d '{
  "object_type": "instrument",
  "object_id": "ed9fdf723c40fada03b6ce783beb182376a85c735b3e2f27"
}'
```

Request Parameters

object_type String **Required**

Indicates the object type intended for reversal. Currently you are able to apply reversal to instrument, payments, and refunds.

object_id String **Required**

The ID that uniquely identifies the element to participate in the reversal.

RESPONSE

```
{
```

```
"reversal": {
  "id": "3f21bd06113cc34a27c50633491886091da99ea45cf7ca1a",
  "external_ids": {
    "processor": "8ac7a4a16b277c82016b27ed781065ed"
  },
  "object_id": "846ed631e9ebb72b867d38bd6d3f39a5bc9cd2025cf7c9fb",
  "object_type": "instrument",
  "amount": 10,
  "initiated_by": "merchant",
  "status": "success",
  "success": true,
  "failure_code": null,
  "failure_description": null,
  "request_log": {
    "country": null,
    "ip_address": "172.22.0.1",
    "user_agent": "PostmanRuntime/7.11.0",
    "library_version": null
  },
  "external": false,
  "created_at": "2019-06-05T13:56:43.586212+00:00"
}
```

Response Parameters

reversal JSON Object

A JSON object containing the details of the reversal element that was created as a result of the object being voided.

id String

The ID that uniquely identifies the reversal element that was created to void the instrument.

external_ids JSON Object

A JSON Object documenting any external ID that the payment provider has related to the current reversal, if applicable.

object_id String

The unique ID of the object to which the void pertains to, in this case the instrument element.

object_type String

The type of object to which the void pertains to, in this case the instrument element.

amount Number

The amount that was voided.

initiated_by String

Indicates the transaction entity that requested the reversal. Required since a reversal may be requested by the merchant or the provider.

status String

Documents the execution status for the void action.

- **success:** the instrument was successfully voided;
 - **error:** the instrument could not be voided;
 - **pending:** the instrument is in the process of being voided, used for providers that have an asynchronous process.
-

success Boolean

This boolean flag documents if the void operation has been completed successfully.

- **true:** the instrument was successfully voided;
 - **false:** the instrument has not yet been voided.
-

failure_code Number

For reversal elements that failed to be created, this field will document the associated error code.

request_log JSON Object

Stores information about the location and browser used to create the reversal element, such as the country of the IP address or the user agent.

external JSON Object

This boolean flag indicates if the current reversal element has been created in the Payshop Online Payments Platform or was later injected via external sources.

- **true:** the reversal has been created by the Platform;
 - **false:** the reversal was created outside of the Platform.
-

created_at JSON Object

Contains the date and time when the reversal element was originally created.

05.5_Refund

In a refund the payment has already been settled, but you need to return the money to the customer, this means that with a refund you complete the transaction in reverse. When handling refunds, you should be aware of which payment needs to be refunded, the amount of said refund and the appropriate justification for it.

POST /v2/refunds

Method	Path	Description
POST	Sandbox https://switch-processing.teya.com/v2/refunds	Creates a new refund for a given payment.
	Production https://switch-processing.teya.com/v2/refunds	

REQUEST

```
$ curl -vX POST https://switch-processing.teya.com/v2/refunds -u accountId:privateKey -d '{
```



```
"payment": "ed9fdf723c40fada03b6ce783beb182376a85c735b3e2f27",
"amount": 42,
"description": "Refund description"
}'
```

Request Parameters

payment String **Required**

The identifier of the payment previously created.

amount Number **Required**

The ID that uniquely identifies the element to participate in the reversal.

RESPONSE : HTTP 201

```
{
  "id": "d243e39cdf7f55f6a185235ac8ae8d67d469b29c5b44a1f0",
  "success": true,
  "status": "success",
  "amount": 42,
  "created_at": "2018-07-10T12:09:21.227579+00:00",
  "description": "Refund description",
  "external": false,
  "external_ids": {
    "descriptor": "0957.6372.2914 Switch CC",
    "processor": "8a829449646618a10164841895d1506c",
    "transaction_id": "7b6dfed71377d4065d873342422362121600fcef5b2bb2e9",
    "transaction_id_trunc": "7b6dfed71377d4065d873342422362"
  },
  "failure_code": null,
  "failure_description": null,
  "payment": {
    "id": "ed9fdf723c40fada03b6ce783beb182376a85c735b3e2f27"
  },
  "request_log": {
    "country": "PT",
    "ip_address": "11.123.1.23",
    "library_version": null,
  }
}
```

```
"user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.87 Safari/537.36"
},
"instrument": {
  "id": "0370cf3fa5f2d8b541d18bc6e39e28b01ae7aa0f5f2291f0",
  "channel": {
    "id": "9c62fb17a095f6dc32f3f36ea289f8a9473afcb95b1ead14",
    "label": "card_onetime_acapture",
    "processor": "acapture"
  }
},
"charge": {
  "id": "8f3e835780450782d228f395a12624f2fe819ed55f2291ec",
  "charge_type": "card_onetime",
  "charge_type_label": "Card One-Time",
  "metadata": {
    "order_id": "1235f4e5456d4e56f3",
  }
}
}
```

Response Parameters

id String

Unique identifier for the refund.

success Boolean

Flag indicating whether the refund was successfully created or not.

- **true:** the refund was successfully created;
 - **false:** the refund was not created.
-

amount Number

The refund amount. It can correspond to a partial refund or the total amount set in the payment.

created_at Date

The date and time when the refund was created.

description String

Description of the transaction in question.

external Boolean

This boolean flag indicates if the current charge has been created in the Payshop Online Payments platform or if it was incorporated via external sources.

- **true:** the charge has been created by the platform;
 - **false:** the charge was created outside of the platform.
-

failure_code Number

If the refund was not successful, this field contains the respective failure code.

failure_description String

If the refund was not successful, this field contains details about the failure.

payment JSON Object

Contains a summary of the payment object related to this refund.

request_log JSON Object

Contains information on the origin of the request, such as `country`, `ip_address`, `user_agent` and the `library_version`.

instrument JSON Object

This JSON Object stores the properties of the instrument element related to this payment, as described in the previous sections.

charge JSON Object

This JSON Object stores the properties of the charge element related to this payment, as described in the previous sections.

Next Steps

Understand how you can use settlements and sources to watch over your transactions and how to access agreements and disputes. Learn more about Reconciliation.

06_Dynamic Routing

Optimize your payments' performance.

Dynamic Routing provides real-time switching capabilities that allow you to select the best provider for a given transaction.

Through our Dynamic Routing components, you can add routing rules which guide transactions around a wide network of payment channels in real-time to maximize payment performance.



Get a better understanding of our Channels feature.

The combination of a payment method and a provider represents a channel in the Payshop Online Payments Platform. Understand how you can make the most of channels.



Routing Rules

Learn how to create and manage the routing rules that feeds Dynamic Routing.



Conditions and Priority

Set your conditions for Dynamic Routing. Work around the amount and processing currency towards the optimization of your transactions



Strategy

Choose the course of your transactions towards a more effective and cost-efficient payment operation.



Channels

The combination of a payment method and a provider, including the integration of choice, represents a channel in the Payshop Online Payments Platform.

06.1_Routing Rules

In case you are using the Dashboard , you can set up routing rules by accessing the specific channel on which you want to implement Dynamic Routing.

1. On the main navigation menu, select channels.
2. From your list of available channels, select the payment method you would like to apply Dynamic Routing to.
3. Add routing rules or edit the existing ones according to your strategy.
4. Drag and drop routing rules in the list to make changes to their priority levels.

Want to know more about the Dashboard?

By enabling the Dashboard you get access to a user-friendly management tool to oversee your transactions. Head over to Dashboard and learn more about it.



Dynamic Routing is established when an instrument is created.

If you are looking to understand when Dynamic Routing enters the lifecycle of your transaction, know that routing rules are triggered upon the creation of an instrument, after a charge has already been created and before it evolves into a payment.



POST/v1/router/rules

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/router/rules	Create a new Dynamic Routing rule.
	Production https://switch-gateway.teya.com/v1/router/rules	

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/router/rules?merchant_id={accountId} -u
accountId:merchantApiKey -d '{
  "charge_type": "card_onetime",
  "conditions": {
    "amount__gt": 50,
    "processing_currency__in": [
      "DOP"
    ]
  },
  "created_at": "2019-09-10T12:09:47.583718+00:00",
  "enabled": true,
  "priority": 2,
  "params": [
    {
      "channel": {
        "label": "card_onetime_cardnet",
        "processor": "cardnet",
        "id": "7581c500240442c1ec8b2f755583cdd4d488708f5ccb1ba5"
      },
      "value": 1
    }
  ],
  "type": "percentage",
  "id": "7d50afab5076d99ffcf84ce3abe7b056ab8844c45cd0190b"
}
```

Request Parameters

charge_type String

Identifier of the charge type (e.g. card_onetime, card_recurring, paypal).

conditions JSON Object

The routing conditions.

created_at String

The date when the rule was created.

updated_at String

The date when the rule was last updated.

enabled Boolean

Whether the rule is currently enabled or not.

priority Integer

The rule's priority. 0 being the top priority.

params Array

Required configurations to calculate the rule's output.

type String

Type of algorithm that will be used to select the provider.

id String

The rule's ID.

RESPONSE : HTTP 201

```
{
  "charge_type": "card_onetime",
  "conditions": {
    "amount__gt": 50,
    "processing_currency__in": [
```

```

    "DOP"
  ]
},s
"created_at": "2019-09-10T12:09:47.583718+00:00",
"enabled": true,
"priority": 2,
"params": [
  {
    "channel": {
      "label": "card_onetime_cardnet",
      "processor": "cardnet",
      "id": "7581c500240442c1ec8b2f755583cdd4d488708f5ccb1ba5"
    },
    "value": 1
  }
],
"type": "percentage",
"id": "7d50afab5076d99ffcf84ce3abe7b056ab8844c45cd0190b"
}

```

Result Parameters

charge_type String

Identifier of the charge type (e.g. card_onetime, card_recurring, paypal).

conditions JSON Object

The routing conditions.

created_at String

The date when the rule was created.

updated_at String

The date when the rule was last updated.

enabled Boolean

Whether the rule is currently enabled or not.

priority Integer

The rule's priority. 0 being the top priority.

params Array

Required configurations to calculate the rule's output.

type String

Type of algorithm that will be used to select the provider.

id String

The rule's ID.

GET/v1/router/rules

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/router/rules	List the available Dynamic Routing rules.
	Production https://switch-gateway.teya.com/v1/router/rules	

REQUEST

```
$ curl GET https://switch-gateway.teya.com/v1/router/rules?charge_type={chargeType}&merchant_id={accountId} -u accountId:merchantApiKey
```

RESPONSE : HTTP 200

```
{  
  "card_onetime": [  
    {  
      "charge_type": "card_onetime",
```

```

"conditions": {
  "amount__gt": 50,
  "processing_currency__in": [
    "DOP"
  ]
},
"created_at": "2019-09-10T12:09:47.583718+00:00",
"enabled": true,
"priority": 2,
"params": [
  {
    "channel": {
      "label": "card_onetime_cardnet",
      "processor": "cardnet",
      "id": "7581c500240442c1ec8b2f755583cdd4d488708f5ccb1ba5"
    },
    "value": 1
  }
],
"type": "percentage",
"id": "7d50afab5076d99ffcf84ce3abe7b056ab8844c45cd0190b"
},
{
  "charge_type": "card_onetime",
  "conditions": null,
  "created_at": null,
  "enabled": true,
  "updated_at": null,
  "priority": null,
  "params": [
    {
      "channel": {
        "label": "card_onetime_acapture",
        "processor": "acapture",
        "id": "9c62fb17a095f6dc32f3f36ea289f8a9473afcb95b1ead14"
      },
      "value": 1
    }
  ],
  "type": "percentage",
  "id": null
}
]
}

```

Result Parameters

charge_type String

Identifier of the charge type (e.g. `card_onetime`, `card_recurring`, `paypal`). If the request does not include `charge_type`, all charge types will be returned.

conditions JSON Object

The routing conditions.

created_at String

The date when the rule was created.

updated_at String

The date when the rule was last updated.

enabled Boolean

Whether the rule is enabled or not.

priority Integer

The rule's priority. 0 being the top priority.

params Array

Necessary configurations to calculate the rule's output.

type String

Type of algorithm that will be used to select the processor.

id String

The rule's ID.

Check out the following use case

It is not possible to create a routing rule without registering all the required fields. Tending to this error case, a list with all the invalid fields and the respective errors is returned.

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/router/rules?merchant_id={accountId} -u  
accountId:merchantApiKey -d '{  
  "charge_type": "card_onetime",  
  "type": "percentage"  
}'
```

Result Parameters

message String

String with the error details, in this case: "Invalid parameters".

parameters JSON Object

A list with all the invalid fields and the respective errors.

RESPONSE : HTTP 400

```
{  
  "message": "Invalid parameters",  
  "parameters": {  
    "params": [  
      "This field is required."  
    ]  
  }  
}
```

GET/v1/router/rules/{id}

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/router/rules/%7Bid%7D	Search for a specific routing rules using its ID.
	Production https://switch-gateway.teya.com/v1/router/rules/%7Bid%7D	

REQUEST

```
$ curl -vX GET https://switch-gateway.teya.com/v1/router/rules/{id}?merchant_id={accountId} -u  
accountId:merchantApiKey
```

RESPONSE : HTTP 200

```
{  
  "charge_type": "card_onetime",  
  "conditions": {  
    "amount__gt": 50,  
    "processing_currency__in": [  
      "DOP"  
    ]  
  },  
  "created_at": "2019-09-10T12:09:47.583718+00:00",  
  "enabled": true,  
  "priority": 2,  
  "params": [  
    {  
      "channel": {  
        "label": "card_onetime_cardnet",  
        "processor": "cardnet",  
        "id": "7581c500240442c1ec8b2f755583cdd4d488708f5ccb1ba5"  
      },  
      "value": 1  
    }  
  ],  
}
```

```
"type": "percentage",  
"id": "7d50afab5076d99ffc84ce3abe7b056ab8844c45cd0190b"  
}
```

Result Parameter

charge_type String

Identifier of the charge type (e.g. card_onetime, card_recurring, paypal).

conditions JSON Object

The routing conditions.

created_at String

The date when the rule was created.

updated_at String

The date when the rule was last updated.

enabled Boolean

Whether the rule is currently enabled or not.

priority Integer

The rule's priority. 0 being the top priority.

params Array

Required configurations to calculate the rule's output.

type String

Type of algorithm that will be used to select the provider.

id String

The rule's ID.

PUT/v1/router/rules/{id}

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/router/rules/%7Bid%7D	Substitute routing rule through its ID.
	Production https://switch-gateway.teya.com/v1/router/rules/%7Bid%7D	

REQUEST

```
$ curl -vX PUT https://switch-gateway.teya.com/v1/router/rules/{id}?merchant_id={accountId} -u  
accountId:merchantApiKey -d '{  
  "charge_type": "card_onetime",  
  "conditions": {  
    "amount__gt": 50,  
    "processing_currency__in": [  
      "DOP"  
    ]  
  },  
  "created_at": "2019-09-10T12:09:47.583718+00:00",  
  "enabled": true,  
  "priority": 2,  
  "params": [  
    {  
      "channel": {  
        "label": "card_onetime_cardnet",  
        "processor": "cardnet",  
        "id": "7581c500240442c1ec8b2f755583cdd4d488708f5ccb1ba5"  
      },  
      "value": 1  
    }  
  ],  
  "type": "percentage",  
  "id": "7d50afab5076d99ffcf84ce3abe7b056ab8844c45cd0190b"
```

```
}
```

Request Parameters

charge_type String

Identifier of the charge type (e.g. card_onetime, card_recurring, paypal).

conditions JSON Object

The routing conditions.

created_at String

The date when the rule was created.

updated_at String

The date when the rule was last updated.

enabled Boolean

Whether the rule is currently enabled or not.

priority Integer

The rule's priority. 0 being the top priority.

params Array

Necessary configurations to calculate the rule's output.

type String

Type of algorithm that will be used to select the provider.

id String

The rule's ID.

```
RESPONSE : HTTP 200
```

```
{
```



```
"charge_type": "card_onetime",
"conditions": {
  "amount__gt": 50,
  "processing_currency__in": [
    "DOP"
  ]
},
"created_at": "2019-09-10T12:09:47.583718+00:00",
"enabled": true,
"priority": 2,
"params": [
  {
    "channel": {
      "label": "card_onetime_cardnet",
      "processor": "cardnet",
      "id": "7581c500240442c1ec8b2f755583cdd4d488708f5ccb1ba5"
    },
    "value": 1
  }
],
"type": "percentage",
"id": "7d50afab5076d99ffcf84ce3abe7b056ab8844c45cd0190b"
}
```

Result Parameter

charge_type String

Identifier of the charge type (e.g. card_onetime, card_recurring, paypal).

conditions JSON Object

The routing conditions.

created_at String

The date when the rule was created.

updated_at String

The date when the rule was last updated.

enabled Boolean

Whether the rule is currently enabled or not.

priority Integer

The rule's priority. 0 being the top priority.

params Array

Required configurations to calculate the rule's output.

type String

Type of algorithm that will be used to select the provider.

id String

The rule's ID.

DELETE/v1/router/rules/{id}

Conditions set in routing rules can be related to five different variables: amount, processing currency, card BIN, card brand, and issuer country.

Condition Type	Condition	Example
Amount	greater than	amount__gt: 100

	greater than or equals	{amount__gte: 101}
	less than	{amount__lt: 103}
	less than or equals	{amount__lte: 102}
Processing Currency	in	processing_currency__in: ["EUR"]
	not in	processing_currency__not_in: ["USD"]
Card BIN	in	{card_bin__in: ["377750", "377753"]}
Card Brand	in	{card_brand__in: ["AMERICAN EXPRESS"]}
	not in	{card_brand__not_in: ["AMERICAN EXPRESS"]}
Issuer Country	in	{issuer_country__in: ["US"]}
	not in	{issuer_country__not_in: ["US"]}

POST/v1/router/rules/priority

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/router/rules/priority?merchant_id={accountId} -u
accountId:merchantApiKey -d '{
  "charge_type": "card_onetime",
  "priority": 2
}'
```

Request Parameter

charge_type String

Identifier of the charge type (card_onetime, card_recurring, paypal...)

priority Integer

The rule's priority (the lower, the more priority it has)

RESPONSE : HTTP 200

```
{
  "card_onetime": [
    {
      "charge_type": "card_onetime",
      "conditions": {
        "amount__gt": 50,
        "processing_currency__in": [
          "DOP"
        ]
      },
      "created_at": "2019-09-10T12:09:47.583718+00:00",
      "enabled": true,
      "priority": 2,
      "params": [
        {
          "channel": {
            "label": "card_onetime_cardnet",
            "processor": "cardnet",
            "id": "7581c500240442c1ec8b2f755583cdd4d488708f5ccb1ba5"
          },
          "value": 1
        }
      ]
    }
  ]
}
```

```

    },
    "type": "percentage",
    "id": "7d50afab5076d99ffcf84ce3abe7b056ab8844c45cd0190b"
  },
  {
    "charge_type": "card_onetime",
    "conditions": null,
    "created_at": null,
    "enabled": true,
    "updated_at": null,
    "priority": null,
    "params": [
      {
        "channel": {
          "label": "card_onetime_acapture",
          "processor": "acapture",
          "id": "9c62fb17a095f6dc32f3f36ea289f8a9473afcb95b1ead14"
        },
        "value": 1
      }
    ],
    "type": "percentage",
    "id": null
  }
]
}

```

Result Parameter

charge_type String

Identifier of the charge type (card_onetime, card_recurring, paypal...). If the request does not include charge_type, all charge types will be returned.

charge_type String

Identifier of the charge type (card_onetime, card_recurring, paypal...).

conditions JSON Object

The routing conditions.

created_at String

The date when the rule was created.

updated_at String

The date when the rule was last updated.

enabled Boolean

Whether the rule is enabled or not.

priority Integer

The rule's priority (the lower, the more priority it has).

params Array

Necessary configurations to calculate the rule's output.

type String

Type of algorithm that will be used to select the processor.

id String

The rule's ID.

06.2_Strategy

When considering the course of action to take with your Routing Rules there are two strategies to keep in mind: Percentage and Fallback.

Tackle acceptance rates and recover lost revenue.

Declined transactions cost you money and potential customers. When planning a future-proof payment structure, the acceptance rate is seen as the critical metric to account for. Dynamic Routing can help with that. Learn more about acceptance rates.



type: "percentage"

With percentage, you can distribute your transactions across any number of channels available to you. Under params you should list the `channel_id` of the channels you want to pick and in value the respective percentages of transactions intended for routing.

The probability of a given transaction landing on a specific channel is defined by an algorithm. With this in mind, the resulting distribution will not match precisely the one configured, but the bigger the number of transactions, the closer it will get to the set percentage goals. In the following example, we select two channels for routing, each of them holding 50% of the transactions.

STRATEGY PERCENTAGE EXAMPLE

```
{
  "charge_type": "card_onetime",
  "conditions": [
    { amount__gt: 10000},
  ],
  "params": [
    {"channel_id": "1a2b3c4d5e", "value": 0.5},
    {"channel_id": "2a3b4c5d6e", "value": 0.5}
  ],
  "type": "percentage"
}
```

type: "fallback"

With the fallback strategy, you are able to define which channels become the default for handling your transactions when a certain condition is verified. As you can observe in the request example that follows, the structure of the request is similar, minus the value attributed to the different percentages and changing the type of strategy. You are free to select multiple channels for this routing strategy.

STRATEGY FALLBACK EXAMPLE

```
{
  "charge_type": "card_onetime",
  "conditions": [
    { amount__gt: 10000},
  ],
  "params": [
    {"channel_id": "1a2b3c4d5e"},
  ],
  "type": "fallback"
}
```


07_Analytics

Turn big data into valuable insights.

Managing millions of transactions makes it harder to control unit costs and spot commission inconsistencies. Getting onboard with Analytics will transform your payment setup, promote efficiency, and push for bigger profits.

Our system aggregates all data and makes it searchable. We track commission variations across providers, markets, payment methods and fees. This way you can follow your payments lifecycle and find areas for improvement. Analytics is here to help you gain insight into your business and make the most of your payment operation.

1. **Sum Up**

Look back on your year, month, or week. Find all your results in one place with ease of access and user-friendly interpretations. With Analytics you can tap into the sum of your transaction information, weed out the unnecessary, and focus on what needs work.

2. **Filter**

Find the specifics. Filter Analytics by categories like device, country, or payment method. Quickly call out results for your payment strategies.

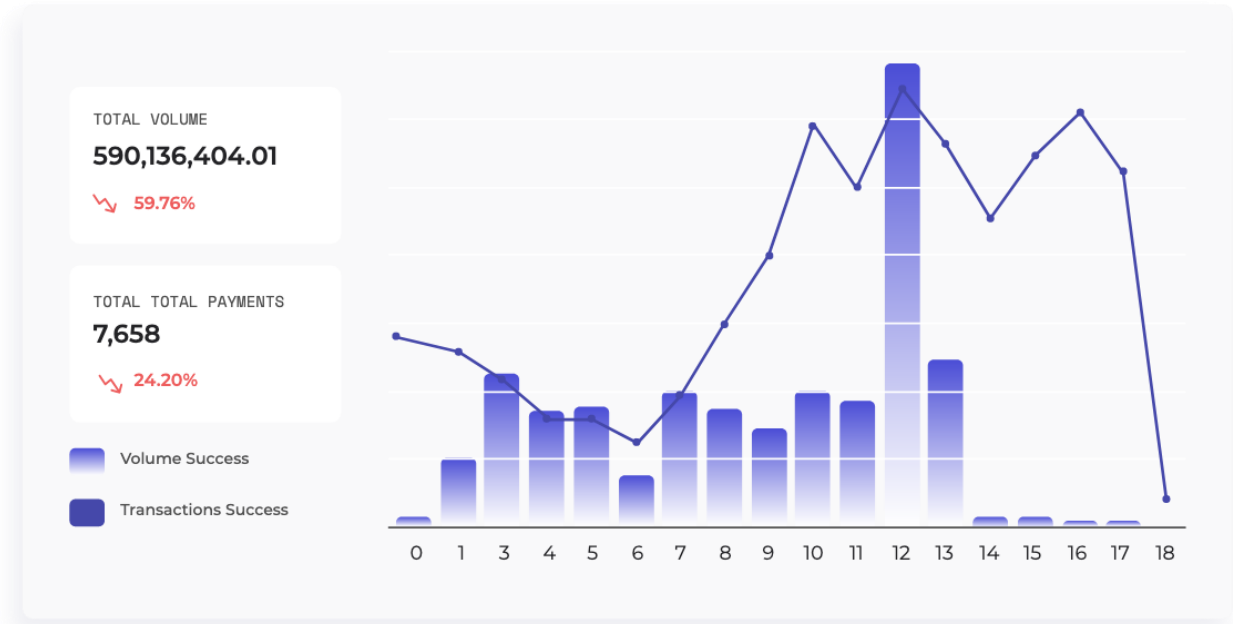
3. **Gain Insight**

We show you the money. Analyze rates, balances, and refunds. Figure out how to convert your efforts into bolder profits.

With Analytics you can gain insight into multiple variables essential to your payment operation. Here are some of them.

Have we mentioned the Dashboard?

The Dashboard offers a comprehensive and user-friendly interface to support your payment strategy.



Analytics section for totals.

Balance

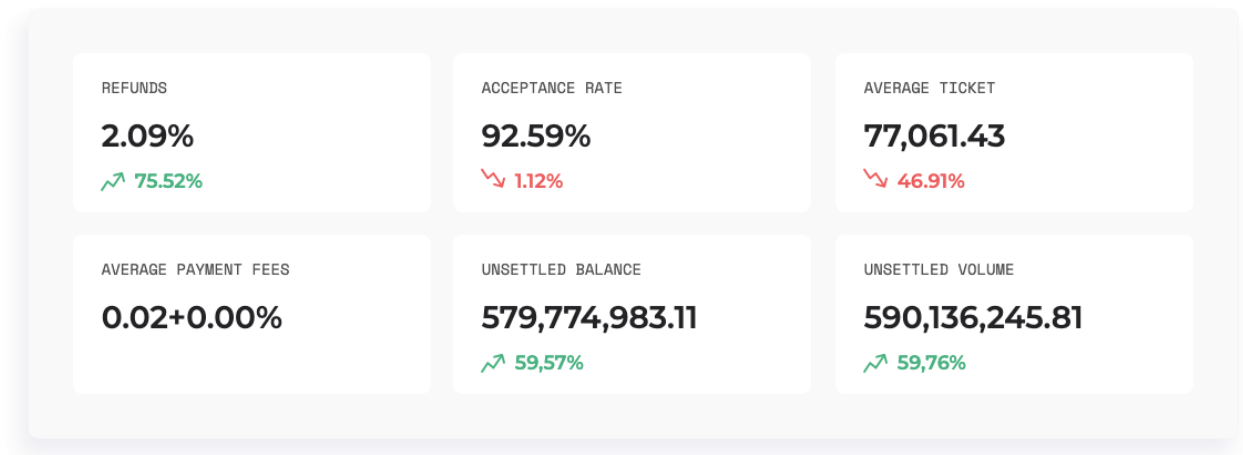
Balance is the total Payment amount deducted from refunds, fees, and disputes. It represents the amount outstanding on all providers' merchant accounts, which is set to be transferred to the merchant's bank accounts in the future. Compare different time periods and find out if your income is going up or down.

Dates and Date Granularity

Filter your Analytics by time period, understand the efficiency of your last seasonal campaign, explore your customers' habits. Sum up data by year, month, week, or day, zoom into the details, and display the big picture to your team and associates.

Failures

Filter failures, find out if mistakes are being repeated and fix them.



Analytics section for payment operations.

Refunds

Refund percentage calculated based on transaction volume. Notice your refunds and tap into customer satisfaction.

Average Payment Fees

Processing fees based on past settlements. See where your money goes, figure out if your partnerships carry on being beneficial, watch out for agreements.

Acceptance Rate

Consumers will not settle for anything less than perfection and uninterrupted functionality in the payment processing department. Understand their journey through Acceptance Rate, which mirrors the volume of successful transactions by the total volume of transactions.

Unsettled Balance

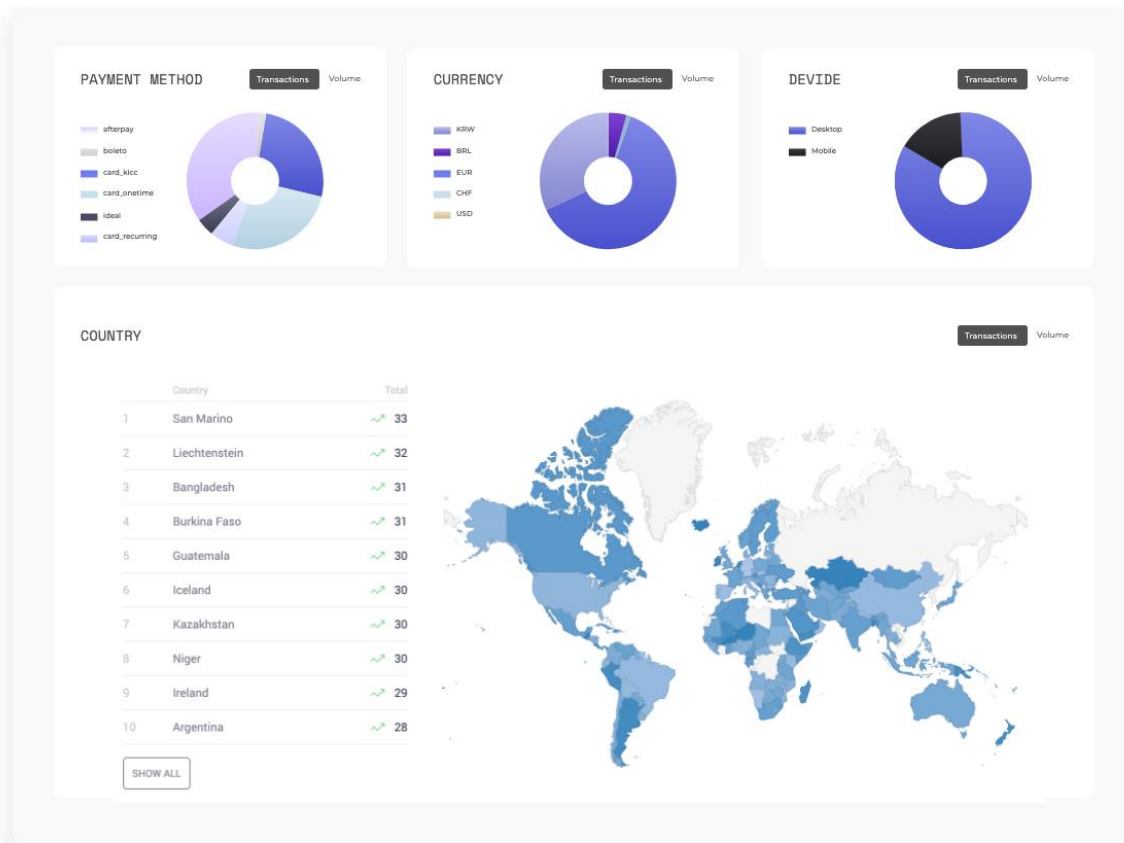
Processing fees based on past settlements. See where your money goes, figure out if your partnerships carry on being beneficial, watch out for agreements.

Average Ticket

The average of all successful payment amounts. Find the middle ground in purchases, be aware of what to expect from checkout.

Unsettled Volume

Payment volume, still outstanding in all providers' merchant accounts.



Analytics section for demographics.

Payment Method

The payment methods used by your customers. Identify the most popular payment options, understand if your customers might be missing something, pick the best options for your strategy.

Currency

Currencies used by your customers. Paint a clearer picture of your exchanges, see your strongest currencies.

Device

The device used by your customer during the purchase. Keep track of your customer journey, improve navigation, and user experience.

Country

The location of your customers. Understand where to place your international efforts, look for areas of improvement.

GET /v1/analytics

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/analytics	Get analytics data.
	Production https://switch-gateway.teya.com/v1/analytics	

REQUEST

```
$ curl -vX GET https://switch-gateway.teya.com/v1/analytics?merchant_id=accountId -u  
accountId:MerchantAPIKey  
{  
  "start_date": 2020-05-26T23:00:00,  
  "end_date":2020-05-27T10:00:00,  
}
```

Request Body Parameters

start_date Date **Required**

The start date for the results.

end_date Date **Required**

End date for the results.

RESPONSE : HTTP 200

```
{
  "filters": {
    "end_date": "2020-05-27T10:00:00",
    "merchant_id": "Lq9aVoz7eI852ID7ffNgGD9rr9Grv3ZQeE9dtfF0KvE6QxGQLwmjCPFkqmpP6td",
    "start_date": "2020-05-26T23:00:00",
    "success": "true"
  },
  "metrics": [
    {
      "data": [
        {
          "commissions": {
            "avg_estimated_payments_fixed_commissions": 2.2457779374775423e-05,
            "avg_estimated_payments_variable_commissions": 4.7295130244395855e-05,
            "avg_estimated_refunds_fixed_commissions": 0.0,
            "avg_estimated_refunds_variable_commissions": 0.0,
            "total_payments_fixed_commissions": 0,
            "total_payments_variable_commissions": 0,
            "total_refunds_fixed_commissions": 0,
            "total_refunds_variable_commissions": 0
          },
          "dimension": "payment_method",
          "label": "card_onetime",
          "payments_count": 22264,
          "payments_count_unsettled": 22264,
          "payments_volume": 634315.2,
          "payments_volume_unsettled": 634315.2,

```

```
    "refunds_count": 1,
    "refunds_count_unsettled": 1,
    "refunds_volume": 10.0,
    "refunds_volume_unsettled": 10.0,
    "settled_balance": 0.0,
    "unsettled_balance": 634274.6999799999
  }
],
"dimension": "success",
"label": true
}
]
```

Response Parameters

filters Array

Analytics filters.

end_date Date

End date for the results.

merchant_id JSON Object

ID of the Merchant account the Analytics data belongs to.

start_date Date

The start date for the results.

success Boolean

Status attributed to the transactions.

metrics Array

Metrics considered in the requested Analytics information.

Next Steps

Take a dive into our Reporting for more information on how to monitor your transactions.

08_Reporting

All of your transactions, summed up. When it comes to monitoring your transactions, organizing your archives, and following regulations, financial reporting is essential to your business. With the Payshop Online Payments Platform, you can do it all in one place by using our Reporting Application.

1. Create and Customize

Generate new reports and tap into the specifics of your transaction landscape. With the Reporting templates you are able to segment your transactional data, tap into new opportunities and push problematic areas to stand out.

2. Search and Filter

Find answers to all your stakeholders' questions in a report. Looking for specifics? Trying to sum up a month's worth of work? Reporting on your strongest currency? You can find it all with the Reporting application.

3. Share and Archive

Manage your archives and share files with the people that matter. Set up billing schedules customized to fit your clients. Handle all the paperwork from one single source.



Reports

Learn how to generate comprehensive reports from your transactions.



Reporting Templates

Understand what conditions are relevant to your analysis and how to set up the ideal report.



Reporting Schedules

Decide on periodicity and frequency in the creation of your reports.

08.1_Reports

Generating reports of your transactions should be a straightforward process that aids the management and upkeep of your company and accounts. Through Reporting you can get a quick overview of the transactions processed through the Payshop Online Payments platform. Reports are .CVS files generated from your list of transactions. You can filter reports following multiple headers related to charge, instrument, payment, refund, dispute, reversal and settlement events. Before being able to proceed with the creation of a report, you need to set up a Reporting template which defines the structure and conditions of your report. You can find more information on how to create Reporting templates in the next chapter.

Existing reports can be revisited by searching for specific designations, namely by the description given to the report, by date and by details like currency.

POST /v1/reporting/reports{id}

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/reporting/reports	Generate a new report.
	Production https://switch-gateway.teya.com/v1/reporting/reports	

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/reporting/reports?merchant_id=accountId -u
accountId:APIKey
{
  "description": "Example",
  "template": "862d7a6939de9eda2ce47af9cc2c848211ab5f0a5e7cc539"
}
```

Request Body Parameters

description String **Required**

Description of the new report being generated.

template String **Required**

Reporting template ID applied to the report being generated.

RESPONSE : HTTP 201 CREATED

```
{
  "id": "0e22990c2a581f3257394771cb52b85821a7b6da5f3be6a9",
  "type": "default",
  "status": "pending",
  "currency": "EUR",
  "description": "Example",
  "transactions_query": null,
  "transactions_file_type": null,
  "transactions_file_headers": null,
  "failure_description": null,
  "metadata": null,
  "merchant_id": "Lq9aVoz7eI852ID7ffNgGD9rr9Grv3ZQeE9dtfF0KvE6QxGQLwmjCPFkqmpP6td",
  "template": {
    "items": [
      {
        "label": "Example Item",
        "quantity_type": "fixed",
        "quantity_config": {
          "value": 2.0
        },
        "value_type": "tiered",
        "value_config": {
          "tiers": [
```

```
{
  {
    "value": 200,
    "minimum_quantity": 0
  }
]
},
"minimum_quantity": null,
"minimum_total": null,
"maximum_quantity": null,
"maximum_total": null
}
],
"id": "403d9c8fa29319515bbf454ebdc450146f156ad75f3bd75c"
},
"created_at": "2020-08-18T14:33:13.142267+00:00",
"updated_at": "2020-08-18T14:33:13.142332+00:00"
}
```

Response Parameters

id String

Identifier for the generated report.

type String

Type of report generated. It can either be default or billing.

currency String

Currency present in the report.

description String

Description attributed to the report.

metadata String

Metadata associated with the report.

merchant_id String

Identifier of the merchant to whom the report belongs to.

template JSON Object

Details and conditions defined by the report template.

created_at Date

Date when the report was created.

created_at Date

Date when the report was last updated.

DELETE /v1/reporting/reports{id}

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/reporting/reports{id}	Delete a report.
	Production https://switch-gateway.teya.com/v1/reporting/reports{id}	

REQUEST

```
$ curl -vX DELETE https://switch-gateway.teya.com/v1/reporting/reports/{id}?merchant_id=accountId -u  
accountId:APIKey
```

RESPONSE : HTTP 204 NO CONTENT

The server successfully processed the request and it is not returning any content.

GET /v1/reporting reports

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/reporting/reports	Get the list of existing reports.
	Production https://switch-gateway.teya.com/v1/reporting/reports	

REQUEST

```
$ curl -vX GET https://switch-gateway.teya.com/v1/reporting/reports?merchant_id=accountId -u  
accountId:APIKey  
{  
  "created_at__gte": "2020-07-31T23:00:00",  
  "created_at__lte": "2020-08-04T22:59:59",  
  "type": "default",  
  "currency": "EUR",  
  "description": "example"  
}
```

Request Body Parameters

created_at__gte Date

When filtering the results by date, this is the start date.

created_at__lte Date

When filtering the results by date, this is the end date.

type String

You can filter reports by type. Billing reports are set to `billing`, all other reports fit the default type.

currency String

Currency applied in the report.

description String

Designation attributed to the report.

RESPONSE : HTTP 200 CREATED

```
{
  {
    "collection": [
      {
        "id": "bce180137bfd5393633b18dd4748390b46fe6c565f299085",
        "type": "default",
        "status": "finished",
        "currency": "EUR",
        "description": "example",
        "transactions_query": {
          "type": [
            "charge",
            "payment"
          ],
          "currency": [
            "TRY",
            "USD"
          ],
          "created_at__gte": "2020-08-03T23:00:00",
          "created_at__lte": "2020-08-05T10:47:00"
        },
        "transactions_file_type": "csv",
        "transactions_file_headers": [
          {
            "name": "Merchant ID",
            "field": "merchant_id"
          }
        ],
        "failure_description": null,
        "metadata": {
          "cenas": "cenas"
        },
        "merchant_id": "Lq9aVoz7eI852ID7ffNgGD9rr9Grv3ZQeE9dtfF0KvE6Q",
        "template": {
          "items": [
            {
              "label": "asd",
              "value_type": "tiered",
              "value_config": {
                "tiers": [
                  {
                    "value": 2,
```

```

        "minimum_quantity": 0
    }
]
},
"maximum_total": null,
"minimum_total": null,
"quantity_type": "count",
"quantity_config": {
    "filters": [
        {
            "charge_type__in": [
                "card_onetime"
            ],
            "transaction_type__in": [
                "charge"
            ]
        }
    ]
},
"maximum_quantity": null,
"minimum_quantity": null
},
{
    "label": "poknj",
    "value_type": "tiered",
    "value_config": {
        "tiers": [
            {
                "value": 20,
                "minimum_quantity": 0
            },
            {
                "value": 10,
                "minimum_quantity": 100
            }
        ]
    },
    "maximum_total": null,
    "minimum_total": null,
    "quantity_type": "count",
    "quantity_config": {
        "filters": [
            {
                "charge_type__in": [

```

```

        "card_onetime"
      ],
      "transaction_type__in": [
        "payment"
      ]
    }
  ]
},
"maximum_quantity": null,
"minimum_quantity": null
}
]
},
"created_at": "2020-08-04T16:44:53.466096+00:00",
"updated_at": "2020-08-04T16:44:54.882597+00:00",
"items": [
  {
    "label": "asd",
    "quantity": 4.0,
    "original_quantity": 4,
    "value": 2.0,
    "total": 8.0,
    "original_total": 8.0,
    "template_item_details": {
      "quantity_type": "count",
      "quantity_config": {
        "filters": [
          {
            "charge_type__in": [
              "card_onetime"
            ],
            "transaction_type__in": [
              "charge"
            ]
          }
        ]
      }
    },
    "value_type": "tiered",
    "value_config": {
      "tiers": [
        {
          "value": 2,
          "minimum_quantity": 0
        }
      ]
    }
  }
]

```



```

    ]
  },
  "minimum_quantity": null,
  "minimum_total": null,
  "maximum_quantity": null,
  "maximum_total": null
}
},
{
  "label": "poknj",
  "quantity": 1.0,
  "original_quantity": 1,
  "value": 20.0,
  "total": 20.0,
  "original_total": 20.0,
  "template_item_details": {
    "quantity_type": "count",
    "quantity_config": {
      "filters": [
        {
          "charge_type__in": [
            "card_onetime"
          ],
          "transaction_type__in": [
            "payment"
          ]
        }
      ]
    }
  },
  "value_type": "tiered",
  "value_config": {
    "tiers": [
      {
        "value": 20,
        "minimum_quantity": 0
      },
      {
        "value": 10,
        "minimum_quantity": 100
      }
    ]
  },
  "minimum_quantity": null,
  "minimum_total": null,

```

```

        "maximum_quantity": null,
        "maximum_total": null
    }
}
],
"total": 28.0
}
],
"filters": {
    "created_at__gte": "2020-07-31T23:00:00",
    "created_at__lte": "2020-08-04T22:59:59",
    "type": "default"
},
"pagination": {
    "page": 1,
    "per_page": 30,
    "total_pages": 1,
    "total_items": 1
}
}
}

```

08.2_Templates

Reports are automatically generated from your transaction data. You can schedule reports and also define different types of reports. This is where templates come in. When setting up your Reporting system, you should consider the information you want to collect from your reports, beyond getting the full list of transactions for a given time period, you can monitor specific information that is relevant to the course of your business.

1. See the big picture.

For instance, get the gross amount of payments made in the last month. Label: Gross Amount, Quantity: Sum, Sum Field: Amount, Charge type: Check all, Transaction type: Payment, Value: Fixed.

2. Zoom into the details.

For instance, get the sum of exchange rate expenditures in all my payments. Label: MSC Fixed, Quantity: Sum, Fixed: 0.19, Charge type: Check All, Transaction Type: Payment, Value: Fixed.

Report Feature	Conditions
Description	Descriptions are strings set by the user. You can label the Reporting template according to your needs.
Currency	You can select multiple currencies from the available list.
Label	Templates can have multiple items which represent conditions present in that template. The label is the name you attribute to these conditions.
Quantity Fixed Count Sum	You can analyze transactions in reports by counting the absolute number of transactions that fit your conditions and also by generating a sum of the values present in those transactions.
Value Fixed Tiered	You can segment the values present in your reports either by a fixed interval or multiple intervals or tiers.
Minimum/Caps Quantity Total	It is possible to cap the transactions present in your reports at a minimum or maximum quantity and value of transactions.

POST /v1/reporting/templates

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/reporting/templates	Create a new Reporting template.
	Production https://switch-gateway.teya.com/v1/reporting/templates	

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/reporting/templates?merchant_id=accountId -u  
accountId:APIKey  
{  
  "currency": "EUR",  
  "description": "Example01"  
}
```

Request Body Parameters

currency String **Required**

Currency applied to the reports following this template.

description String **Required**

Designation for the template being created.

label String **Required**

Designation for each item of the template being created.

quantity String **Required**

Operation to be performed on the transactions included in this report. It can be fixed, sum or count.

value String **Required**

How the results should be presented in the reports following this template, either fixed or tiered.

RESPONSE : HTTP 200 CREATED

```
{
  "id": "1da379eaa030cede73d8e425968ecdcff51418285f3bec60",
  "description": "Example01",
  "currency": "EUR",
  "items": [],
  "created_at": "2020-08-18T14:57:36.382929+00:00",
  "updated_at": "2020-08-18T14:57:36.383009+00:00"
}
```

Response Parameters

id String

Identification of the report template.

description String

Designation for this template.

currency String

Currency applied to the reports generated through this template.

created_at String

Date and time in which the report template was created.

updated_at String

Date and time in which the report template was last updated.

PATCH /v1/reporting/templates/{id}

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/reporting/templates/{id}	Make changes to an existing Reporting template.
	Production https://switch-gateway.teya.com/v1/reporting/templates/{id}	

REQUEST

```
$ curl -vX POST https://switch-gateway.teya.com/v1/reporting/templates/{id}?merchant_id=accountId -u accountId:APIKey {
  "currency": "USD"
}
```

Request Body Parameters

currency String

In this example, we are making changes to the currency applied to the reports under this template. You can make changes to any of the parameters available in the creation of a Reporting template.

RESPONSE : HTTP 201 OK

```
{
  "id": "1da379eaa030cede73d8e425968ecdcff51418285f3bec60",
  "description": "Example01",
  "currency": "USD",
  "items": [],
  "created_at": "2020-08-18T14:57:36.382929+00:00",
  "updated_at": "2020-08-18T14:57:36.383009+00:00"
}
```

Response Parameters

id String

Identification of the report template.

description String

Designation for this template.

currency String

Currency applied to the reports generated through this template.

created_at String

Date and time in which the report template was created.

updated_at String

Date and time in which the report template was last updated.

DELETE /v1/reporting/templates/{id}

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/reporting/templates/{id}	Delete a report template.
	Production https://switch-gateway.teya.com/v1/reporting/templates/{id}	

REQUEST

```
$ curl -vX DELETE https://switch-gateway.teya.com/v1/reporting/templates/{id}?merchant_id=accountId -u accountId:APIKey
```

RESPONSE : HTTP 201 OK

The server successfully processed the request and it is not returning any content.

08.3_Reporting Schedules

You can schedule your reports to be generated automatically on a particular date and with specific frequency to better match your monitoring needs. This can be accomplished by setting up a reporting schedule. You can create as many reporting schedules as you would like and associate them to previously set Reporting templates.

When creating a new Reporting schedule, you should keep in mind what Reporting template you intend to associate to this schedule, the frequency to which it should be set and also the date of the first occurrence for the data included and the timezone it should guide itself by.

POST /v1/reporting/schedules

Method	Path	Description
POST	Sandbox https://switch-gateway.teya.com/v1/reporting/schedules	Create a new Reporting schedule.
	Production https://switch-gateway.teya.com/v1/reporting/schedules	

REQUEST

```
$ curl -vX GET https://switch-gateway.teya.com/v1/reporting/schedules?merchant_id=accountId -u  
accountId:APIKey  
{  
  "base_report_description": "Repo"  
  "execution_timezone": "Europe/Lisbon"  
  "first_execution": "2020-05-05T18:00:00.000000+00:00"  
  "frequency": "monthly"  
  "report_type": "default"  
  "template": "862d7a6939de9eda2ce47af9cc2c848211ab5f0a5e7cc539"  
}
```


Request Body Parameters

base_report_description String

Title attributed to the Reporting schedule.

execution_timezone String

Timezone applied to the Reporting schedule.

first_execution Date

Starting date for the report information.

frequency String

Frequency in which the reports should be generated. It can be monthly, weekly, or daily.

report_type String

Reports should have the report_type set to default. Billing specific reports are set to billing.

template String

Template applied to the reports in this Reporting schedule.

RESPONSE : HTTP 201 CREATED

```
{
  base_report_description: "Repo"
  created_at: "2020-05-05T13:35:23.192929+00:00"
  execution_timezone: "Europe/Lisbon"
  first_execution: "2020-05-05T18:00:00.000000+00:00"
  frequency: "monthly"
  id: "c5190ed3006fa0797a64074ef5f619319fb342fc5eb16b9b"
  report_metadata: null
  report_type: "default"
  template: "862d7a6939de9eda2ce47af9cc2c848211ab5f0a5e7cc539"
  updated_at: "2020-05-05T18:00:00.336279+00:00"
}
```

Response Parameters

base_report_description String

Title attributed to the Reporting schedule.

created_at Date

Date in which this schedule was created.

execution_timezone String

Timezone applied to the Reporting schedule.

first_execution Date

Starting date for the report information.

frequency String

Frequency in which the reports should be generated. It can be monthly, weekly, or daily.

id String

Identifier for the reporting schedule.

report_metadata Array

Additional information included when creating the Reporting schedule.

report_type String

Reports should have the report_type set to default. Billing specific reports are set to billing.

template String

Template applied to the reports in this Reporting schedule.

updated_at Date

Date on which this schedule was last updated.

Next Steps

Jump into the interfaces, learn more about the capabilities of the Dashboard and how it can simplify your payments operations.

09_Dashboard

Centralized overview and full control of your payment operation.

The Dashboard offers a comprehensive and user-friendly interface to support your payment strategy. By allowing Merchants to quickly interact with Transactions, study Analytics, and tap into Settlements and Reports, the Dashboard makes setting up and managing your payment operation much simpler, quicker, and straightforward.



Dashboard Features

The Payshop Online Payments platform Components are just one click away in the Dashboard.



Dashboard Settings

Customize your Merchant account and define the settings for your operation.

Analytics

The insights provided by the Analytics' tab allow Merchants to verify the status of Transactions and spot opportunities for transaction optimization. At the same time, this part of the Dashboard offers an in-depth look at the landscape of Merchants' Transactions, be it by location, time, or device.

This overview can show results from a specific date or summarize the data acquired over the week, month, or year.

Filters can aid navigation in the Analytics tab. These filters refer to Payment Methods employed, currency, and country, and device used for Payment. It is also possible to filter by the success or failure of the Transactions and by date.

Filter	Description
Payment Method	The Payment method dropdown mirrors the chosen channels. The user is able to compare Transactions between channels and understand the status of each Payment Method in regards to its Merchant. (e.g. Alipay, Paypal, Sofort).
Currency	The currency applied to Transactions can also filter the results presented in the Analytics tab.
Country	Understanding the landscape of Transactions for particular countries or groups of countries may prove very useful for Merchants.
Device	Consumers' choice of device for payments allows merchants to understand which platforms bring in more or less Payments. (Desktop, Tablet, Mobile).
Success/Error	Analytics on successful Transactions and failed Transactions are available to the user through filtering.
Dates Granularity	Users can compare the Payment's flow between different hours, days, months or years.
Start Date and End Date	Users can also tap into the Payments' flow over a specific period of time by setting up a start and an end date to their Analytics results.
Compare	Users can compare all the Analytics results from two instance perspectives. This means it is possible to compare the Transactions between two different payment methods, currencies, countries, devices or periods of time.

The first data display shown in Analytics is balance. Here users can check the total volume of transactions and the total number of payments over a day, week, month, or year.

In the Dashboard, users can tap into all sorts of additional information. This includes the percentage of refunds, the acceptance rate, and the average Transaction amount, and the average Payment fees. The unsettled balance is also on display so Merchants can check the amount of funds still outstanding throughout their Providers' base.

Details on Transactions range from the split of Payment Methods used, the currency applied and the device used for payment credential collection in the transaction. Information like this is useful to the user as it provides insight into customers' payment environment.

It is also possible to understand the geographical source of Payments by paying attention to the mapped distribution per country. Customers' locations appear listed on this interactive display.

Transactions

When it is time to dive deep into the Transactions going through your Payment Channels, everything is laid out for you on the Transactions tab. This includes Payment Methods, Providers involved, and the status of Transactions. Merchants can generate reports with the data they see fit and export particular transactions.

Whilst generating a Report, it is possible to select the metadata and parameters you are looking for, be it for Charges, Instruments or Payments. After filtering transaction data, it's possible to export both in a CSV or Excel file. You can observe transactions through a simple Charge-Instrument-Payment logic or pinpointing all Lifecycle Events.

When watching out for transactions, quickly understanding the status of each one is very important. There are five possible statuses for transactions in the Dashboard. Invalid instruments are highlighted by default.

Status	Description
Confirmed	Charge confirmed.
Success	Payment successful.

Authorized	Instrument authorized.
Pending	Instrument pending.
Invalid	Instrument invalid.

When consulting a Charge, you can find its id, amount, currency and status in the header. The Channel through which it was processed and the details from the request log which include the consumer's IP address and country. The Risk Rules and how the Charge abides by them or not can also be found in the Charge details.

Going into the details of each registered Instrument, you can observe its respective Charge, as well as the amount, currency and id. Besides the details already granted by the Charge, you will get access to the Instrument parameters, which describe the authentication parameters provided by your Customer, as long as they do not fall under the scope of sensitive data according to the relevant Payment Scheme, in which case we will not expose them for security reasons.

Following the Instrument comes the Payment, which is also registered with every Transaction and includes the results of the Payment process. Through the Payment page, you can also generate Refunds, be it full Refunds or partial Refunds.

Reporting

Under Reporting, you can set report templates for your Transactions. Through these report templates, you can download transaction data that is recurrently necessary to your operations or make a note of a specific sales strategy or Payment's time frame. Reports allow you to map out total Transactions for a specific date, understand the average amount per transaction, and the total profit. You can limit reports to a specific currency, charge_type, and transaction_type.

09.1_Dashboard Settings

Account Details

Under the User's Account option, it is possible to find account details, like name and email, and then set up two-factor authentication. One important detail to consider under Account is the personalization board. Here the user can select the language of the Dashboard, the time zone, the date format, and the state of the Merchant API (Live, Staging, Custom).

Account Details

In Account Details you can register the User-specific fields and personalize the User profile, Name, Email, Password.

Two Factor Authentication

The Dashboard offers Two Factor Authentication. It is under Account that you can generate the respective codes and register trusted devices.

Codes, Trusted Devices.

Personalization

You can change the language, time zone, and date format using your Account settings.

Language, Timezone, Date Format, Merchant API.

Merchants

You can search for existing Merchants by name under your Dashboard's logotype. To get access to the full list of merchants and respective account names and account id access the Merchants tab.

Merchant information is condensed in the Merchants tab. Here you can find the main merchant, its account id, and environment. You can also find the list of Sub-merchant Accounts, in case they exist, and search and manage said sub-merchant Accounts.

Merchant Details

You can come back to Merchant details to check your Environment, TEST/LIVE and your account id.

Environment, Account id, Approval.

Billing

Billing settings can be found under Merchant. In this section you can go over your billing history and set up a billing schedule.

History, Schedule, Frequency.

Sub-merchants

In the Merchants tab you can get an overview of your sub-merchant Accounts and access each of them for further details or to proceed with changes, namely to billing.

Account Name, Account id.

Settings

In Settings, you can create Processing API Keys and Merchant API Keys and manage the ones already in use. Settings also include permissions and authorization groups for users. This is the part of the Dashboard where you can hand out different permissions to different members of your organization so that they can access all the different Dashboard tools.

Processing API Keys

The Processing API Key set includes the Public Key and Private Key used during Authentication.

Description, Status.

Merchant API Keys

Merchant API Keys can be set per Authorization Group. These are necessary to manage different accounts and Users inside your operation.

Description, Status.

Users

In the Merchants tab you can get an overview of your sub-merchant Accounts and access each of them for further details or to proceed with changes, namely to billing.

Account Name, Email Address, Account id.

10_Annex

10.1_Payshop Channels

10.1.1_Card

Fields	Description
Channel ID	Unique Switch ID identifying the channel.
Charge Type	Charge type name.
Provider	Charge type provider.
Integration	Type of integration made between Switch and the Provider.
Client ID	The Client ID/Access Key ID to request access token.
Client Secret	The Client Secret/Access Key Secret to request access token.
Processing Channel ID	ID from the Checkout processing channel to be used.
Sub Entity ID	ID of the Sub Entity on behalf of which the payments will be processed.
Webhook Key	The Key used to sign/verify the webhooks.

Transactions Type	Transaction type to be used on 3DS transactions. By default it is assumed "Goods / Service Purchase".
Channel Label	Label identifying the channel (the default should be card_checkout_payfac).
Channel for sharing	Whether this channel will be used for sharing with sub-merchants, or for payment processing.
Enable	Whether the channel is active (ready to be used) or not.

10.1.2_MBWay

Fields	Description
Channel ID	Unique Switch ID identifying the channel.
Charge Type	Charge type name.
Provider	Charge type provider.
Integration	Type of integration made between Switch and the Provider.
Entity ID	Entity ID to use.
Client ID	Client ID provided by SIBS.

Terminal ID	Terminal ID provided by SIBS.
API Token	API Token provided by SIBS.
Webhook Secret for Encryption	The secret key to decrypt webhooks.
Channel Label	Label identifying the channel (the default should be mbway_sibs).
Channel for sharing	Whether this channel will be used for sharing with sub-merchants, or for payment processing.
Enable	Whether the channel is active (ready to be used) or not.

10.1.3_Multibanco

Fields	Description
Channel ID	Unique Switch ID identifying the channel.
Charge Type	Charge type name.
Provider	Charge type provider.
Integration	Type of integration made between Switch and the Provider.

Entity ID	Entity ID to use.
Client ID	Client ID provided by SIBS.
Terminal ID	Terminal ID provided by SIBS.
API Token	API Token provided by SIBS.
Webhook Secret for Encryption	The secret key to decrypt webhooks.
Channel Label	Label identifying the channel (the default should be multibanco_sibs).
Channel for sharing	Whether this channel will be used for sharing with sub-merchants, or for payment processing.
Enable	Whether the channel is active (ready to be used) or not.

10.1.4_Payshop Reference

Fields	Description
Channel ID	Unique Switch ID identifying the channel.
Charge Type	Charge type name.
Provider	Charge type provider.

Integration	Type of integration made between Switch and the Provider.
IBM Client ID	Client ID of Payshop app.
IBM Client Secret	Client Secret of Payshop app.
Payshop Client ID	Payshop Client ID.
Payshop Entity ID	Payshop Entity ID.
Channel Label	Label identifying the channel (the default should be payshop_reference).
Channel for sharing	Whether this channel will be used for sharing with sub-merchants, or for payment processing.
Enable	Whether the channel is active (ready to be used) or not.

10.2_Asynchronous payments

All channels used in the Payshop Online Payments platform, with the sole exception of Cards, are asynchronous payment methods: this means that, after the successful creation of an element *“charge”*, the element *“instrument”* will be also generated but its status shall remain as *“Pending”* until callback information from the channel's transaction provider entity is received. This information is sent via webhook to the Merchant account where the charge was made.