echo

How to Get Started with the API Documentation

Table of Contents

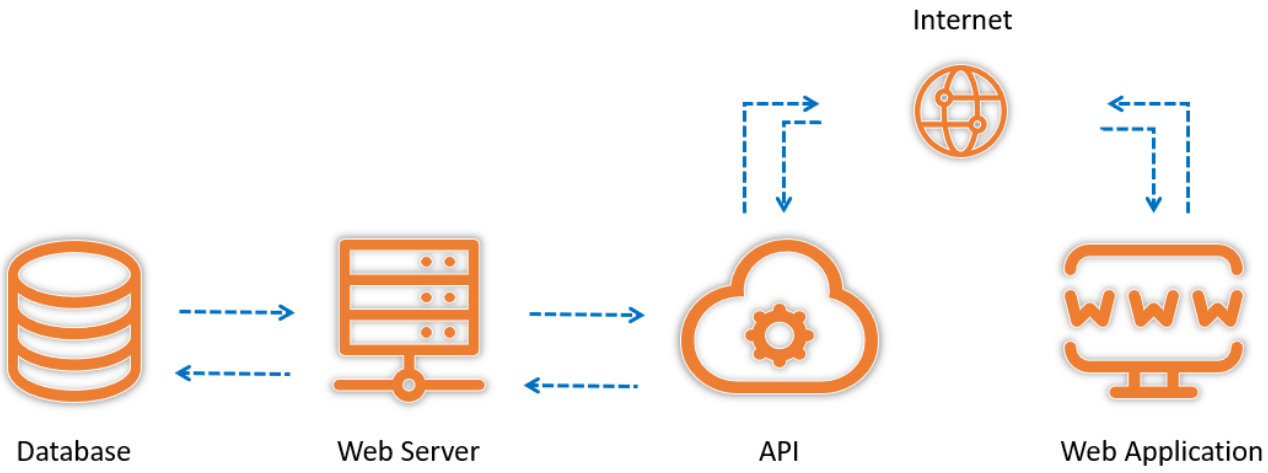
LINKS TO API DOCUMENTATION

B2B Gateway	https://docs.echoloyalty.com/api/b2b.html
B2C Gateway	https://docs.echoloyalty.com/api/b2c.html

** All additional links used in this document are just used as an example, and the correct ones will be shared with you via email during activation.*

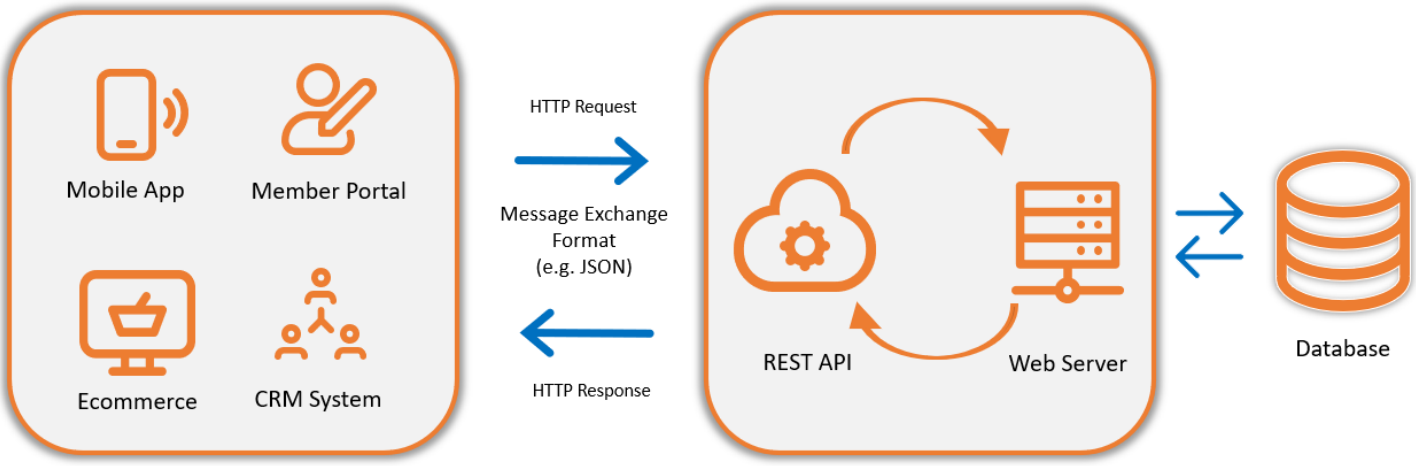
What is a REST API?

A REST API (Representational State Transfer Application Programming Interface) is a set of rules and conventions for building and interacting with web services. It allows **different software systems to easily communicate** over the internet using standard HTTP methods.



REST compliance systems, or REST-ful systems, are characterized by the stateless communication and the way they separate the code of a company's IT ecosystem from the ones of a server. In the REST architecture style, **the implementation of your IT ecosystem and the server can be done independently** without one knowing about the other. The code on your side can be changed at any time without affecting the operation of the server, and the code on the server side can be modified without affecting your operations.

REST Architecture

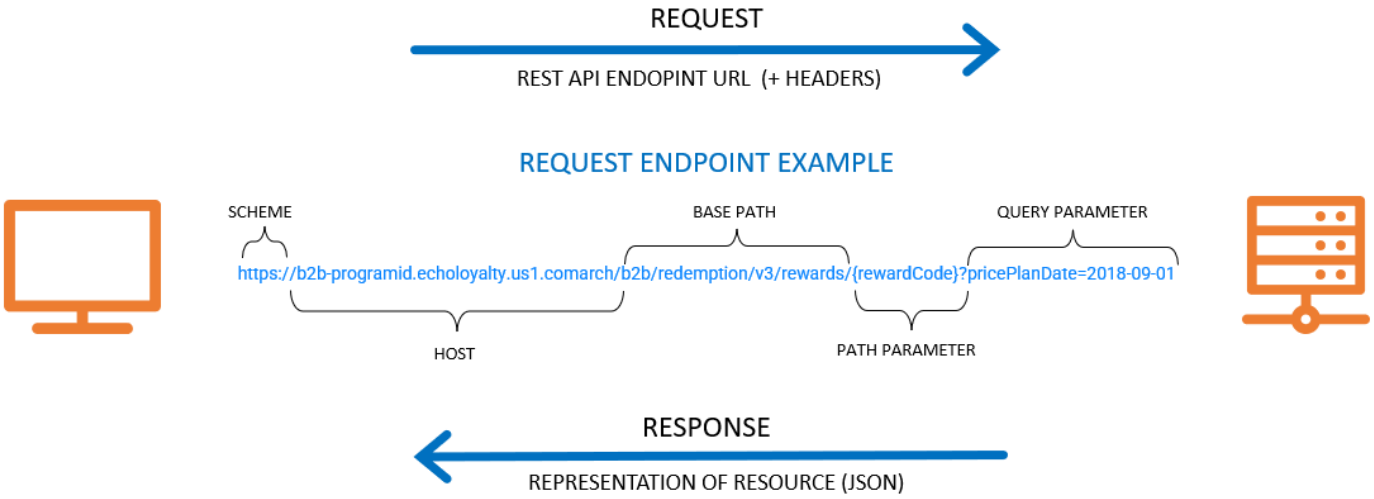


REST API in Echo Loyalty

As part of the REST architecture, you need to send a request to retrieve or modify particular resources on the servers and for the servers to send responses to these requests.

An API request usually consists of:

- **HTTP verb**, which defines what kind of operation has to be performed. The HTTP verbs used in Echo Loyalty are:
    - GET - used only to read data and not change it. They can be considered safe and call anytime, with no risk or corrupting or modifying the data. Multiple identical requests produce the same result as a single request.
    - POST – most often utilized to create new resources, in particular, to create subordinate resources. When creating a new resource, the POST request sent to the parent and to the server takes care of associating the new resource with the parent assigning an ID, new resource URI, etc.
    - PUT - utilized to replace the entire resource with a new representation, meaning that all the fields of the resource are sent in the request body, even if they are not modified.
    - PATCH - used to apply partial updates to a resource, meaning that only the fields that need to be changed are sent in the request body.
- REST API **endpoint URL** path to a resource
- **Headers**, which allow the client to pass along information about the request. API headers are like an extra source of information for each API call you make (e.g. Authorization, Accept-Language, X-Correlation-ID)**.** Their job is to represent the meta-data associated with an API request and response.
- an optional **message body** containing additional data



In Echo Loyalty, the endpoint URL path is a combination of:

1. Base URL – is the combination of scheme, hostname, and base path on the root level of the API.
    a. Scheme – this is the transfer protocol in the full URL for an API. In Echo Loyalty we use *HTTPS*
    b. Host
    c. Base Path
2. Path Parameter – In a typical REST scheme, the path portion of the URL represents entity class hierarchy. It is surrounded by curly brackets and offers a unique opportunity for developers to control the representation of a specific resource. '{rewardCode}' represents the parameter that is required for the call. Values must be given in correspondence with the format that is dictated by the URL path, and can't be omitted since it would mean changing the URL path.
3. Query Parameter – they should be at the end after the question mark '?' and can be defined as optional. They help determine specific content or action based on the data being delivered. The question mark separates path and query parameters. If you want to add multiple query parameters, an '&' sign is placed in between them to form what is known as a query string.

As a response to a request, we receive a **JSON (JavaScript Object Notation) file**, as a live representation of the resource. JSON is a text base data exchange format; it is a collection of key-value pairs where the key has to be of a string type and the value can be of any type: string, number, array, object, Boolean, or null.

The key must be enclosed in double quotation marks and separated by colons. There could be multiple value points defined and need to be separated by commas. The JSON data format doesn't allow for adding comments to the records.

Rest Api Calls Examples

HTTP Verb	**GET** retrieve a specific resource (by code) or a collection of resources
URL Endpoint	https://b2b-programid.echoloyalty.us1.comarch/b2b/redemption/v3/rewards/SFH9hdtHR57?pricePlanDate=2018-09-01
Request Body	no request body required
Response Body	

HTTP Verb	**POST** create a new resource
URL Endpoint	https://b2b-programid.echoloyalty.us1.comarch/b2b/trnprocessor/v3/coupon-redemptions?identifierNo=900000000023

Request Body	{
	"partner": "DEFAULT",
	"trnNo": "123000000001",
	"location": "SAW01",
	"date": "2019-08-24T14:15:22Z",
	"comment": "Example comment",
	"coupons": [
	{
	"couponNumbers": [
	"93262600002"
	],
	"quantity": 1
	}
	],
	"cashierId": "14050"
	}

Response Body	{
	"transactionId": 19070,
	"status": "B",
	"memberBalance": {
	"basicPoints": 23
	},
	"burnedCoupons": [
	{
	"couponNumber": "463463734735",
	"couponTypeCode": "GS4gksgd5gssg11",
	"useQuantity": 56
	}
	]
	}

HTTP Verb	**PUT /PATCH** update a specific resource (by code)
URL Endpoint	https://b2b-programid.echoloyalty.us1.comarch/b2b/profile/v3/customers/identifierNo=5870000012/tac-acceptance

Request Body	{
	"gdpr": {
	"personalDataProcessing": {
	"memberConsent": true,
	"acceptedVersion": 12
	}
	},
	"loyalty": {
	"communications": {
	"memberConsent": true,
	"communicationChannels": [
	{
	"code": "O"
	}
	]
	}
	}
	}

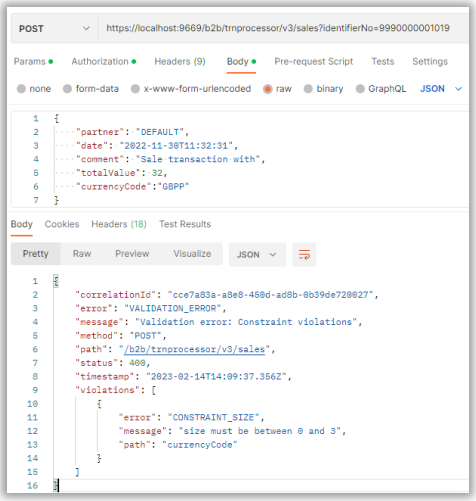
Response Body	no response body if successful

Response Codes

STATUS CODE	MEANING
200 (OK)	successful HTTP requests
201 (CREATED)	an HTTP request that resulted in an item being successfully created
204 (NO CONTENT)	nothing is being returned in the response body
400 (BAD REQUEST)	wrong request preparation
403 (FORBIDDEN)	not allowed to perform this method
404 (NOT FOUND)	resource not exist
410 (GONE)	no longer available
500 (INTERNAL SERVER ERROR)	unexpected failure

ERROR RESPONSE COMPONENTS

- correletationId – request token (When not provided, it is assigned authomatically by the system)
- error – machine readable error code (Represents the reason for the failure)
- message – human readable error message
- method – HTTP method
- path – REST API endpoint path (Source of the error)
- status = HTTP status code
- timestamp – current timestamp
- violations – list of occurred violations



Echo Loyalty App Versioning

2.12.2

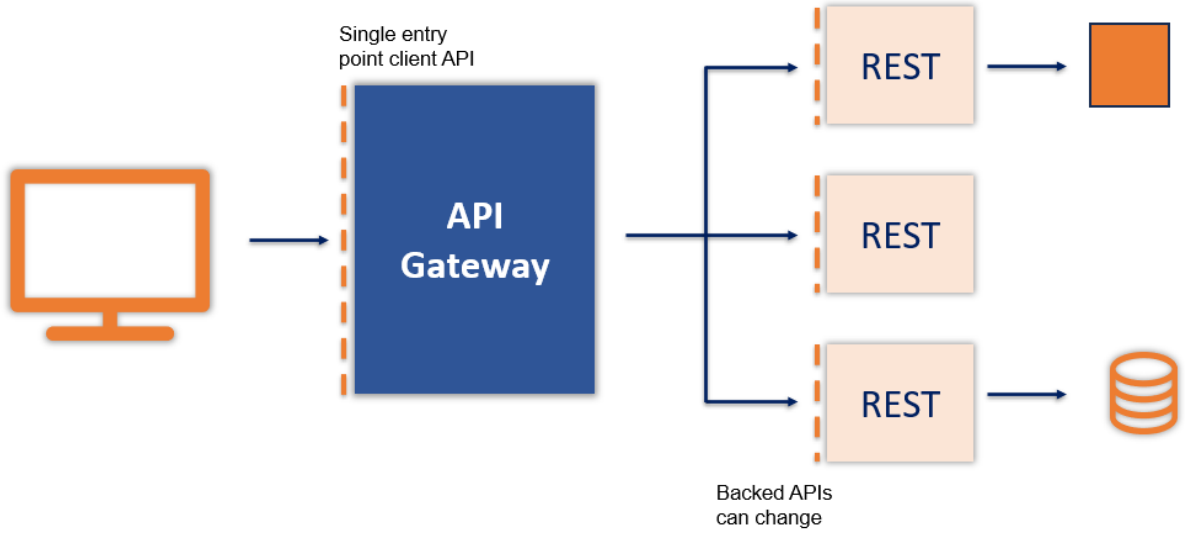
MAJOR VERSION	MINOR VERSION	PATCH VERSION
Ordinal number of version incremented if backward compatibility* of API would be corrupted. It applies to API version number filled during API methods calls	Ordinal number of version incremented every 2 weeks as we release new API version	Ordinal number of version incremented if the need reveal itself for i.e. 2.12.x RELEASE

*An API is backwards compatible if a client written against one version of that API will continue to work the same way against future versions of the API.

Echo Loyalty API Gateways

What is an API Gateway?

An API gateway is a virtual passage placed assuredly between an API and its various backend services. It takes care of invites or requests, matching them to the suitable stations or services for request/call processing, and sends them back to the target resource.



Echo Loyalty Gateways

A Gateway REST API streamlines signing up, signing in, management, operation of APIs, enhancing security and regulatory compliance through authentication and authorization. It provides central definition and management of security and other operational governance policies across multiple instances. The Gateway enables enterprises to standardize API and service delivery with high security, performance, and availability.

In Echo Loyalty, we can utilize two different Gateways:

1. **Business to Business (B2B) Gateway**

The B2B Gateway is used for third party back-office applications, ecommerce platforms and POS applications (physical store, vending machine, e-commerce) to communicate with Echo Loyalty.

2. **Business to Customer (B2C) Gateway**

The B2C Gateway is used for member frontends such as a member Mobile App or Web App to communicate with Echo Loyalty.

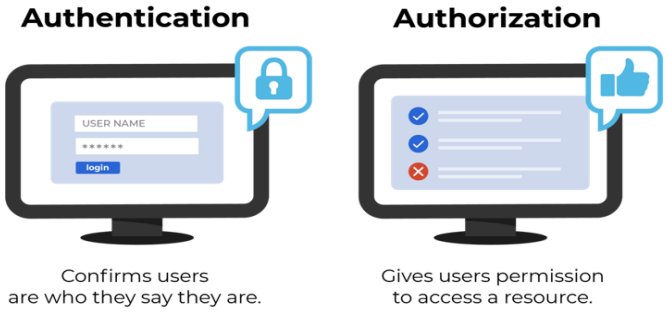
B2B Gateway vs. B2C Gateway	
B2B Gateway	**B2C Gateway**
➢ 3rd party applications ➢ POS Systems ➢ Ecommerce Platforms	➢ Mobile App (frontend) ➢ Web App (frontend)

Authentication vs. Authorization

Authentication is the act of validating that users are who they claim to be. This is the first step in any security process, usually consisting of usernames and passwords. If a user enters the correct data, the system assumes the identity is valid and grants access.

To complete this process in Echo Loyalty, we need to have a username and a password defined in the system. Username and passwords are the most common authentication factors. Once a user provides the correct data, the system assumes the identity is valid and grants access.

Authorization is the process of giving the user permission to access a specific resource or function, such as downloading a particular file or providing administrative access to an application. In secure environments, authorization must always follow authentication. Users should first prove their identity is genuine before the administrators grant them access to the requested resources.

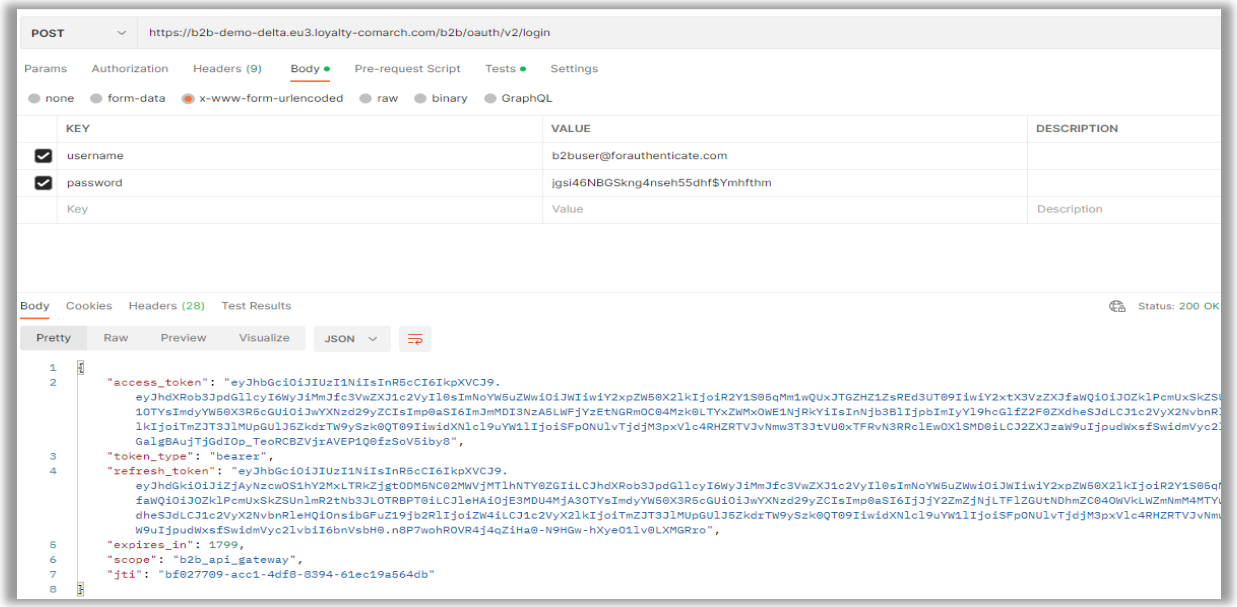


Determines whether users are who they claim to be	Determines what users can and cannot access
Challenges the user to validate credentials	Verifies whether access is allowed through profiles and privileges
Usually done before authorization	Usually done after successful authentication
Generally, transmits info through an ID Token	Generally, transmits info through an Access Token
Generally governed by the OpenID Connect (OIDC) protocol	Generally governed by the Oauth 2.0 framework

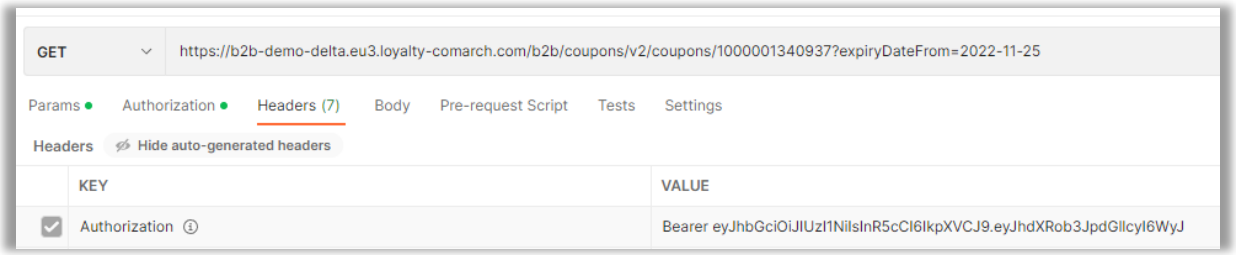
Authentication & Authorization in Echo Loyalty REST API

In Echo Loyalty, to **authenticate** we need to use a log in POST API message. Unlike our other API methods, we do not pass a request body in the JSON format and the x-www-form-urlencoded format of the request body is required for safety issues. In response, we get the access token along with the token time, which is the type of that token that we need for authorization. In Echo Loyalty, it is always a bearer token.

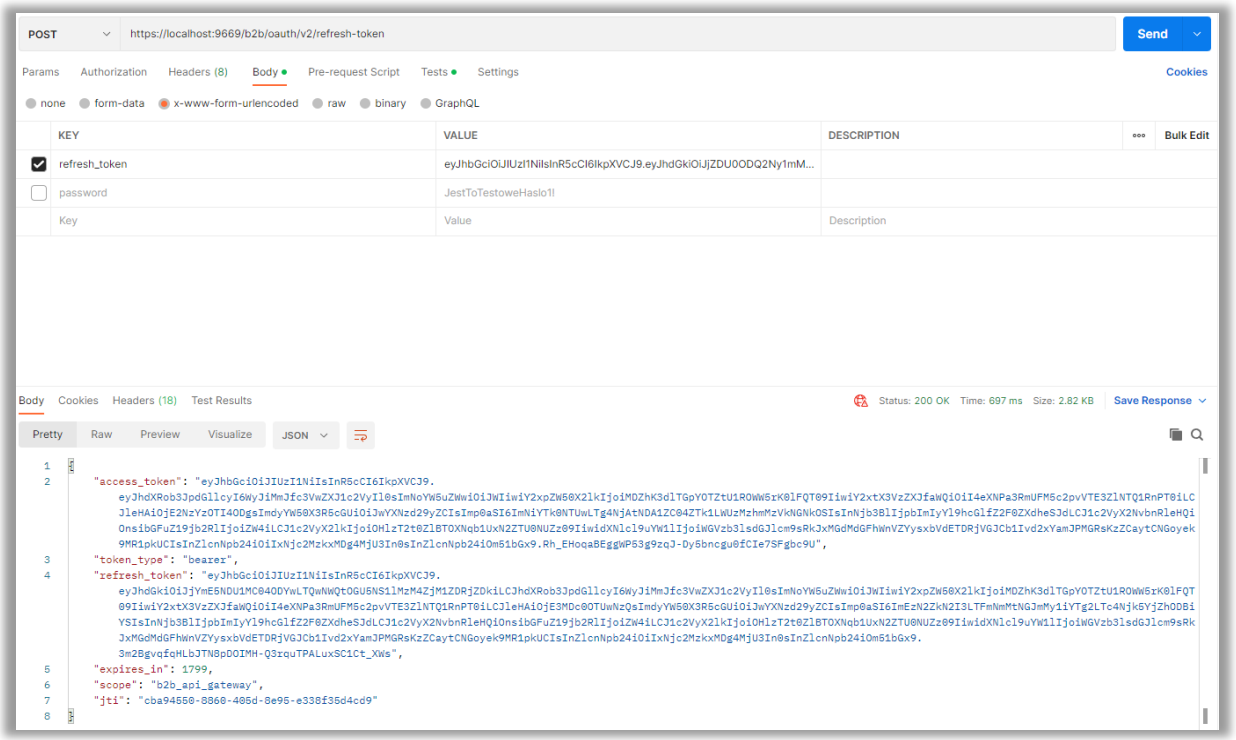
There is a refresh token along with the expiring parameters expressed in seconds which determines the validity period of the access token. Prior to its expiration, we can make use of the refresh token method to sustain access to the API.



At the end, there is also a GTI parameter which is a unique identifier for that particular authentication procedure use for authorizing other REST API methods that require **authorization**. The token type (bearer) needs to be placed in the authorization header along with the access token as shown below.



The **API refreshing** process is not about extending the validity period of the access token, but rather about retrieving a new access token and making use of the refresh token value retrieved during the log in method. To do that, we need to call the refresh token method and pass the property value of the refresh token on the request body as x-www-form-urlencoded like before. In response, we get the same structure as in the log in method; we retrieve a new access method along with the new refresh token for the repeated process of refreshing access to the API in the future if needed.



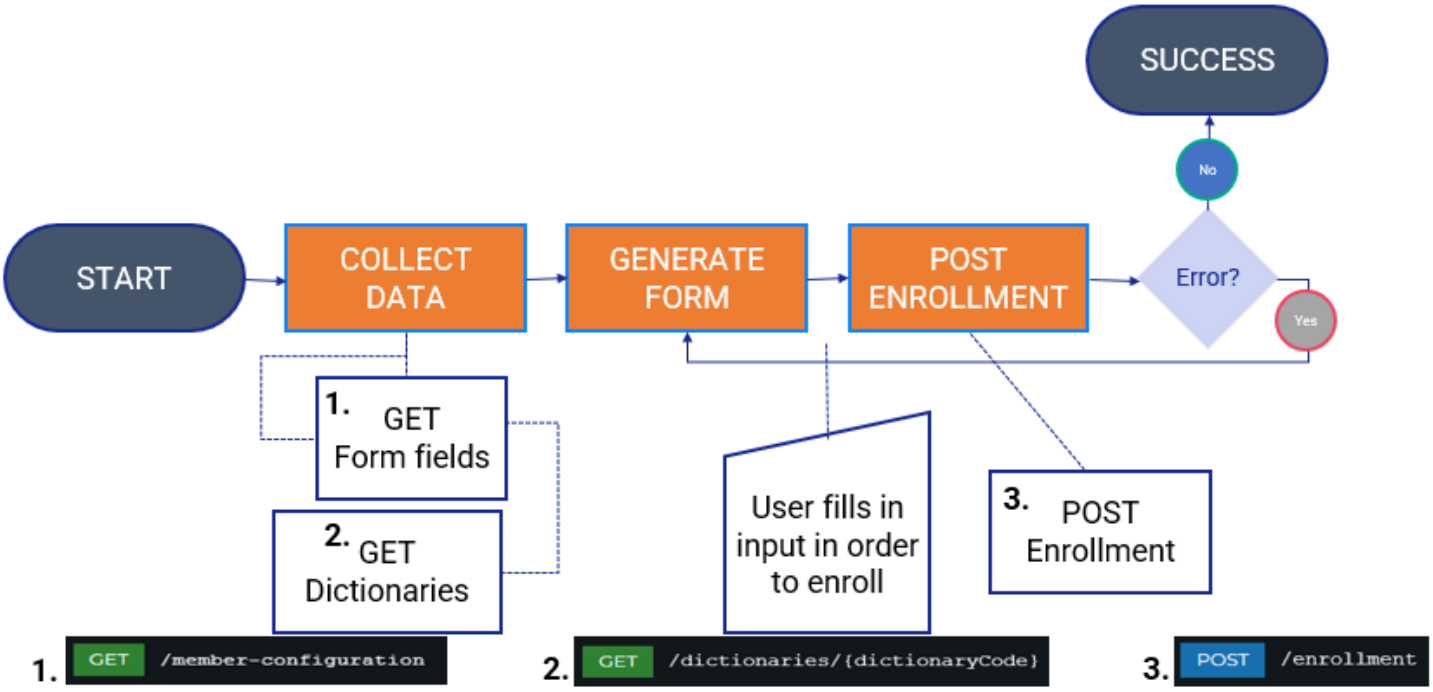
Sample Processes

Enrollment

- Member enrolling through a frontend application
- Using B2C Gateway (this process can also be done via B2B Gateway)

Appropriate fields preconfigured earlier in the marketer panel to be placed in the enrollment form need to be loaded with the use of the member configuration REST API method. Later on, potential members will need to populate these fields. Some fields will require selecting dictionaries values from a dropdown list or checking check boxes. These dictionary values are retrieved by calling the dictionaries REST API method.

Once all these elements are in place, the frontend application will be able to generate the enrollment form for the member to fill in. After that, the frontend places the provided values in the request body of the enrollment method. If the member provides all the values in the proper format, we will get a new member enrolled. If there are any mandatory parameters missing values or in an improper format, the system displays an appropriate error message prompting the enrolling member to introduce the necessary changes.



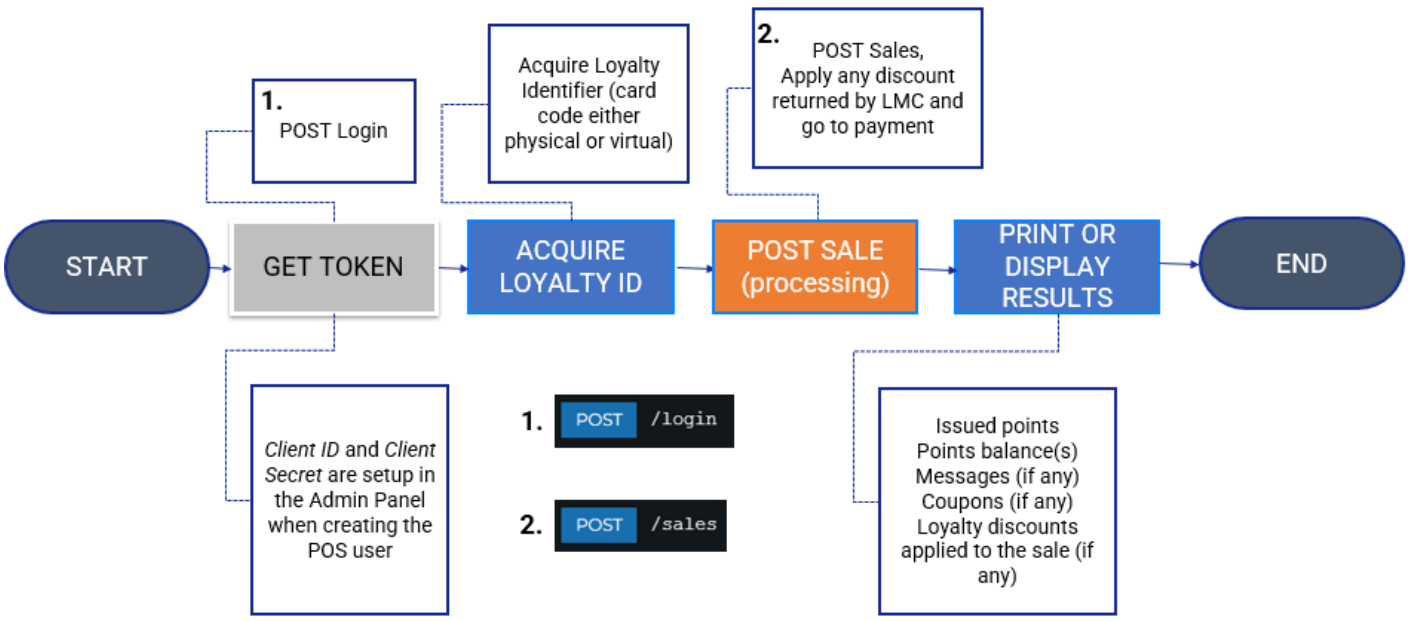
Basic Purchase

- Member performing a transaction
- B2B Gateway

At the very beginning, we need to log in to the gateway to obtain a token that will allow us to obtain the require secure methods, and we also need to know the identifier number of the member who executed the transaction.

Once we have this information, along with other mandatory properties required in the sales request body, we are allowed to call that method. After a successful sales operation, the response body should contain information such as the member's balance, possible discounts, coupons, messages, assigned points and others. If anything goes wrong, the system will display an error message.

At this point, it is worth to mention that you can take advantage of the simulated query parameter option in the sales operation with the simulation query parameter set to TRUE, where not actual operation is performed in the database and only a possible result of that operation is simulated, so nothing is changed in the data base. This feature has been introduced to ease the decision process in order to display loyalty discounts in the POS screen until the transaction is finalized.

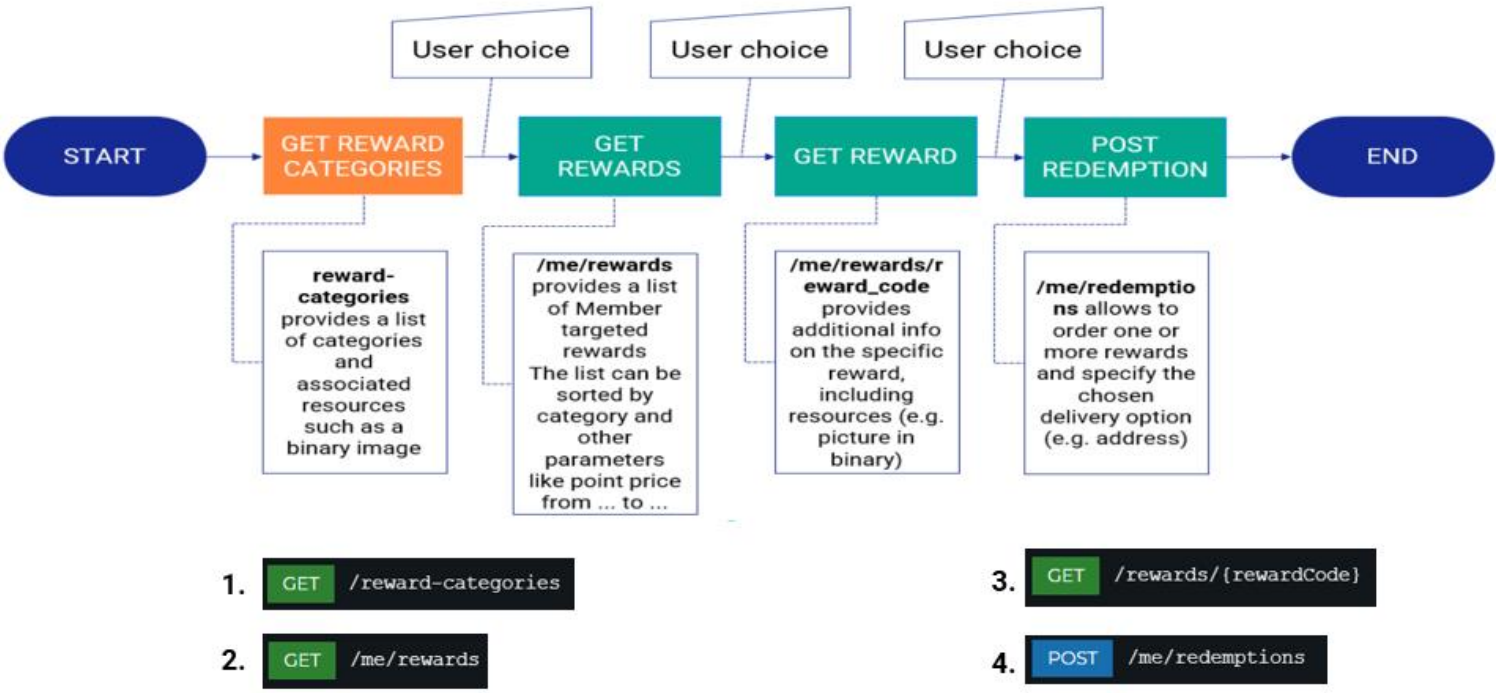


Redemption

- Member access the catalog and order a reward
- B2C Gateway
- Member authentication process omitted

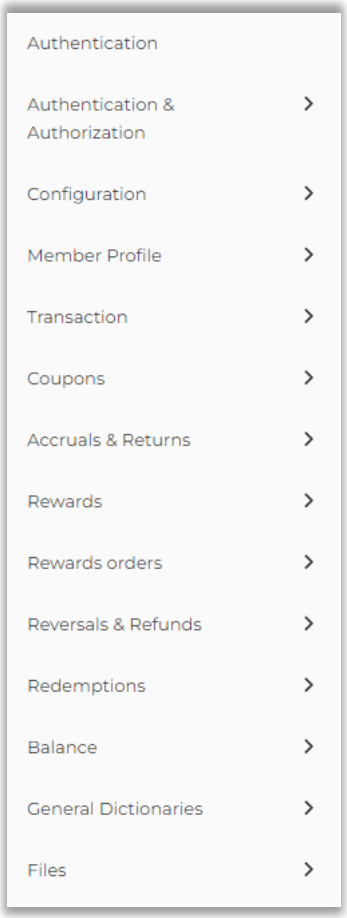
At the beginning, the frontend application calls a secure *rewards-categories* method, which can assign a proper image for each reward category. Then the application calls the *me/rewards* method in order to display the rewards available for a specific customer, whose token was obtained during the authentication stage.

Later, the member may take a specific reward using the *me/rewards* method followed by the code of that particular reward to display details. Following the selection of a particular reward, the customer needs to click the order button which refers the execution on the *me/redemption* method performed by the frontend application.



API Documentation Overview

When we access Echo Loyalty API Documentation, the first thing we see at the top is the title of the documentation (either if it is the B2C or B2C type) along with its version with a button to 'Download' on Json format. Following this, there is a **glossary** listing the definitions of the most commonly used terms during the integration process. Although this covers the main API terminology, there are some custom terms exclusively used in Echo Loyalty documentation.

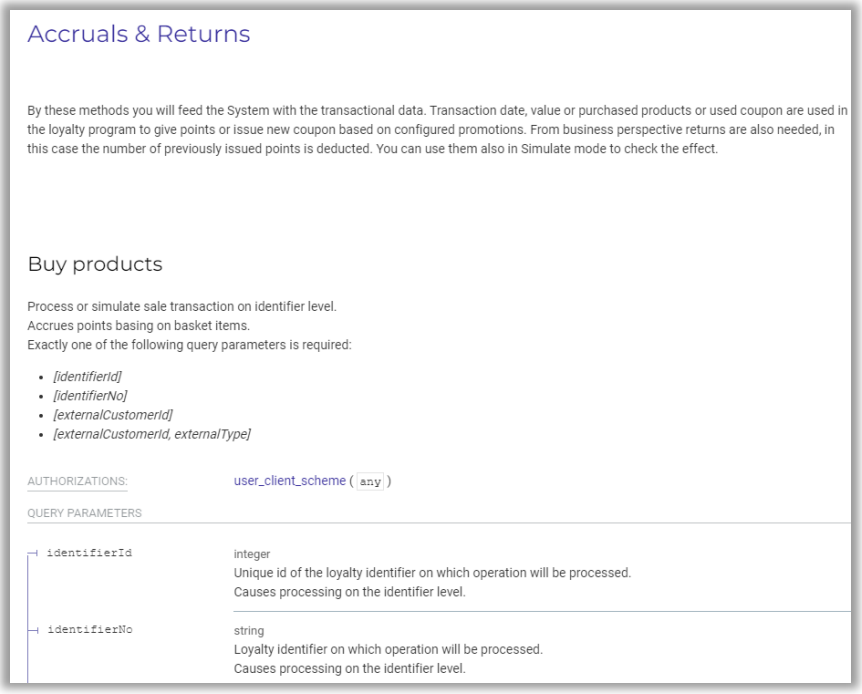


On the left side, there is a **search bar and a menu** that enable accessing different API methods divided into specific categories. When we click in each category, additional information is deployed.

The first API method is **Authentication**, listing authentication schemes offered by the gateway. In this case. It is the user client scheme: the security scheme type (i.e., OAuth2) along with the Token URL, which can be obtained by calling the log in method. In the case of this particular scheme, the scope is defined as any which means that grants access to any secured operations.

Below, we have other particular methods. One of the most important ones is the **Accrual** method or sales operation located on the sales end point. Within the method, we can see the title, a short description, and information regarding the necessary structure.

Additionally, we can see more details of the *query parameters*, including the data type of each parameter and its default value if available, *body parameters,* and *request body schema*. At the end, we have a list of possible responses with different status codes, each with different response headers and response schema.



On the right-hand side, we have a **HTTP method** displayed together with its type, URL end point and the request sample underneath. The code can be copied, expanded or collapse, either entirely by using the 'Expand all' and 'Collapse all' buttons, or partially by clicking on the '+' or '-' buttons.

If we decide to make the exact same request using the 'Copy' option, we need to bear in mind that such request any validation as we haven't defined some data. Finally, below the request samples, there are also response samples for different status codes.