ACQUIBASE api

AQUIBASE api let’s an acquirer or developer register & manage merchants, Create maerchant & process transactions

# Create Merchant fields

Merchant is an Object that contains the following fields:

* Name, Category-code, ext-id, state
* Address info
* Contact info
* Transaction settings (booking\_reference\_prefix (what shows on customer statements), a nested "bluecode" object with the member\_id, and an "instant" object specifying networks ["networks": An array (e.g., ["NIBBS" or “NIP”]) that indicates which instant payment networks the merchant will use. ] and beneficiary details. ["beneficiary\_reference" and "beneficiary\_name": These indicate the merchant’s bank account details where instant payments should be directed. ] )
* Fees (fixed & variable fees for different tiers)
* Billing (Configuration settings like period(Monthly), delay or wether billing is active)

# Request body (Payload)

1. Basic merchant details
   1. name = merchant name
   2. Category\_code = Industry class code (e.g super-market 5411)
   3. ext\_id = external id assigned by acquirer, must be unique for each merchants
   4. "group\_id" = A grouping identifier for organizational purposes. *“Optional”*
2. Contact: Array of names, emails, gender & phone
3. Address
   1. City
   2. Country
   3. ZIP
   4. Line\_1
   5. Line\_2
4. Transaction settings:
   1. booking\_reference\_prefix = A prefix that appears on the payer’s bank statement
5. Identifier
   1. ext\_id = Branch Unique external identifier
   2. merchant\_branch\_id = identifier assigned by the merchant for the branch
   3. merchant\_ext\_id = Reference the external ID of the parent merchants
6. Name & State
   1. name = The branch name
   2. State = The current branch state (e.g “Active”)
7. Time-Stamp
   1. Timestamps like "inserted\_at" or "updated\_at" indicate when the resource was created or modified.

# Global Authentication & Event handlers

1. Authentication block:

A collection is configured to use Basic Authentication with variables for “Username” & “password”. This credentials are applied to every request in the collection

1. Event handlers:

Both the “pre-request” and “test” scripts are empty. This indicates that there are no custom “pre-request” modification or test validation being run automatically; you can add your own scripts here if needed for your integration.

# API calls, URL’s & ENDPOINTS

1. **Create Merchant**
   1. **Http method & URL:**
      * The request is a **POST** to a URL constructed using a variable (e.g., *{{aquibaseNg}}/v2/merchants*).
      * The base URL is defined in the info section; notice there’s a distinction between sandbox (using *.int.bluecode.ng*) and production endpoints (should be *.bluecode.ng* or *.com* as per the checklist).

**Response Examples:**

* The JSON file includes sample responses (with HTTP status 201, meaning "Created") showing what the API returns upon successful creation of a merchant.
* The response body contains a "data" object with details like *access\_id*, *access\_secret\_key*, address, billing info, contact info, and transaction settings.
* There are also HTTP headers such as location that indicate the URL of the newly created merchant resource.

NOTE: We need to store the *access\_id, access\_secret\_key* in our database, Make sure that parameters are sent in the request body (not as query strings) and that every request uses a unique merchant\_tx\_id if applicable.

1. **FETCH MERCHANT DETAILS**

The URL for fetching details is structured as *{{aquibaseNg}}/v2/merchants/{{merchant\_ext\_id}}*. **GET**

1. When you call this endpoint, the API returns the merchant’s data similar to what was returned during creation.
2. The response includes the merchant’s address, contact details, billing, fees, and transaction settings, among other data.
3. The returned JSON follows a similar structure as the creation response, nested under a "data" object.

### ****Create Branch Request****

* **HTTP Method & URL:**
  + **Method:** POST
  + **URL:** The endpoint is constructed as *{{aquibaseNg}}/v2/merchants/{{merchant\_ext\_id}}/branches, where {{merchant\_ext\_id}}* is a variable representing the external ID of the merchant for whom you want to add a branch.
* **Request Body (Raw JSON):**  
  The request payload contains a top-level "branch" object with details such as:
  + **Branch Name:** "name": "Test Outlet 1" or "New Action Branch HQ".
  + **Merchant Branch ID and ext\_id:** Unique identifiers for the branch (e.g., "merchant\_branch\_id": "OT1289879", "ext\_id": "OT1289879").
  + **Address:** Details of the branch’s address (e.g., "city": "Westville", "line\_1": "Dock Street 2", "zip": "098765", "country": "NG").
  + **Contact:** Information for the branch contact, similar to the merchant’s contact details.
  + **Booking Reference Prefix:** A prefix that will appear on bank statements (e.g., "Test Outlet 1").
  + **State:** Typically "ACTIVE".
  + **Virtual Terminal (optional):** Some examples include a "virtual\_terminal" object or URL, which might be used to access a cloud-based terminal for processing payments.

### ****2. Fetch Branch (Merchant Details) Requests****

* **HTTP Method & URL:**
  + A **GET** request is used to fetch details about a specific branch or merchant.
  + The URL will look similar to *{{aquibaseNg}}/v2/merchants/{{merchant\_ext\_id}}* or include a branch path.
* **Response:**
  + The response returns a JSON object under "data" with details about the branch (or merchant), echoing the information provided during creation.

# ****3. Fetch Branch Details****

### ****Request Overview****

* **Method:** GET
* **URL Format:**
  + The URL is built from a base variable (e.g., *{{aquibaseNg}}*) and a path that includes dynamic segments for the merchant’s external ID (*{{merchant\_ext\_id}}*) and the branch’s external ID (*{{branch\_ext\_id}}*).
  + Example URL structure:

*{{aquibaseNg}}/v2/merchants/{{merchant\_ext\_id}}/branches/{{branch\_ext\_id}}*

* **Purpose:**
  + This endpoint is used to retrieve the full details of a specific branch associated with a merchant.

#### ****4. Update Branch Details****

### ****Request Overview****

* **Method:** PUT
* **URL Format:**
  + The URL here is similar to the Fetch request but indicates an update action. It’s built using variables:

*{{aquibaseNg}}/v2/merchants/{{member\_id}}/branches/{{branch\_ext\_id}}*

* + Note: In one example, the URL uses {{member\_id}} instead of {{merchant\_ext\_id}}—this might be a variable naming difference. Ensure consistency in your implementation.

**Enhancement Ideas:**

* **Real-Time Updates:**
  + Consider using AJAX or WebSockets to fetch and update branch details without requiring full page reloads.

**Detailed Reporting:**

* + - Provide additional analytics or reporting on branch performance, such as transaction volumes or sales trends.

### 3. Authentication & Security

**What It Does:**

* The API uses Basic Authentication (with username and password variables) for securing requests.
* All requests and responses are in JSON, encoded in UTF-8, ensuring compatibility and ease of integration.

**How to Use It in Your Project:**

* **Secure API Integration:**
  + Configure your PHP backend to include Basic Auth headers on every API request.
  + Store authentication credentials securely (using environment variables or a secure configuration file).

**Enhancement Ideas:**

* **Additional Security Measures:**
  + Consider wrapping API calls in HTTPS, implementing rate limiting, and possibly adding logging and monitoring for API usage.
  + Evaluate if additional layers (such as token-based authentication for your own backend) are necessary.

### 4. Environment Configuration (Sandbox vs. Production)

**What It Does:**

* The API documentation specifies different base URLs for Sandbox and Production environments.
* Your Postman examples use variables (like {{aquibaseURL}} and {{aquibaseNg}}) to switch between these environments.

**How to Use It in Your Project:**

* **Environment Variables:**
  + Configure your project to use environment variables for the base URL, which allows you to easily switch between testing (sandbox) and live (production) environments.

**Enhancement Ideas:**

* **Dynamic Environment Switching:**
  + Build an admin tool or configuration panel in your backend that lets you switch environments without code changes.
  + Integrate logging to track which environment is active during API calls.

# Updating a Merchant

Updates an existing merchant.

Information on a merchant can be updated at any time by sending a PUT request to /merchant followed by the ext\_id that was assigned to the merchant when it was created.

The parameter format to the update call is identical to the one used during merchant creation so see [Create merchant](https://bluecode-acquirer.readme.io/reference/create-merchant) for details. None of the parameters are required. If a parameter is not sent, the current value for the merchant is maintained.

Almost all information on a merchant can be updated, with the **exception** of the following attributes:

* ext\_id
* access\_id
* inserted\_at
* settlement/iban
* settlement/bic
* settlement/account\_holder
* transaction\_settings/instant/beneficiary\_reference
* transaction\_settings/instant/beneficiary\_name

**Note:** it is the responsibility of the acquirer to make sure data consistency is maintained. Changing the merchant name or organization number to that of a different merchant is obviously not allowed.

Changes in the billing cycle or settlement cycle on a merchant that is already accepting payments can lead to unpredictable results and should not be performed without consulting SPT.

MERCHANT DASHBOARD DESIGN

Based on your requirements, here’s a detailed wireframe and dashboard design plan tailored for merchants (not for developers) that supports branch management and QR code–based transactions.

## ****Merchant Dashboard – Wireframe & Layout****

### ****1. Navigation Sidebar (Left Panel)****

This panel remains fixed and provides quick access to the core modules:

* **Dashboard (Overview)**
* **Branches**
  + View, create, update, and delete branch information.
* **Payments**
  + Request payment
  + Payment history
* **QR Code**
  + Generate and view your merchant QR code (for receiving payments)
* **Profile & Settings**
  + Manage account details, update transaction/settlement settings, etc.
* **Support/Help**

### ****2. Top Navigation Bar****

Placed at the top of every page for quick global access:

* **Merchant Logo/Name**
* **Search Bar:** Allow merchants to quickly search for transactions or branch info.
* **Notifications:** Alert merchants for new payment requests or issues.
* **User Profile Icon:** For accessing account settings or logout.

### ****3. Main Dashboard (Overview)****

When merchants first log in, they see a dashboard with key performance indicators:

* **Summary Widgets:**
  + **Total Branches:** Count of active branches.
  + **Total Transactions:** Today’s, weekly, monthly totals.
  + **Revenue:** A quick summary of the total amount processed.
  + **Pending Payments:** Alerts for any payment requests not yet processed.
* **Recent Activity Feed:**
  + Latest payment requests (success, pending, failed).
  + Recent branch updates or notifications.
* **Charts/Graphs:**
  + **Transaction Trends:** A line or bar chart showing payment volumes over time.
  + **Revenue Trends:** Graphs to track revenue performance.

### ****4. Branch Management Module****

This section allows merchants to manage multiple branches:

* **Branches Table/List:**
  + Columns: Branch Name, Address, Contact Number, Status, Actions (Edit/Delete).
* **Branch Details Form:**
  + A form (accessible via a modal or separate page) where merchants can add/edit branch information.
* **Map Integration (Optional):**
  + Display branch locations on a map for quick geographic reference.

### ****5. Payments Module****

This module is specifically for QR code–based transactions:

* **Payment Request Form:**
  + **Input Fields:** Amount, Customer details (if required), and optional remarks.
  + **Mode Selector:**
    - **Merchant QR Code Mode:** The merchant generates a QR code that the customer scans.
    - **Customer Barcode Mode:** The merchant scans a barcode presented by the customer.
* **Payment History Table:**
  + Columns: Transaction ID, Date, Amount, Status, Branch (if applicable), and actions like view details or resend.
* **Real-Time Payment Status:**
  + An alert or status indicator for recent or pending payment requests.

### ****6. QR Code Generation Section****

A dedicated page or widget where the merchant’s unique QR code is generated and displayed:

* **QR Code Display:**
  + The QR code image (generated using a PHP library or an API) that encodes the merchant’s payment details (or a URL to the payment gateway).
* **Download/Print Options:**
  + Allow merchants to download or print their QR code for in-store use.
* **Toggle for Modes:**
  + If the merchant can switch between “Merchant QR” and “Customer Barcode,” include toggles or radio buttons to select the mode.

### ****7. Settings & Profile****

A section for merchants to manage:

* **Transaction Settings:**
  + Adjust booking reference prefix, default source, etc.
* **Settlement Settings:**
  + Update bank account information, settlement period, delay, etc.
* **API Credentials (if applicable):**
  + View or update API integration details.
* **General Account Info:**
  + Personal information, contact details, etc.

## ****Wireframe Overview****

Imagine the layout divided as follows:

-------------------------------------

| Top Navbar |

-------------------------------------

| Sidebar | Main Content |

| | -------------------------|

| | | Dashboard Overview | |

| | | [Widgets & Charts] | |

| | -------------------------|

| | |

| | [Branch Management] |

| | |

| | [Payments & QR Code] |

-------------------------------------

## ****Additional Considerations****

* **Mobile Responsiveness:**  
  Ensure the dashboard layout is responsive. Use a grid system (like Bootstrap) so that on smaller devices, the sidebar becomes a hamburger menu and widgets stack vertically.
* **User Experience:**
  + Keep the interface clean and intuitive.
  + Use clear labels and tooltips for any technical terms (like "booking reference prefix" or "settlement delay").
  + Incorporate consistent color coding: for example, success alerts in green, warnings in yellow, and errors in red.
* **Integration with QR Code Libraries:**  
  For generating QR codes, you can integrate a PHP library (like [Endroid QR Code](https://github.com/endroid/qr-code)) or use an external API to generate the QR code image.

## ****Summary****

Your merchant dashboard should be designed to help merchants:

* **Monitor their business performance** with clear summaries and trends.
* **Manage multiple branches** easily.
* **Process QR code–based transactions** with dedicated payment and QR code sections.
* **Adjust settings and view their account information** in a dedicated area.

This comprehensive design should cover all the key functionalities your merchants need to manage their operations effectively. Let me know if you'd like further details on any section or additional wireframe sketches!