Fluxbase

Backend Integration Guide

How to connect your backend to Fluxbase using the REST API

Node.js Python Go Java Ruby PHP Rust cURL

v4.0 • Enhanced edition • March 2026

Overview & Correct API Reference

Fluxbase exposes a single SQL execution endpoint that allows any backend application to run queries against a Fluxbase-managed project database. Authentication uses a per-project API Key sent as a Bearer token.

IMPORTANT — Read Before Integrating

The API endpoint is POST /api/execute-sql (NOT /v1/projects/.../query). The request body uses the field name "query" (NOT "sql"). The response rows are at data.result.rows (NOT data). Earlier documentation contained errors on all three points — this edition is the corrected reference.

API Endpoint

HTTP

```
Method:   POST
URL:      https://fluxbase.vercel.app/api/execute-sql
Auth:     Bearer <YOUR_API_KEY>
Content:  application/json
```

Request Body

JSON

```
{
  "projectId": "YOUR_PROJECT_ID",
  "query":     "SELECT * FROM users LIMIT 10;"
}
```

Fields

› "projectId" — Your project's unique ID (found in Dashboard ⁄' Settings). If y
project, this field can be omitted and will be auto-injected.

› "query" — The SQL statement to execute (SELECT, INSERT, UPDATE, DELETE, CREATE TABLE, etc.).

Response Shape

SUCCESS RESPONSE

```
{
  "success": true,
  "result": {
    "rows":    [ { "id": 1, "name": "Alice" }, ... ],
    "columns": [ "id", "name" ],
    "message": null
  },
  "explanation": [],
  "executionInfo": {
    "time": "12ms",
    "rowCount": 1
  }
}
```

Error Response

ERROR RESPONSE

```
{
  "success": false,
  "error": {
    "message": "Project not found",
    "code":    "NOT_FOUND"
  }
}
```

Error Codes

› AUTH_REQUIRED (401) — Missing or invalid API key.
› SCOPE_MISMATCH (403) — API key is scoped to a different project.
› NOT_FOUND (404) — Project does not exist or you lack access.
› RATE_LIMIT_EXCEEDED (429) — 30 requests per 10 seconds per project exceeded.
› BAD_REQUEST (400) — Missing projectId or query field.
› EXECUTION_ERROR (200) — SQL syntax or runtime error (check body.error.message).

Node.js — native fetch (REST)

No external packages required. Uses built-in fetch (Node 18+).

JAVASCRIPT (NODE.JS)

```
const FLUXBASE_URL = 'https://fluxbase.vercel.app/api/execute-sql';
const API_KEY      = process.env.FLUXBASE_API_KEY;
const PROJECT_ID   = process.env.FLUXBASE_PROJECT_ID;

async function runQuery(sql) {
  const res = await fetch(FLUXBASE_URL, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${API_KEY}`
    },
    body: JSON.stringify({
      projectId: PROJECT_ID,  // field name is 'query', NOT 'sql'
      query:     sql
    })
  });

  const json = await res.json();
  if (!json.success) throw new Error(json.error.message);

  // Rows are at json.result.rows — NOT json.data
  return json.result.rows;
}

runQuery('SELECT * FROM users LIMIT 5').then(rows => {
  console.log('Rows:', rows);
}).catch(console.error);
```

Python — requests

Install: pip install requests

PYTHON

```python
import os, requests

FLUXBASE_URL = 'https://fluxbase.vercel.app/api/execute-sql'
API_KEY      = os.getenv('FLUXBASE_API_KEY')
PROJECT_ID   = os.getenv('FLUXBASE_PROJECT_ID')

def run_query(sql: str):
    headers = {
        'Authorization': f'Bearer {API_KEY}',
        'Content-Type':  'application/json'
    }
    # Body uses field 'query' (NOT 'sql')
    payload = {'projectId': PROJECT_ID, 'query': sql}
    resp = requests.post(FLUXBASE_URL, json=payload, headers=headers)
    data = resp.json()
    if not data.get('success'):
        raise Exception(data['error']['message'])
    # Access via data['result']['rows']: (NOT data['data'])
    return data['result']['rows']

rows = run_query('SELECT * FROM users')
print(rows)
```

Go — net/http (REST)

Uses Go's standard library only.

GO

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "net/http"
    "os"
)

func runQuery(sql string) ([]map[string]interface{}, error) {
    body, _ := json.Marshal(map[string]string{
        "projectId": os.Getenv("FLUXBASE_PROJECT_ID"),
        "query":     sql,  // field is "query", not "sql"
    })
    req, _ := http.NewRequest("POST", "https://fluxbase.vercel.app/api/execute-sql",
        bytes.NewBuffer(body))
    req.Header.Set("Authorization", "Bearer "+os.Getenv("FLUXBASE_API_KEY"))
    req.Header.Set("Content-Type", "application/json")

    resp, err := http.DefaultClient.Do(req)
    if err != nil { return nil, err }
    defer resp.Body.Close()

    var result struct {
        Success bool `json:"success"`
        Result  struct {
            Rows []map[string]interface{} `json:"rows"`
        } `json:"result"` // nested under "result", not "data"
        Error struct{ Message string `json:"message"` } `json:"error"`
    }
    json.NewDecoder(resp.Body).Decode(&result)
    if !result.Success { return nil, fmt.Errorf(result.Error.Message) }
    return result.Result.Rows, nil
}
```

Java — HttpURLConnection (REST)

Uses Java 11+ standard library. No Maven dependency required.

JAVA

```java
import java.net.URI; import java.net.http.*;
import java.util.Map; import com.fasterxml.jackson.databind.ObjectMapper;

HttpClient client = HttpClient.newHttpClient();
ObjectMapper mapper = new ObjectMapper();

// Body: projectId + query (NOT "sql")
String body = mapper.writeValueAsString(Map.of(
    "projectId", System.getenv("FLUXBASE_PROJECT_ID"),
    "query",     "SELECT * FROM users"
));

HttpRequest req = HttpRequest.newBuilder()
    .uri(URI.create("https://fluxbase.vercel.app/api/execute-sql"))
    .POST(HttpRequest.BodyPublishers.ofString(body))
    .header("Content-Type", "application/json")
    .header("Authorization", "Bearer " + System.getenv("FLUXBASE_API_KEY"))
    .build();

HttpResponse<String> resp = client.send(req, HttpResponse.BodyHandlers.ofString());
Map<?,?> json = mapper.readValue(resp.body(), Map.class);
// Rows: ((Map)json.get("result")).get("rows")
System.out.println(((Map<?,?>)json.get("result")).get("rows"));
```

Ruby — net/http (REST)

Uses Ruby's standard library.

```ruby
require 'uri', 'net/http', 'json'

FLUXBASE_URL = URI('https://fluxbase.vercel.app/api/execute-sql')

def run_query(sql)
  http = Net::HTTP.new(FLUXBASE_URL.host, FLUXBASE_URL.port)
  http.use_ssl = true
  req = Net::HTTP::Post.new(FLUXBASE_URL)
  req['Authorization'] = "Bearer #{ENV['FLUXBASE_API_KEY']}"
  req['Content-Type']  = 'application/json'
  # Body field is 'query', NOT 'sql'
  req.body = JSON.dump({ projectId: ENV['FLUXBASE_PROJECT_ID'], query: sql })
  body = JSON.parse(http.request(req).read_body)
  raise body['error']['message'] unless body['success']
  body['result']['rows']  # Access via result.rows, NOT data
end

p run_query('SELECT * FROM users')
```

PHP — cURL (REST)

Uses cURL, which ships with PHP by default.

PHP

```php
<?php
$url    = 'https://fluxbase.vercel.app/api/execute-sql';
$apiKey = getenv('FLUXBASE_API_KEY');
$projId = getenv('FLUXBASE_PROJECT_ID');

// Body field is 'query' (NOT 'sql')
$payload = json_encode(['projectId' => $projId, 'query' => 'SELECT * FROM users']);

$ch = curl_init($url);
curl_setopt_array($ch, [
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_POST           => true,
    CURLOPT_POSTFIELDS     => $payload,
    CURLOPT_HTTPHEADER     => [
        'Content-Type: application/json',
        'Authorization: Bearer ' . $apiKey,
    ],
]);
$resp = json_decode(curl_exec($ch), true);
curl_close($ch);

if (!$resp['success']) throw new Exception($resp['error']['message']);
$rows = $resp['result']['rows']; // NOT $resp['data']
print_r($rows);
```

Rust — reqwest (REST)

Add to Cargo.toml: reqwest = { version = "0.12", features = ["json"] } tokio = { version = "1", features = ["full"] } serde_json = "1"

RUST

```rust
use reqwest::header::{AUTHORIZATION, CONTENT_TYPE};
use serde_json::{json, Value};

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let api_key    = std::env::var("FLUXBASE_API_KEY")?;
    let project_id = std::env::var("FLUXBASE_PROJECT_ID")?;

    let client = reqwest::Client::new();
    // Body uses "query" field, NOT "sql"
    let body = json!({ "projectId": project_id, "query": "SELECT * FROM users" });

    let resp: Value = client
        .post("https://fluxbase.vercel.app/api/execute-sql")
        .header(AUTHORIZATION, format!("Bearer {}", api_key))
        .header(CONTENT_TYPE, "application/json")
        .json(&body)
        .send().await?
        .json().await?;

    if !resp["success"].as_bool().unwrap_or(false) {
        eprintln!("Error: {}", resp["error"]["message"]);
    } else {
        // Rows are at resp["result"]["rows"] NOT resp["data"]
        println!("{:#?}", resp["result"]["rows"]);
    }
    Ok(())
}
```

cURL — Shell / Terminal

Directly test your integration from any terminal.

BASH

```bash
curl -X POST "https://fluxbase.vercel.app/api/execute-sql" \
  -H "Authorization: Bearer $FLUXBASE_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "projectId": "YOUR_PROJECT_ID",
    "query":     "SELECT * FROM users LIMIT 5"
  }'

# Expected response structure:
# { "success": true, "result": { "rows": [...], "columns": [...] } }
```

Common SQL Examples

Insert a Row

JSON

```
// Body to send:
{
  "projectId": "YOUR_PROJECT_ID",
  "query": "INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com')"
}

// Response: result.message will contain row count info
```

Update a Row

JSON

```
{
  "projectId": "YOUR_PROJECT_ID",
  "query": "UPDATE users SET name = 'Bob' WHERE id = 42"
}
```

Delete a Row

JSON

```
{
  "projectId": "YOUR_PROJECT_ID",
  "query": "DELETE FROM users WHERE id = 42"
}
```

Join Tables

JSON

```
{
  "projectId": "YOUR_PROJECT_ID",
  "query": "SELECT u.name, o.total FROM users u JOIN orders o ON u.id = o.user_id"
}
```

Create a Table (DDL)

```json
{
  "projectId": "YOUR_PROJECT_ID",
  "query": "CREATE TABLE products (id SERIAL PRIMARY KEY, name TEXT, price NUMERIC)"
}
```

DDL statements (CREATE, ALTER, DROP) return { success: true, result: { message: "..." } } with no rows.

Rate Limiting

- Limit: 30 requests per 10 seconds per project per user.
- When exceeded the API returns HTTP 429 with code RATE_LIMIT_EXCEEDED.
- SELECT queries are cached server-side for 15 seconds — duplicate identical reads are free.

Getting Your Credentials

- Project ID — Dashboard → Your Project → Settings → Project ID
- API Key — Dashboard → Your Project → Settings → API Keys → Create Key
- Scope your API key to a single project for maximum security.

Support

- Web: https://fluxbase.vercel.app/docs
- Email: sumithsumith4567890@gmail.com

Webhooks

Webhooks let Fluxbase automatically notify your application whenever data changes in a table — a row is inserted, updated, or deleted. Fluxbase fires an outbound HTTP POST to your app's URL within ~1–2 seconds of the event, no persistent connection required.

Architecture — Outbound HTTP (No Extra Server Needed)

Direction: Fluxbase (on Vercel) %%% POST %%%!' Your App

Webhooks are OUTBOUND from Fluxbase. You do NOT need Render or any long-running server for webhooks. Fluxbase calls YOUR endpoint. Any publicly accessible URL works — Vercel, Railway, Render, AWS Lambda, Cloudflare Workers, etc.

NOTE: SSE/Realtime is different and DOES require a persistent server. Webhooks and Realtime are separate features with different infrastructure needs.

Webhook Payload

Every webhook event delivers this JSON body to your endpoint:

WEBHOOK PAYLOAD (JSON)

```
{
  "event_type": "row.inserted",
  "table_id":   "orders",
  "timestamp":  "2026-03-27T19:00:00.000Z",
  "data": {
    "new": { "id": "abc123", "amount": 500, "status": "pending" },
    "old": null
  }
}
```

Payload Fields

› "event_type" — One of: row.inserted | row.updated | row.deleted
› "table_id" — The table name where the event occurred.
› "timestamp" — ISO-8601 UTC timestamp of the event.
› "data.new" — New row values (present on row.inserted and row.updated).
› "data.old" — Previous row values (present on row.updated and row.deleted).

Step 1 — Create a Receiver Endpoint in Your App

Add any HTTP POST route to your application. Fluxbase will call this URL every time a relevant event fires.

Next.js (App Router)

TYPESCRIPT (NEXT.JS)

```
// app/api/fluxbase-webhook/route.ts
import { NextRequest, NextResponse } from 'next/server';

export async function POST(req: NextRequest) {
  const { event_type, table_id, data } = await req.json();

  if (table_id === 'matches' && event_type === 'row.inserted') {
    await sendMatchNotification(data.new.user_b, data.new.id);
  }

  if (table_id === 'messages' && event_type === 'row.inserted') {
    await broadcastToUser(data.new.recipient_id, data.new);
  }

  return NextResponse.json({ ok: true });
}
```

Node.js / Express

JAVASCRIPT (EXPRESS)

```
const express = require('express');
const app = express();
app.use(express.json());

app.post('/fluxbase-webhook', (req, res) => {
  const { event_type, table_id, data } = req.body;
  if (event_type === 'row.inserted') console.log('New row in', table_id, data.new);
  res.status(200).send("ok");
});
app.listen(3000);
```

Step 2 — Register the Webhook in Fluxbase

Option A — Via the Dashboard: Settings ⁉ Webhooks ⁉ Add Webhook

› Name: A descriptive label (e.g. "New Order Listener")
› URL: The fully public URL (e.g. https://myapp.vercel.app/api/fluxbase-webhook)
› Event: row.inserted | row.updated | row.deleted | * for all events
› Table: A specific table name, or * to listen to all tables

Option B — Via the REST API:

HTTP

```
POST /api/webhooks
Authorization: Bearer YOUR_API_KEY
Content-Type: application/json

{
  "projectId": "YOUR_PROJECT_ID",
  "name":      "New Order Listener",
  "url":       "https://myapp.vercel.app/api/fluxbase-webhook",
  "event":     "row.inserted",
  "table_id":  "orders",
  "is_active": true
}
```

Step 3 — Test Locally with ngrok

Fluxbase can only POST to a public URL — localhost will not work. Use ngrok to expose your local server during development:

BASH

```
# Install ngrok once
npm install -g ngrok

# Start your local dev server
npm run dev   # running on port 3000

# In a 2nd terminal, open a tunnel
ngrok http 3000

# ngrok gives you a URL like:
#   https://a1b2-103-123-456.ngrok-free.app

# Register that URL in Fluxbase:
#   https://a1b2-103-123-456.ngrok-free.app/api/fluxbase-webhook
```

Quick Test Without Code

Go to https://webhook.site, copy the unique URL it gives you, register it as your Fluxbase webhook, then insert a row via the Table Editor. The full JSON payload will appear on webhook.site instantly.

Security Tip

Register a webhook Secret in Fluxbase. Your receiver endpoint can then verify the X-Fluxbase-Signature header using HMAC-SHA256 to confirm requests are genuinely from Fluxbase.

Storage

Fluxbase Storage allows you to upload, manage and serve files (images, PDFs, videos, CSVs) backed by AWS S3. All files are private by default — you use short-lived presigned URLs to serve them securely.

Storage Workflow

```
WORKFLOW

1. Create a Bucket  —  a logical container for your files
2.  Upload  a  File      !'    POST  /api/storage/upload    (mu
3. Save s3_key      —  store the returned s3_key in your own database table
4.  Serve  the  file     !'    GET  /api/storage/url?s3Key=..
5.  Render               !'    <img  src={url}  />  or  trigger
```

Bucket Management

List Buckets — GET /api/storage/buckets

```
HTTP

GET  /api/storage/buckets?projectId=YOUR_PROJECT_ID
Authorization: Bearer YOUR_API_KEY

// Response
{ "success": true, "buckets": [{ "id": "...", "name": "profile-pictures", "is_public": false
}] }
```

Create a Bucket — POST /api/storage/buckets

```
HTTP

POST /api/storage/buckets
Authorization: Bearer YOUR_API_KEY
Content-Type: application/json

{ "projectId": "YOUR_PROJECT_ID", "name": "profile-pictures", "isPublic": false }

// Name rules: lowercase, alphanumeric + hyphens/underscores, 1-63 characters
// Response: { "success": true, "bucket": { "id": "...", "name": "..." } }
```

Uploading a File — POST /api/storage/upload

```http
POST /api/storage/upload
Authorization: Bearer YOUR_API_KEY
Content-Type: multipart/form-data

Form fields:
  file      — the File object (from <input type="file">)
  bucketId  — destination bucket ID or Bucket Name (e.g. "photos")
  projectId — your project ID

// Response
{
  "success": true,
  "file": {
    "id": "...", "name": "avatar.jpg",
    "s3_key": "project_xxx/buckets/yyy/1711000000_avatar.jpg",
    "size": 204800, "mime_type": "image/jpeg"
  }
}
```

JavaScript Example

```javascript
const form = new FormData();
form.append('file', document.getElementById('file-input').files[0]);
form.append('bucketId', 'photos'); // Supports ID or Name
form.append('projectId', 'YOUR_PROJECT_ID');

const res = await fetch('https://your-fluxbase.app/api/storage/upload', {
  method: 'POST',
  headers: { 'Authorization': 'Bearer YOUR_API_KEY' },
  body: form
});
const { file } = await res.json();
// Store file.s3_key in your DB, then use it later to get a download URL
```

Getting a Download URL — GET /api/storage/url

All files are private. Call this endpoint to get a 15-minute presigned URL to serve a file:

```http
GET /api/storage/url?s3Key=YOUR_S3_KEY&projectId=YOUR_PROJECT_ID
Authorization: Bearer YOUR_API_KEY

// Response
{ "success": true, "url": "https://s3.amazonaws.com/...", "expiresIn": 900 }
```

List Files — GET /api/storage/files

```http
GET /api/storage/files?bucketId=YOUR_BUCKET_ID&projectId=YOUR_PROJECT_ID
Authorization: Bearer YOUR_API_KEY

// Response
{ "success": true, "files": [{ "id", "name", "s3_key", "size", "mime_type", "created_at" }] }
```

Delete a File — DELETE /api/storage/files

```http
DELETE /api/storage/files
Authorization: Bearer YOUR_API_KEY
Content-Type: application/json

{ "fileId": "...", "s3Key": "...", "projectId": "YOUR_PROJECT_ID" }

// Response: { "success": true }
```

Plan Limits & Supported File Types

› Free: up to 50 MB per file | Pro: 500 MB per file | Max: 2 GB per file
› Images: jpeg, png, gif, webp, svg
› Documents: pdf, txt, csv, json
› Video: mp4, webm | Audio: mp3, wav | Archives: zip

Real-time (SSE) — @fluxbaseteam/fluxbase

Fluxbase provides native Server-Sent Events (SSE) for real-time database change notifications. The official SDK wraps the SSE connection with automatic reconnection, exponential backoff, online/offline detection, and Node.js compatibility.

Architecture — Two URLs Required

Fluxbase requires two separate URLs:
url — Vercel deployment: handles all SQL/REST queries
realtimeUrl — Render sidecar: handles persistent SSE connections

The Render sidecar is required because Vercel serverless functions cannot hold persistent connections.
Pass both URLs to createClient().

Installation

BASH

```
npm install @fluxbaseteam/fluxbase
```

Initialize with Both URLs

TYPESCRIPT

```
import { createClient } from '@fluxbaseteam/fluxbase';

const flux = createClient(
  'https://your-app.vercel.app',
  'your-project-id',
  'fl_your-api-key',
  {
    realtimeUrl: 'https://fluxbase-realtime.onrender
    debug: true,     // logs all events to console
    timeout: 8000,
    retries: 3,
  }
);
```

Subscribe to Live Events

TYPESCRIPT

```
const channel = flux.channel('chat', 'messages')

  .on('row.inserted', (payload) => {
    const newRow = payload.data?.new;
    console.log('New row:', newRow);
  })
  .on('row.updated', (p) => console.log('Updated:', p.data?.new))
  .on('row.deleted', (p) => console.log('Deleted:', p.data?.old))
  .on('*', (p) => console.log('Any event:', p.event_type))

  // Lifecycle hooks
  .onConnect(() => setStatus('connected'))
  .onDisconnect(() => setStatus('disconnected'))
  .onReconnect((attempt, delay) => {
    console.log('Retry #' + attempt + ' in ' + delay + 'ms');
  })

  .subscribe();
```

Pause, Resume and Unsubscribe

TYPESCRIPT

```
channel.pause();       // stop receiving (keeps subscription registered)
channel.resume();      // reconnect after pause
channel.unsubscribe(); // permanently close and clean up
console.log(channel.state); // 'connected' | 'connecting' | 'disconnected' | 'paused'
```

React Hook Pattern

TYPESCRIPT (REACT)

```
import { useEffect } from 'react';
import { createClient } from '@fluxbaseteam/fluxbase';

const flux = createClient(
  process.env.NEXT_PUBLIC_FLUXBASE_URL,
  process.env.NEXT_PUBLIC_FLUXBASE_PROJECT_ID,
  process.env.NEXT_PUBLIC_FLUXBASE_API_KEY,
  { realtimeUrl: process.env.NEXT_PUBLIC_FLUXBASE_REALTIME_URL }
);

function ChatRoom() {
  useEffect(() => {
    const ch = flux.channel('chat', 'messages')
      .on('row.inserted', (p) => setMessages(m => [...m, p.data?.new]))
      .subscribe();
    return () => ch.unsubscribe(); // cleanup on unmount
  }, []);
}
```

Required Environment Variables

.ENV.LOCAL

```
# .env.local
NEXT_PUBLIC_FLUXBASE_URL=https://your-app.vercel.app
NEXT_PUBLIC_FLUXBASE_PROJECT_ID=your-project-id
NEXT_PUBLIC_FLUXBASE_API_KEY=fl_your-api-key
NEXT_PUBLIC_FLUXBASE_REALTIME_URL=https://fluxbase-realtime.onrender.com
```

Raw EventSource (Without SDK)

For non-JS environments or advanced use only — the SDK handles reconnection automatically.

JAVASCRIPT (RAW SSE)

```
const url = new URL('https://fluxbase-realtime.onrender.com/api/realtime/subscribe');
url.searchParams.set('projectId', 'YOUR_PROJECT_ID');
url.searchParams.set('apiKey', 'fl_YOUR_API_KEY');

const source = new EventSource(url.toString());
source.onopen = () => console.log('SSE connected');
source.onmessage = (event) => {
  const payload = JSON.parse(event.data);
  if (payload.type === 'connected') return; // ignore heartbeat
  console.log(payload.event_type, payload.data?.new);
};
source.onerror = () => console.warn("Connection lost — browser will auto-retry");
```

SDK Advantage

The @fluxbaseteam/fluxbase SDK adds: exponential backoff reconnect (1s, 2s, 4s, ma
auto-pause/resume on browser offline/online events, Node.js compatibility via EventSource
ponyfill, and typed error codes. Use the SDK in production apps.

Structured Error Codes

Fluxbase APIs and the @fluxbaseteam/fluxbase SDK return standardized error objects so your app can handle failures programmatically. Import ERROR_CODES for type-safe matching.

SDK Error Handling

TYPESCRIPT

```
import { createClient, ERROR_CODES } from '@fluxbaseteam/fluxbase';

const flux = createClient(url, projectId, apiKey, { realtimeUrl });

// Global auth error handler
flux.onAuthError((err) => {
  if (err.code === ERROR_CODES.UNAUTHORIZED) router.push('/login');
});

// Per-query error handling
const { data, error, success } = await flux.from('users').select('*');
if (!success) {
  switch (error.code) {
    case ERROR_CODES.TIMEOUT:
      showToast('Request timed out — check your connection.');
      break;
    case ERROR_CODES.CORS_ERROR:
      console.error('CORS: Add Authorization to Access-Control-Allow-Headers');
      break;
    case ERROR_CODES.RATE_LIMIT_EXCEEDED:
      showToast('Too many requests — slow down.');
      break;
    default:
      console.error(error.message, error.hint);
  }
}
```

All Error Codes

› AUTH_REQUIRED (401) — Missing or invalid API key.
› UNAUTHORIZED (401) — API key is expired or revoked.
› SCOPE_MISMATCH (403) — API key scoped to a different project.
› TOKEN_EXPIRED — Session token has expired.
› BAD_REQUEST (400) — Missing projectId or query field.
› PROJECT_NOT_FOUND (404) — Project does not exist.
› TABLE_NOT_FOUND (404) — Table referenced in query does not exist.
› SQL_EXECUTION_ERROR — SQL syntax or runtime error. Check error.message.
› RATE_LIMIT_EXCEEDED (429)— 30 requests / 10 seconds per project exceeded.
› NETWORK_ERROR — Could not reach the server (connection issue).
› TIMEOUT — Request exceeded the configured timeout ms.
› CORS_ERROR — Cross-origin request blocked. Check CORS headers.

› ABORTED — Request cancelled via AbortController.abort().
› REALTIME_CONNECTION_FAILED — SSE connection could not be established.
› INTERNAL_ERROR (500) — Server-side error. Check Render/Vercel logs.