Business Partner Platform

Integration Guide

Version 1.0

Last Updated: June 2026

Base URL: https://giftcardshop.co

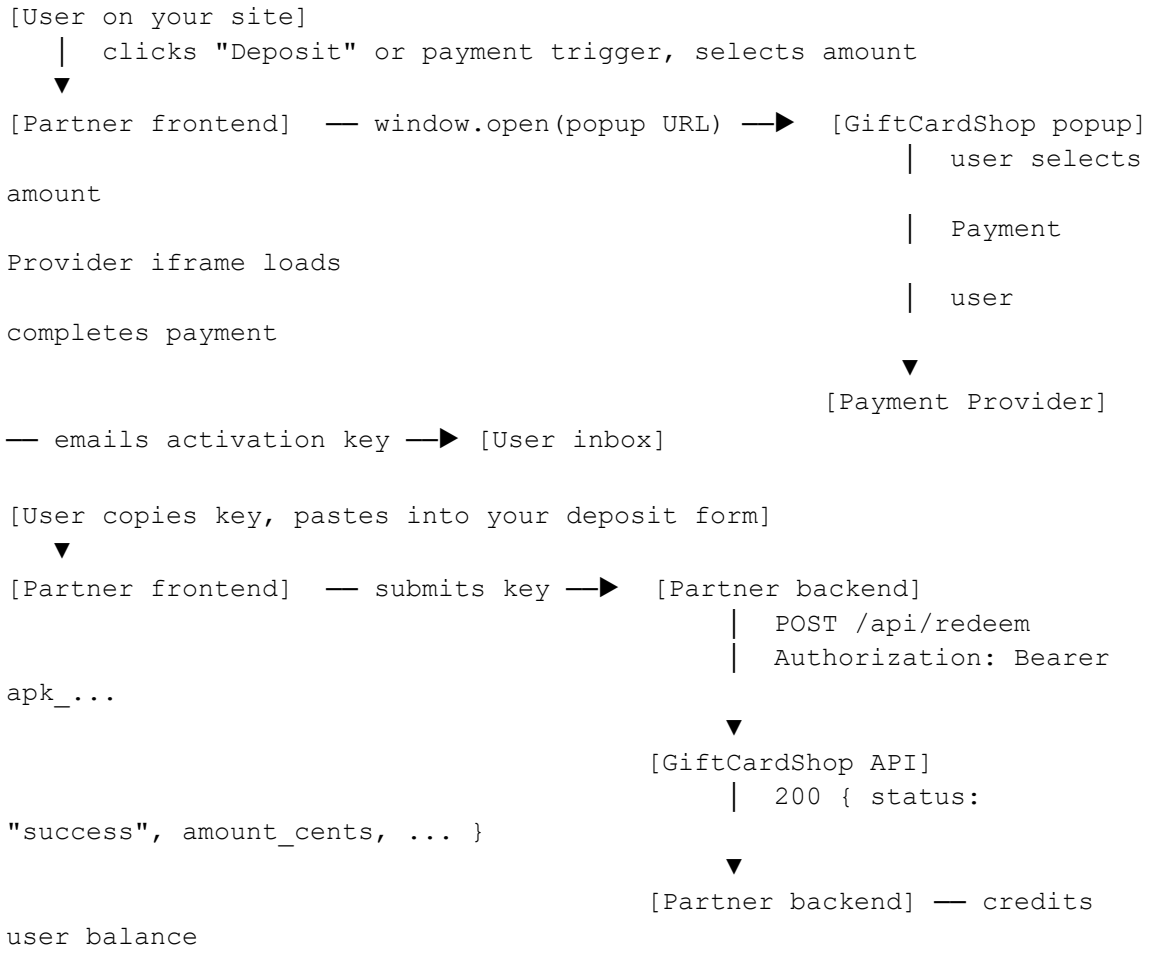
Overview

This document covers everything a Business Partner needs to integrate the balance-deposit flow end-to-end. The platform allows your users to purchase balance for your site in the form of a gift card code, which they then redeem on your platform. There are two integration touchpoints:

#	Touchpoint	Who Calls It	When
1	Popup Checkout	Partner frontend (browser)	User clicks "Deposit" (or any button that triggers the payment journey) → selects amount → pays
2	Redeem API	Partner backend (server-to-server)	User enters their activation code into your platform

Complete Flow at a Glance

The deposit flow involves three parties: your platform (frontend and backend), the popup checkout hosted by GiftCardShop, and the Payment Provider (payment processing).



1. Getting Started

1.1 Your Business Partner Account

Once your Business Partner account is approved you will receive:

- Login credentials for the **Business Partner Portal**
- Your **storefront slug** — a short URL-safe identifier unique to your account (e.g. `your-store`). This slug is used in the popup URL and never changes.

You can find your slug under **Profile → Account Details** in the portal.

1.2 Generating Your API Key

The API key is required for the Redeem API (Step 2 of the integration). Generate it once and store it securely.

- Log in to the Business Partner Portal.
- Go to **Profile → API Keys**.
- Click **Generate Key**.
- Copy the key immediately — **it is shown exactly once**. It looks like:

```
apk_Xk9mR3vTqZpL2wN8cYdA...  (64 characters total)
```

Note: *Store this in an environment variable or secret manager. Never commit it to source code, log it, or expose it in frontend JavaScript.*

If you lose the key, you must regenerate it (the old key is immediately invalidated).

1.3 Products & Offers

You do not need to manage products or offers yourself. The GiftCardShop platform team creates and manages all products, pricing, and stock on your behalf. Once an offer is approved and goes live, it appears in your popup automatically. You will receive an email confirmation when new offers become available.

2. Popup Checkout Integration

The popup is a self-contained page hosted by GiftCardShop. Your frontend opens it as a browser popup window. No iframes, no SDK to install — just `window.open`.

2.1 URL Structure

```
GET https://giftcardshop.co/popup/checkout/{partnerSlug}
```

Replace `{partnerSlug}` with your slug (e.g. `your-store`).

Optional query parameters let you preselect a value so the user skips the catalog and goes straight to checkout:

Parameter	Type	Description
`value`	float (EUR)	EUR amount the user chose on your side. Opens the matching offer directly.
`product_id`	string	Payment Provider product ID. Matches the first live offer for that product.
`offer_id`	string	Exact offer ID. Opens that offer directly. (Most specific.)

If none of the parameters match a live offer, the popup falls back to the full catalog grid. If no parameters are provided at all, the catalog grid is shown.

2.2 Opening the Popup

Use `window.open` with `popup=yes` so the browser opens a real popup window (not a tab):

```
function openDepositPopup(partnerSlug, valueEur) {
    const base = 'https://giftcardshop.co/popup/checkout/'
               + encodeURIComponent(partnerSlug);
    const url  = valueEur ? base + '?value=' + valueEur : base;

    const width  = 500;
    const height = 800;
    const left   = window.screenLeft + (window.innerWidth - width) / 2;
    const top    = window.screenTop + (window.innerHeight - height) /
2;

    const features = [
        'popup=yes',
        'width=' + width, 'height=' + height,
        'left=' + Math.round(left), 'top=' + Math.round(top),
        'resizable=yes', 'scrollbars=yes',
    ].join(',');

    const popup = window.open(url, 'partnerCheckoutPopup', features);
    if (!popup) {
        alert('Please allow popups for this site.');
    }
    return popup;
}

// Example: open for "your-store" with €25 preselected
openDepositPopup('your-store', 25);
```

Recommended popup dimensions: 500 × 800 px. The popup is responsive but designed for this size.

2.3 Behavior Inside the Popup

Catalog mode (no preselection, or preselection did not match):

- A grid of all live offers is shown, ordered by price.
- The user taps the denomination they want.
- The Payment Provider checkout iframe opens full-screen inside the same popup.

Preselect mode (a matching offer was found):

- The popup skips the catalog and opens the Payment Provider checkout iframe immediately.
- If no live offers exist for your account, a friendly "no offers available" message is displayed.

2.4 Popup Lifecycle

- The popup is self-contained. Your page does not need to listen for any postMessage events (no JS handshake required).
- After payment, the Payment Provider emails the activation key directly to the user. The popup does not relay the key back to your page.
- The user manually closes the popup or it closes on its own after the payment flow finishes.

2.5 Popup Blocked

Some browsers block `window.open` if it is not called directly from a user gesture. Always call `openDepositPopup` inside a click handler — never on page load or from a timer.

```
// Good — inside a click handler
document.getElementById('depositBtn').addEventListener('click', () => {
    openDepositPopup('your-store', 25);
});

// Bad — this will be blocked
setTimeout(() => openDepositPopup('your-store', 25), 1000);
```

3. The Payment Flow

Once the user selects a denomination in the catalog (or the popup opens in preselect mode), a Payment Provider checkout iframe fills the popup.

What happens inside the popup:

- The user enters their email address (where the activation key will be sent).
- The user selects a payment method and completes payment.
- The Payment Provider confirms the purchase and emails the activation key.

Your integration does nothing here — this is entirely between the user and the Payment Provider.

After payment:

- The user checks their email and finds the activation key.
- The user copies the key and pastes it into your deposit/top-up form.

- Your backend then calls the Redeem API to validate and register the redemption.

4. Redeem API

This is the server-to-server call your backend makes when a user submits their activation key. It validates the key, marks it as redeemed, and returns the face value so you know how much to credit the user.

> ***Note:*** *All calls must come from your backend server — never from the browser. Your API key would be exposed if called from frontend JavaScript.*

4.1 Endpoint

```
POST https://giftcardshop.co/api/redeem
```

4.2 Authentication

```
Authorization: Bearer apk_YOUR_KEY_HERE
```

4.3 Request Headers

Header	Required	Value
Authorization	Yes	Bearer apk_YOUR_KEY_HERE
Content-Type	Yes	application/json
Accept	Yes	application/json
Idempotency-Key	Recommended	A unique string per redemption attempt (see §4.6)

4.4 Request Body

```
{
  "key":             "XXXX-XXXX-XXXX-XXXX-XXXX",
  "partner_user_id": "player_8723",
  "idempotency_key": "order_9981_redeem"
}
```

Field	Type	Required	Description
`key`	string	Yes	The activation key from the Payment Provider. 4–200 chars, alphanumeric, dashes and underscores only.
`partner_user_id`	string	Yes	Your internal user/player ID. Stored for reconciliation. Max 255 chars.

Field	Type	Required	Description
`idempotency_key`	string	Recommended	Unique string for this attempt. Reuse on retry to prevent double-processing. Max 255 chars.

4.5 Responses

Success — 200 OK

The key was valid, had not been used before, and is now permanently marked as redeemed.

```
{
  "status":           "success",
  "transaction_id":   42,
  "partner_user_id":  "player_8723",
  "amount_cents":     2500,
  "currency":         "EUR"
}
```

Field	Description
`status`	Always "success" on a 200 response.
`transaction_id`	Internal ID. Save this — support uses it to investigate disputes.
`partner_user_id`	The ID you sent, echoed back for confirmation.
`amount_cents`	Face value in cents (e.g. 2500 = €25.00). Amount to credit the user.
`currency`	ISO 4217 code, typically "EUR".

Note: *Credit your user immediately on a 200 response. The key is now permanently consumed.*

Error Responses

All errors follow the same shape:

```
{
  "status":           "<error_code>",
  "transaction_id":   43,
  "partner_user_id":  "player_8723"
}
```

HTTP	Status	Meaning	What To Do
400	`invalid_format`	Key has illegal characters or wrong length.	Prompt user to re-enter the key.

HTTP	Status	Meaning	What To Do
404	`invalid_key`	Key not found or not redeemable.	Ask user to check email and re-enter.
409	`already_redeemed`	Key was already redeemed.	Do NOT credit again. Check transaction_id.
403	`blocked`	Key administratively blocked.	Contact support. Do not retry.
500	`credit_failed`	Internal error.	Retry after delay with same idempotency_key.
401	(no body)	Missing or invalid API key.	Check your API key.
429	(no body)	Rate limit exceeded.	Back off 60s, retry with same idempotency_key.

4.6 Idempotency

Network failures happen. If you send a request and never receive a response, you do not know whether it was processed.

The correct pattern:

- Before each redeem attempt, generate a unique `idempotency_key` (e.g. a UUID, or `order_{id}_redeem`).
- If the request times out or returns 500, **retry with the exact same idempotency_key**.
- The server returns the original response — no double-crediting.

```
const idempotencyKey = `order_${orderId}_redeem`;

// First attempt (times out)
await callRedeem({ key, partnerUserId, idempotencyKey });

// Retry — safe, uses the same idempotencyKey
await callRedeem({ key, partnerUserId, idempotencyKey });
```

Note: *Never generate a new idempotency key on retry — that creates a second attempt.*

4.7 Code Examples

cURL

```
curl -X POST https://giftcardshop.co/api/redeem \
  -H "Authorization: Bearer apk_YOUR_KEY_HERE" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -H "Idempotency-Key: order_9981_redeem" \
  -d '{
    "key": "XXXX-XXXX-XXXX-XXXX-XXXX",
```

```
    "partner_user_id": "player_8723"
  }'
```

JavaScript (Node.js)

```
async function redeemKey({ key, partnerUserId, orderId }) {
    const idempotencyKey = `order_${orderId}_redeem`;

    const response = await fetch('https://giftcardshop.co/api/redeem',
{
        method: 'POST',
        headers: {
            'Authorization':   'Bearer ' +
process.env.PLATFORM_API_KEY,
            'Content-Type':    'application/json',
            'Accept':          'application/json',
            'Idempotency-Key': idempotencyKey,
        },
        body: JSON.stringify({ key, partner_user_id: partnerUserId }),
    });

    const data = await response.json();
    if (data.status === 'success') {
        const amountEur = data.amount_cents / 100;
        console.log(`Credited €${amountEur} — TX
#${data.transaction_id}`);
        return data;
    }
    throw new Error(`Redeem failed: ${data.status}`);
}
```

PHP (Guzzle)

```
$client = new \GuzzleHttp\Client();

$response = $client->post('https://giftcardshop.co/api/redeem', [
    'headers' => [
        'Authorization'   => 'Bearer ' . env('PLATFORM_API_KEY'),
        'Content-Type'    => 'application/json',
        'Idempotency-Key' => 'order_' . $orderId . '_redeem',
    ],
    'json' => [
        'key'             => $activationKey,
        'partner_user_id' => $partnerUserId,
    ],
]);

$data = json_decode($response->getBody(), true);
```

```
if ($data['status'] === 'success') {
    $amountEur = $data['amount_cents'] / 100;
    // credit user balance here
}
```

Python

```
import os, requests

def redeem_key(key: str, partner_user_id: str, order_id: int) -> dict:
    response = requests.post(
        'https://giftcardshop.co/api/redeem',
        headers={
            'Authorization':   'Bearer ' +
os.environ['PLATFORM_API_KEY'],
            'Idempotency-Key': f'order_{order_id}_redeem',
        },
        json={ 'key': key, 'partner_user_id': partner_user_id }
    )
    data = response.json()
    if data['status'] == 'success':
        print(f"Credited €{data['amount_cents'] / 100:.2f}")
        return data
    raise RuntimeError(f"Redeem failed: {data['status']}")
```

4.8 Rate Limits

Limit	Value
Requests per minute per API key	120
HTTP status when exceeded	429 Too Many Requests

Each API key has its own quota — one key per Business Partner means your rate limit is independent of other partners.

5. Testing

5.1 Test Harness (Popup)

A built-in test harness simulates exactly what the Business Partner platform does:

`https://giftcardshop.co/popup/test`

Use it to:

- Open the popup for any of your partner accounts
- Try catalog mode (no params) and preselect mode (`value`, `product_id`)
- Test the Redeem API interactively (paste your API key, enter an activation code, fire the call, see the response)

The test harness is available in both sandbox and production.

5.2 Sandbox vs Production

	Sandbox	Production
Base URL	https://giftcardshop.co	(provided at go-live)
Real money	No	Yes
Activation keys	Test keys from sandbox	Real keys

Work against sandbox until you complete the go-live checklist.

5.3 What to Test

Work through these scenarios before go-live:

- Popup opens — window.open works, popup is 500 × 800, no blank screen.
- Catalog mode — open /popup/checkout/{slug} with no params, confirm offers appear.
- Preselect mode — add ?value=25, confirm Payment Provider iframe opens directly without catalog.
- No offers — popup shows "no offers" message gracefully (not a white screen).
- Popup blocked — test on a browser that blocks popups, verify your UI shows a clear message.
- Redeem success — submit a valid sandbox key, receive 200 success, verify amount_cents.
- Invalid key — submit a random string, expect 404 invalid_key.
- Already redeemed — redeem the same key twice, second call must return 409 already_redeemed.
- Idempotency — redeem with an idempotency_key, retry with the same key, verify same transaction_id.
- Rate limit — fire 121 requests/minute and verify the 121st returns 429.

6. Security Best Practices

- **Never expose your API key in frontend code.** All calls to POST /api/redeem must come from your backend server.
- **Store your key in environment variables:** e.g. PLATFORM_API_KEY=apk_YOUR_KEY_HERE in your .env file.
- **Always send an idempotency_key.** This protects your users from double-crediting if a network error causes a retry.
- **Log transaction_id on every successful redeem.** This is what support uses to investigate disputes. Store it in your database.
- **Never retry on 409 already_redeemed.** This is a terminal state. The key has been used — do not credit the user again.
- **Never retry on 403 blocked or 400 invalid_format.** These are not transient errors.
- **Rotate your API key periodically.** Regenerate under Profile → API Keys. Update your environment variable before regenerating to avoid downtime.

7. Viewing Transactions

All redemptions are visible in the Business Partner Portal under **Portal → Transactions**.

Each row shows: date, partner_user_id, key (last 4 characters only), face value, and status. Use this view to reconcile your records or to pull a transaction_id when contacting support.

8. Go-Live Checklist

Complete every item before requesting production access:

- Business Partner account approved and slug confirmed
- At least one offer approved and live in sandbox
- Popup opens correctly from a button click (not a timer or page load)
- Popup dimensions set to 500 × 800
- Popup-blocked case handled with a user-visible message
- All 10 test scenarios from §5.3 passed in sandbox
- POST /api/redeem called server-side only (API key not in any frontend code)
- API key stored in an environment variable (not in source code)
- idempotency_key sent on every redeem call
- transaction_id logged in your own database on every successful redeem
- Retry logic in place for 500 and network errors (using same idempotency key)

- 409 already_redeemed handled without crediting the user
- 429 handled with a 60-second backoff

Once all items are ticked, contact the platform team to enable your production credentials.

9. Support

If you encounter any issue — missing transactions, key errors, integration questions — contact the platform team and include:

- Your storefront slug (e.g. your-store)
- The transaction_id from the API response (if available)
- The last 4 characters of the activation key (never the full key)
- The approximate time of the request (with timezone)
- The HTTP status and status field from the response body

For urgent production issues, include all of the above so the audit trail can be looked up immediately.