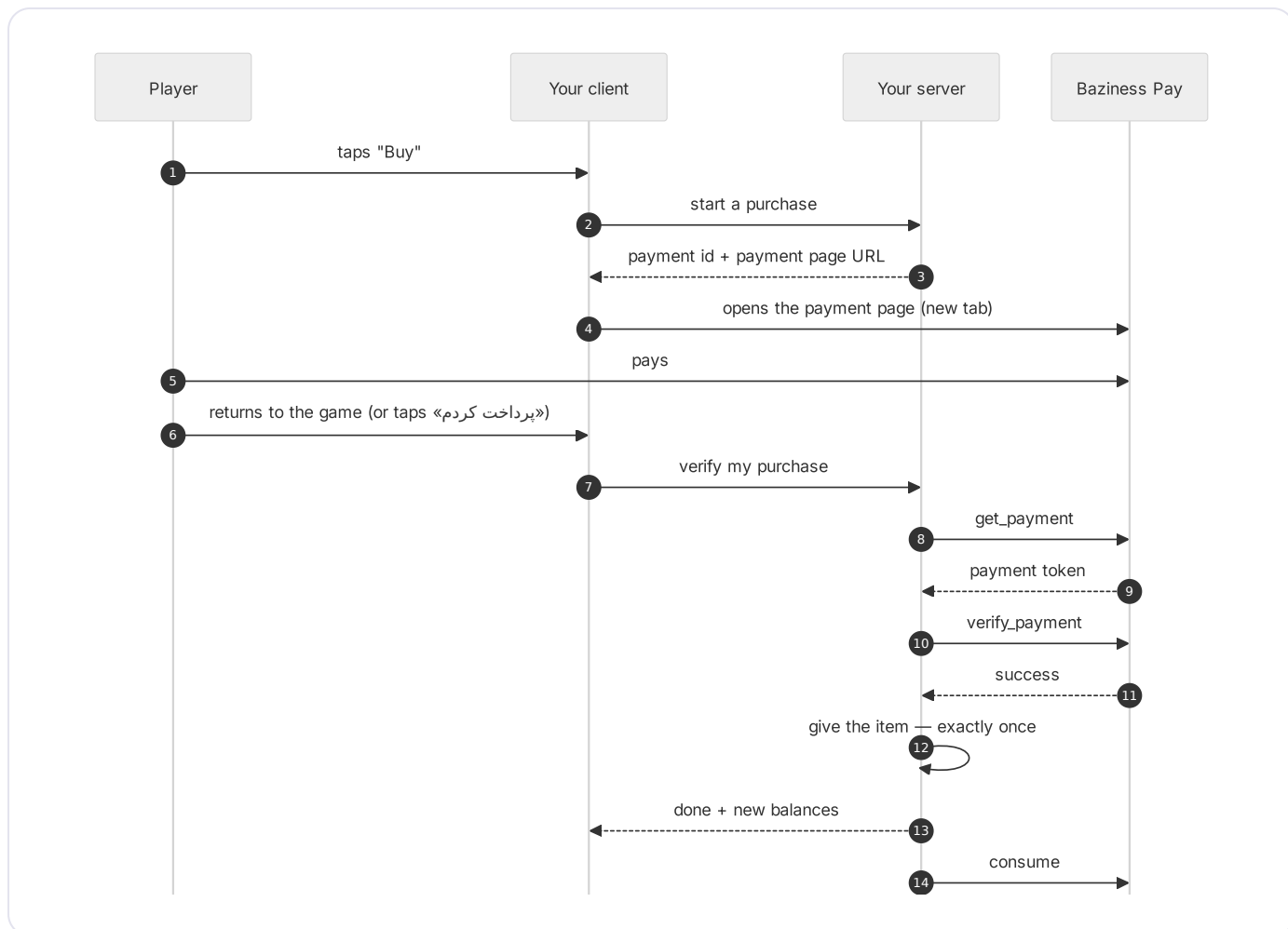


Baziness Pay, step by step

Selling items for real money, in order — open the payment page, check and verify the payment, give the item exactly once, and consume the purchase so it can be bought again.

Before you start

Base URL <code>{pay}</code>	development: <code>https://pay-dev.karizmastudios.org</code> production: <code>https://pay.baziness.com</code>
Who calls it	your server. The client only opens the payment page, and otherwise calls your own server
<code>pname</code>	your Pay package name . It may differ from the platform package name
<code>sku</code>	the product id registered in the Baziness dashboard (for example <code>example_pack</code>)
<code>uid</code>	your stable id for the player, and it must be numeric (send it as a string, e.g. <code>"1042"</code>). A non-numeric uid is rejected with <code>no_required_params</code>



Step 1 — Open the payment page

What it does: shows the player Baziness's payment page for one product.

How: when the player taps "Buy", your client asks **your server** to start a purchase. Your server saves an order (see step 4), builds the page URL and returns it. The client opens it in a **new tab** on the web, or in the system browser on mobile.

HTTP

```
GET {pay}/buy?sku=example_pack&uid=1042&pname=com.karizmastudio.example
```

→ the hosted payment page (opened in a new tab / the system browser)

Tip

Browsers block new tabs that aren't opened **directly** by a tap. Open an empty tab in the tap handler first, then send it to the URL once your server answers ([client code](#)).

Step 2 — Check whether the player paid

What it does: asks Baziness whether this player has a completed, not yet consumed payment for this product, and if so returns its **payment token**.

When to call it: when the player comes back to the game (the window or app regains focus), when they tap «پرداخت کردم» ("I've paid"), and when your shop opens, to catch payments finished while the game was closed.

HTTP

```
POST {pay}/get_payment
Content-Type: application/json

{ "uid": "1042", "sku": "example_pack", "pname": "com.karizmastudio.example" }

→ 200 OK    (paid, not consumed yet)
{ "uid": "1042", "sku": "example_pack", "token": "TEST-633...", "time": 1791366000000,
  "success": true }

→ 200 OK    (nothing to verify yet – the player hasn't paid)
{ "error": "payment_is_not_valid", "success": false }
```

`payment_is_not_valid` simply means "nothing paid yet": keep waiting quietly.

Step 3 — Verify the payment

What it does: confirms with Baziness that the payment token is genuine. **This is the call that protects you:** never give anything based on what the client says.

HTTP

```
POST {pay}/verify_payment
Content-Type: application/x-www-form-urlencoded

uid=1042&token=TEST-633...&sku=example_pack&pname=com.karizmastudio.example

→ 200 OK
{ "success": true }
```

Warning

`verify_payment` takes a **form** body (`application/x-www-form-urlencoded`). The other calls take JSON. Sending JSON here is the most common reason verification fails.

Step 4 — Give the item exactly once

Baziness tells you the payment is real. Making sure the player receives the item **once** is your server's job. Keep a **payments table** in your database:

Column	Meaning
<code>id</code>	your order id (returned to the client in step 1)
<code>user_id</code> , <code>product_id</code> , <code>sku</code> , <code>price</code>	who buys what
<code>status</code>	<code>REDIRECTED</code> (payment page opened) → <code>FULFILLED</code> (item given), or <code>FAILED</code> / <code>CANCELLED</code>
<code>reward_snapshot</code>	exactly what the player was promised when they tapped "Buy"
<code>provider_token</code>	the payment token from step 2, with a unique constraint

Then, after a successful step 3, in **one database transaction**:

1. Lock the order row and check it isn't already `FULFILLED` . If it is, answer "done" without giving anything again.
2. If the same payment token already fulfilled **another** order, mark this one `FAILED` and stop.
3. Give the items in `reward_snapshot` , not the product's current definition. An admin who edits a pack while the player pays must not change the order.
4. Save the token and set the order to `FULFILLED` .

Step 5 — Consume the purchase

What it does: marks the purchase as used, so the same product can be bought again. Until it is consumed, step 2 keeps returning it and a second purchase of the SKU is blocked.

When: right **after** step 4 is committed. Try a few times if it fails. The item is already given, so a failure here never blocks the player.

HTTP

```
POST {pay}/consume
Content-Type: application/json

{ "uid": "1042", "sku": "example_pack", "pname": "com.karizmastudio.example" }
```

```
→ 200 OK
{ "success": true }
```

What the player sees

Moment	Your client shows
after tapping "Buy"	a small window: "the Baziness payment page is open, come back after paying", with a «پرداخت کردم» button
back in the game, payment not found yet	nothing new (quiet re-check); only a tap on «پرداخت کردم» says "payment not found yet, try again in a moment"
payment verified	"purchase successful", and the updated balances
window closed before paying	nothing is cancelled; the payment is checked again the next time the shop opens

Testing

- On **pay-dev**, the payment page offers «خرید آزمایشی» (test purchase): the whole flow, from step 1 to step 5, runs without real money.
- Without any gateway access, use the **local fake gateway** from the example game: it speaks the same protocol, with a «پرداخت موفق» button ([settings](#)).

Store builds (Cafe Bazaar / Myket)

A build published in Cafe Bazaar or Myket must use **that store's own in-app billing** instead of Baziness Pay. Keep the same SKU names in both, so one product list serves every build, and send `buildStore` on Baziness page links in those builds ([parameters](#)).