

Platform API, step by step

Every platform endpoint in the order you use it — what it does, when to call it, what to send and what comes back. No code; ready-made code is in the Code samples section.

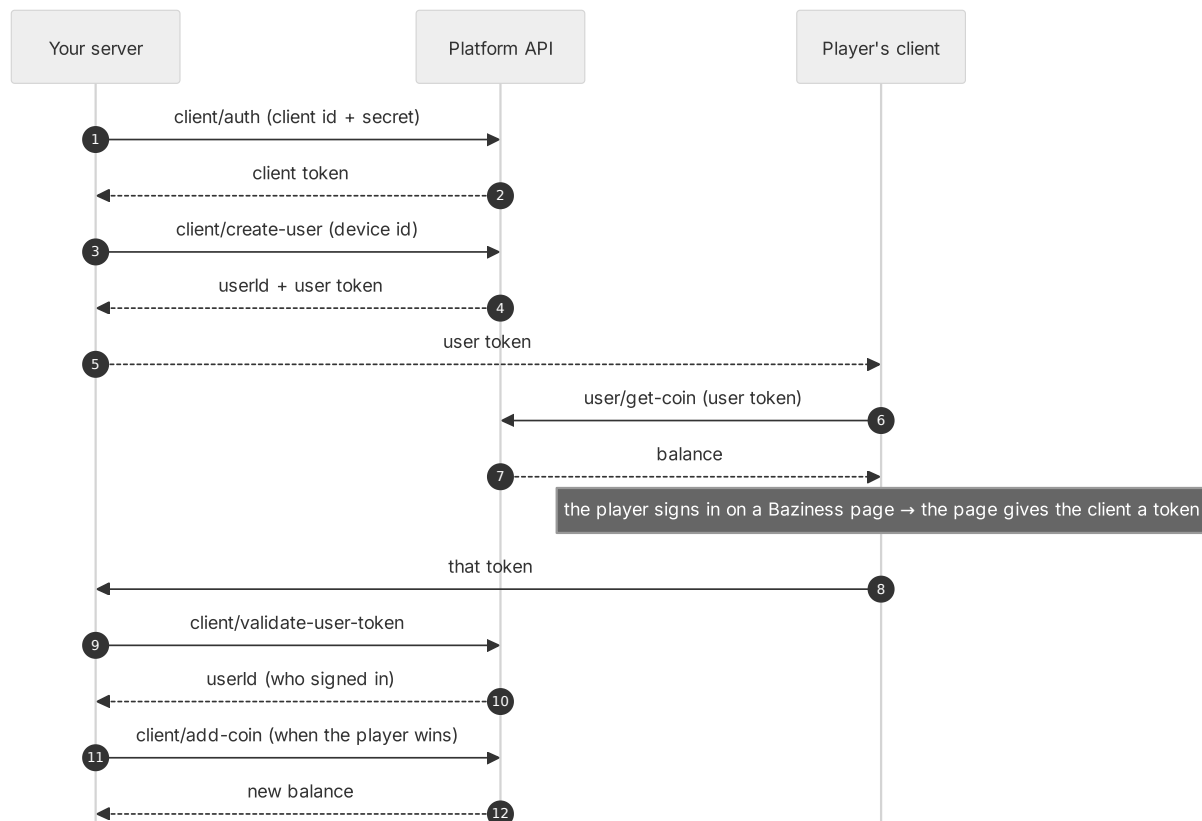
Before you start

Important

User tokens are moving to **15-minute access tokens, refresh tokens kept on your server, and one-time link codes**. The steps below still work during the migration. What to change, and by when: [Token migration](#).

Base URL {api}	development: <code>https://platform-dev.karizmastudios.org/api</code> production: <code>https://platform.karizmastudio.com/api</code>
Format	JSON request and response bodies (<code>Content-Type: application/json</code>), field names in camelCase
Who calls it	your server , except step 6 (<code>user/get-coin</code>), which the player's client calls
Authentication	steps 2–5 need <code>Authorization: Bearer <client token></code> from step 1; step 6 needs the player's user token

The whole sequence for a new player:



Step 1 — Get a client token

What it does: exchanges your **client id** and **client secret** for a **client token**. Every other server call (steps 2–5) sends this token in the `Authorization` header.

When to call it: on the first platform call after your server starts, and again whenever any call answers **401**, because the token has expired.

HTTP

```

POST {api}/client/auth
Content-Type: application/json

{ "clientId": 12, "clientSecret": "....." }

→ 200 OK
{ "clientId": 12, "token": "eyJhbGciOiJIUzI1NiIs..." }
  
```

Rules

- Keep the token **in your server's memory** and reuse it for every call. Don't request a new one per call.
- When a call answers **401 Unauthorized**, call `client/auth` again and **retry that call once**.

- The client token is as sensitive as the secret: never send it to a client.

Step 2 — Create a Baziness user for a player

What it does: gives one of your players a Baziness user. It returns:

- the Baziness `userId` — save it on your player's record; you need it for every later call
- a **user token** for this player — send it to your client
- `isLinkedToAccount` — whether this user has signed in with a phone number (`true`) or is still anonymous (`false`)

When to call it: the first time a player of yours needs Baziness, which is usually their first launch while the platform is on. If you already have a `userId` saved for the player, use step 3 instead.

HTTP

```
POST {api}/client/create-user
Authorization: Bearer <client token>
Content-Type: application/json

{ "deviceId": "8f0c2b7e-...", "platform": "Web", "store": "Direct", "linkedToken": null }
```

→ 200 OK

```
{ "userId": 239236, "token": "eyJhbGciOiJIUzI1NiIs... ", "isLinkedToAccount": false }
```

Field	Value
<code>deviceId</code>	your player's stable id, for example the device id your game generated on first launch
<code>platform</code>	<code>Web</code> , <code>Android</code> , <code>Ios</code> or <code>Editor</code>
<code>store</code>	<code>Direct</code> (web), <code>GooglePlay</code> , <code>AppStore</code> , <code>CafeBazaar</code> or <code>Myket</code>
<code>linkedToken</code>	a user token the player already has, or <code>null</code> . The Baziness app opens games with <code>?linkedToken=...</code> ; passing it here binds the player to that Baziness account

Important

If the returned `userId` is **already saved on another of your players**, it is the same person, for example on a new phone. Don't save it a second time. Switch this device to that player instead (see [Login → account switching](#)). Put a **unique** constraint on the column that stores the `userId`.

Step 3 — Get a fresh user token

What it does: returns a new user token for a Baziness user you already know, and their current `isLinkedToAccount`.

When to call it: on every later session of the player, typically each time your client boots. Then the client always has a valid token, and you learn when an anonymous user has signed in (`isLinkedToAccount` became `true`).

HTTP

```
GET {api}/client/get-user-token/239236
Authorization: Bearer <client token>

→ 200 OK
{ "userId": 239236, "token": "eyJhbGciOiJIUzI1NiIs...", "isLinkedToAccount": true }
```

Step 4 — Verify a token from the client

What it does: tells you whether a user token is genuine, and **which Baziness user it belongs to**.

When to call it: whenever a token arrives from a client and you're about to act on it. The main case: the player signs in on the Baziness login page, the page gives your client a token, and your client sends it to your server. **Never link an account or give anything based on a token you haven't verified.**

HTTP

```
GET {api}/client/validate-user-token/eyJhbGciOiJIUzI1NiIs...
Authorization: Bearer <client token>

→ 200 OK
{ "userId": 239236, "isValid": true }

→ 200 OK (an invalid or expired token is NOT an HTTP error)
{ "userId": 0, "isValid": false }
```

Warning

An invalid token is **not** an HTTP error: the answer is `200` with `"isValid": false` and `"userId": 0`. Always check `isValid`.

Step 5 — Give Gita coins

What it does: adds Gita coins to a player's Baziness balance and returns the new balance.

When to call it: when the player earns coins in your game, for example by winning a match, ranking in a tournament or opening a gift.

HTTP

```
POST {api}/client/add-coin
Authorization: Bearer <client token>
Content-Type: application/json

{
  "userId": 239236,
  "amount": 50,
  "transactionId": "239236_50_9f2c1a7d0b4e8c3f5a6d2e1b",
  "metaData": { "source": "match_win", "matchId": 17 }
}
```

→ 200 OK
{ "resourceId": 1, "amount": 2450 } ← the player's new balance

→ 400 Bad Request (this transactionId was already credited)
{ "success": false, "error": "add resource failed" }

Field	Value
<code>userId</code>	the Baziness user id from step 2
<code>amount</code>	how many coins to add (positive)
<code>transactionId</code>	a unique id for this payout (see below). The platform credits each id only once
<code>metaData</code>	any JSON you want stored with the transaction (source, match id ...), useful for reports

Making `transactionId` safe: build it from **what the payout is**, not from the time:

Text

```
transactionId = <userId>_<amount>_<first 24 hex chars of sha1(grantKey)>
grantKey      = "match:17" | "reward_claim:42" | "payment:88:GITA_COIN" | "tutorial_gift:
<userId>"
```

- The same payout always produces the same id. A retry, a double tap or a repeated request is refused with **400** and never pays twice.
- **Treat that **400** as "already paid"** and finish your own bookkeeping, for example marking the reward as claimed. Otherwise a crash between the credit and your save leaves the player stuck on an error.
- An id made from the clock (**<userId>_<amount>_<time>**) credits again on every call. Use it only for grants that are genuinely repeatable.

Step 6 — Read the balance (client)

What it does: returns the player's Gita coin balance. This is the only platform call the **client** makes, with the player's **user token**. Cross-origin requests are allowed, so a browser game can call it directly.

When to call it: after the client receives the token, when a Baziness page closes (the player may have spent coins), when the game comes back to the foreground, and after your server gave coins.

HTTP

```
GET {api}/user/get-coin
Authorization: Bearer <user token>

→ 200 OK
{ "resourceId": 1, "amount": 2450 }
```

Errors at a glance

Answer	Where	Meaning	What to do
401 Unauthorized	steps 2–5	missing or expired client token	call step 1 again, retry once
200 + "isValid": false	step 4	the token is not genuine or has expired	refuse the request; the client should sign in again
400 + "add resource failed"	step 5	this transactionId was already credited	treat it as paid
network error / 5xx	any	the platform is unreachable	keep the game running without Baziness features; retry later