CODE SAMPLES

Server code

Ready-made server code for every step — the platform client, your own endpoints, Gita coin payouts, Baziness Pay and campaign tags — in TypeScript, C# and Python.

These samples implement the API steps and guides. Each one comes in **TypeScript (Node)**, **C# (.NET)** and **Python**; pick a tab, and the choice is remembered on every page. They use Express, ASP.NET minimal APIs and FastAPI, but the logic carries over to any framework. Where a sample calls `db…`, plug in your own database.

Settings

.env (server)

```
# --- Baziness platform: accounts, Gita coins, reward pages -------------------------------
BAZINESS_PLATFORM_ENABLED=false          # master switch: nothing shows until "true"
BAZINESS_PLATFORM_USE_PRODUCTION=false   # selects the default API + web URLs below
BAZINESS_CLIENT_ID=
BAZINESS_CLIENT_SECRET=
BAZINESS_PLATFORM_PACKAGE_NAME=com.yourstudio.yourgame
# Optional overrides (derived from USE_PRODUCTION when blank):
#   dev  api https://platform-dev.karizmastudios.org/api   web https://platform-web-
dev.karizmastudios.org
#   prod api https://platform.karizmastudio.com/api        web https://baziness.com
BAZINESS_PLATFORM_API_URL=
BAZINESS_PLATFORM_WEB_URL=
BAZINESS_GAME_NAME=yourgame              # ?game_name= of your pages on the platform
(rewards, game offer)
# Optional: the tutorial offer page (blank = {web}/game-offer?game_name=<BAZINESS_GAME_NAME>)
BAZINESS_GAME_OFFER_URL=

# --- Baziness Pay: real-money purchases ---------------------------------------------------
BAZINESS_PAY_ENABLED=false              # buy buttons stay disabled until "true"
#   dev  https://pay-dev.karizmastudios.org   prod  https://pay.baziness.com
BAZINESS_PAY_BASE_URL=https://pay-dev.karizmastudios.org
BAZINESS_PAY_PACKAGE_NAME=com.yourstudio.yourgame   # the Pay app's package (pname) — may
differ from BAZINESS_PLATFORM_PACKAGE_NAME
BAZINESS_SKU_PREFIX=                    # optional prefix: SKU = prefix + product id
DEV_FAKE_BAZINESS_PAY=false             # local development only (see the Pay page)
```

The platform client

One small class that does steps 1–5: it gets the client token on first use, gets it again after a `401`, and builds payout ids from a grant key.

TypeScript (Node)

```
// platform/baziness.ts — all credentials stay on the server.
import { createHash } from "node:crypto";

const DEV_API = "https://platform-dev.karizmastudios.org/api";
const PROD_API = "https://platform.karizmastudio.com/api";
const DEV_WEB = "https://platform-web-dev.karizmastudios.org";
const PROD_WEB = "https://baziness.com";

const env = process.env;
const ENABLED = env.BAZINESS_PLATFORM_ENABLED === "true";
const USE_PROD = env.BAZINESS_PLATFORM_USE_PRODUCTION === "true";
const CLIENT_ID = env.BAZINESS_CLIENT_ID ?? "";
const CLIENT_SECRET = env.BAZINESS_CLIENT_SECRET ?? "";
const PACKAGE_NAME = env.BAZINESS_PLATFORM_PACKAGE_NAME ?? "";
const trim = (u: string) => u.replace(/\/+$/, "");
export const API_BASE = trim(env.BAZINESS_PLATFORM_API_URL || (USE_PROD ? PROD_API :
DEV_API));
export const WEB_BASE = trim(env.BAZINESS_PLATFORM_WEB_URL || (USE_PROD ? PROD_WEB :
DEV_WEB));

export const platformEnabled = () => ENABLED && CLIENT_ID !== "" && CLIENT_SECRET !== "";

/** Public, non-secret config for the browser. */
export const platformClientConfig = () => ({ webBase: WEB_BASE, apiBase: API_BASE,
packageName: PACKAGE_NAME });

export interface PlatformUser { userId: number; token: string; isLinkedToAccount: boolean }

let bearer: string | null = null;

async function authenticate(): Promise<void> {
  const res = await fetch(`${API_BASE}/client/auth`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ clientId: CLIENT_ID, clientSecret: CLIENT_SECRET }),
  });
  if (!res.ok) throw new Error(`client/auth failed (${res.status})`);
  const json = (await res.json()) as Record<string, unknown>;
  bearer = String(json.token ?? json.Token ?? "");
  if (!bearer) throw new Error("client/auth returned no token");
}

async function platformFetch(method: "GET" | "POST", path: string, body?: unknown) {
  if (!bearer) await authenticate();
  const send = () =>
    fetch(`${API_BASE}${path}`, {
      method,
      headers: { Authorization: `Bearer ${bearer}`, ...(body ? { "Content-Type":
"application/json" } : {}) },
      body: body ? JSON.stringify(body) : undefined,
    });
```

```
  let res = await send();
  if (res.status === 401) { await authenticate(); res = await send(); } // bearer expired
  if (!res.ok) throw Object.assign(new Error(`${path} failed (${res.status})`), { status:
res.status }); // 400 from add-coin = already paid
  return (await res.json()) as Record<string, unknown>;
}

const readUser = (j: Record<string, unknown>): PlatformUser => ({
  userId: Number(j.userId ?? j.UserId ?? 0),
  token: String(j.token ?? j.Token ?? ""),
  isLinkedToAccount: Boolean(j.isLinkedToAccount ?? j.IsLinkedToAccount ?? false),
});

export const createUser = async (deviceId: string, platform: string, store: string,
linkedToken: string | null) =>
  readUser(await platformFetch("POST", "/client/create-user", { deviceId, platform, store,
linkedToken }));

export const getUserAccessToken = async (bazinessUserId: number) =>
  readUser(await platformFetch("GET", `/client/get-user-token/${bazinessUserId}`));

export async function validateUserAccessToken(token: string): Promise<number> {
  const j = await platformFetch("GET", `/client/validate-user-
token/${encodeURIComponent(token)}`);
  if (!(j.isValid ?? j.IsValid)) throw new Error("invalid user token");
  return Number(j.userId ?? j.UserId ?? 0);
}

/** Credit Gita coins. `grantKey` makes the transaction id deterministic: a repeat answers
400 (= already paid). */
export async function addCoin(bazinessUserId: number, amount: number, grantKey: string,
metaData?: object) {
  const hash = createHash("sha1").update(grantKey).digest("hex").slice(0, 24);
  const j = await platformFetch("POST", "/client/add-coin", {
    userId: bazinessUserId, amount, transactionId: `${bazinessUserId}_${amount}_${hash}`,
metaData: metaData ?? null,
  });
  return Number(j.amount ?? j.Amount ?? 0); // the player's new balance
}
```

C# (.NET)

```
// BazinessPlatformClient.cs — all credentials stay on the server.
// Program.cs:  builder.Services.AddSingleton<PlatformSettings>(…);
//              builder.Services.AddHttpClient<BazinessPlatformClient>();
using System.Net;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;

public record PlatformSettings(bool Enabled, bool UseProduction, long ClientId, string
ClientSecret, string PackageName)
{
    public string ApiBase => UseProduction ? "https://platform.karizmastudio.com/api" :
"https://platform-dev.karizmastudios.org/api";
    public string WebBase => UseProduction ? "https://baziness.com" : "https://platform-web-
dev.karizmastudios.org";
```

```
    public bool IsOn => Enabled && ClientId != 0 && !string.IsNullOrEmpty(ClientSecret);
}

public record PlatformUser(long UserId, string Token, bool IsLinkedToAccount);

public sealed class BazinessPlatformClient(HttpClient http, PlatformSettings settings)
{
    static readonly JsonSerializerOptions Json = new(JsonSerializerDefaults.Web); //
camelCase + case-insensitive
    static string? bearer;                                    // the client token, shared by
every request
    static readonly SemaphoreSlim authLock = new(1, 1);

    async Task AuthenticateAsync()
    {
        await authLock.WaitAsync();
        try
        {
            var res = await http.PostAsJsonAsync($"{settings.ApiBase}/client/auth",
                new { clientId = settings.ClientId, clientSecret = settings.ClientSecret });
            res.EnsureSuccessStatusCode();
            bearer = (await res.Content.ReadFromJsonAsync<JsonElement>
()).GetProperty("token").GetString()
                      ?? throw new InvalidOperationException("client/auth returned no token");
        }
        finally { authLock.Release(); }
    }

    async Task<JsonElement> SendAsync(HttpMethod method, string path, object? body = null)
    {
        if (bearer is null) await AuthenticateAsync();
        async Task<HttpResponseMessage> Send()
        {
            var req = new HttpRequestMessage(method, $"{settings.ApiBase}{path}");
            req.Headers.Authorization = new AuthenticationHeaderValue("Bearer", bearer);
            if (body is not null) req.Content = JsonContent.Create(body, options: Json);
            return await http.SendAsync(req);
        }
        var res = await Send();
        if (res.StatusCode == HttpStatusCode.Unauthorized) { await AuthenticateAsync(); res =
await Send(); } // token expired
        if (!res.IsSuccessStatusCode) throw new HttpRequestException($"{path} failed
({(int)res.StatusCode})", null, res.StatusCode);
        return await res.Content.ReadFromJsonAsync<JsonElement>(Json);
    }

    static PlatformUser ReadUser(JsonElement j) => new(
        j.GetProperty("userId").GetInt64(), j.GetProperty("token").GetString()!,
j.GetProperty("isLinkedToAccount").GetBoolean());

    public async Task<PlatformUser> CreateUserAsync(string deviceId, string platform, string
store, string? linkedToken) =>
        ReadUser(await SendAsync(HttpMethod.Post, "/client/create-user", new { deviceId,
platform, store, linkedToken }));

    public async Task<PlatformUser> GetUserAccessTokenAsync(long bazinessUserId) =>
        ReadUser(await SendAsync(HttpMethod.Get, $"/client/get-user-
token/{bazinessUserId}"));

    /// <summary>The server-side gate for every token a client sends. Throws when the token
```

```
is invalid.</summary>
    public async Task<long> ValidateUserAccessTokenAsync(string token)
    {
        var j = await SendAsync(HttpMethod.Get, $"/client/validate-user-
token/{Uri.EscapeDataString(token)}");
        if (!j.GetProperty("isValid").GetBoolean()) throw new
InvalidOperationException("invalid user token");
        return j.GetProperty("userId").GetInt64();
    }

    /// <summary>
    /// Credit Gita coins. The transaction id comes from grantKey (what the payout IS, e.g.
"match:17"),
    /// so a repeat is refused with HTTP 400 — treat that 400 as "already paid".
    /// </summary>
    public async Task<long> AddCoinAsync(long bazinessUserId, long amount, string grantKey,
object? metaData = null)
    {
        var hash = Convert.ToHexString(SHA1.HashData(Encoding.UTF8.GetBytes(grantKey)))
[..24].ToLowerInvariant();
        var j = await SendAsync(HttpMethod.Post, "/client/add-coin",
            new { userId = bazinessUserId, amount, transactionId = $"
{bazinessUserId}_{amount}_{hash}", metaData });
        return j.GetProperty("amount").GetInt64(); // the player's new balance
    }
}
```

Python

```
# baziness/platform.py — all credentials stay on the server.
import hashlib
import os
import requests

DEV_API, PROD_API = "https://platform-dev.karizmastudios.org/api",
"https://platform.karizmastudio.com/api"
DEV_WEB, PROD_WEB = "https://platform-web-dev.karizmastudios.org", "https://baziness.com"

ENABLED = os.getenv("BAZINESS_PLATFORM_ENABLED", "false").lower() == "true"
USE_PROD = os.getenv("BAZINESS_PLATFORM_USE_PRODUCTION", "false").lower() == "true"
CLIENT_ID = os.getenv("BAZINESS_CLIENT_ID", "")
CLIENT_SECRET = os.getenv("BAZINESS_CLIENT_SECRET", "")
PACKAGE_NAME = os.getenv("BAZINESS_PLATFORM_PACKAGE_NAME", "")
API_BASE = (os.getenv("BAZINESS_PLATFORM_API_URL") or (PROD_API if USE_PROD else
DEV_API)).rstrip("/")
WEB_BASE = (os.getenv("BAZINESS_PLATFORM_WEB_URL") or (PROD_WEB if USE_PROD else
DEV_WEB)).rstrip("/")


def platform_enabled() -> bool:
    return ENABLED and bool(CLIENT_ID) and bool(CLIENT_SECRET)


def platform_client_config() -> dict:
    """Public, non-secret config for the browser."""
    return {"webBase": WEB_BASE, "apiBase": API_BASE, "packageName": PACKAGE_NAME}


def _pick(j: dict, name: str, default=None):
```

```python
    # The platform is .NET: accept camelCase and PascalCase.
    return j.get(name, j.get(name[0].upper() + name[1:], default))


class BazinessPlatform:
    def __init__(self) -> None:
        self._bearer: str | None = None
        self._http = requests.Session()

    def _authenticate(self) -> None:
        r = self._http.post(f"{API_BASE}/client/auth",
                            json={"clientId": int(CLIENT_ID), "clientSecret": CLIENT_SECRET},
timeout=10)
        r.raise_for_status()
        self._bearer = _pick(r.json(), "token")
        if not self._bearer:
            raise RuntimeError("client/auth returned no token")

    def _call(self, method: str, path: str, body: dict | None = None) -> dict:
        if not self._bearer:
            self._authenticate()
        send = lambda: self._http.request(method, f"{API_BASE}{path}", json=body, timeout=10,
                                          headers={"Authorization": f"Bearer
{self._bearer}"})
        r = send()
        if r.status_code == 401:  # bearer expired
            self._authenticate()
            r = send()
        r.raise_for_status()
        return r.json()

    @staticmethod
    def _user(j: dict) -> dict:
        return {"userId": int(_pick(j, "userId", 0)), "token": str(_pick(j, "token", "")),
                "isLinkedToAccount": bool(_pick(j, "isLinkedToAccount", False))}

    def create_user(self, device_id: str, platform: str, store: str, linked_token: str |
None) -> dict:
        return self._user(self._call("POST", "/client/create-user", {
            "deviceId": device_id, "platform": platform, "store": store, "linkedToken":
linked_token}))

    def get_user_access_token(self, baziness_user_id: int) -> dict:
        return self._user(self._call("GET", f"/client/get-user-token/{baziness_user_id}"))

    def validate_user_access_token(self, token: str) -> int:
        j = self._call("GET", f"/client/validate-user-token/{requests.utils.quote(token,
safe='')}")
        if not _pick(j, "isValid", False):
            raise ValueError("invalid user token")
        return int(_pick(j, "userId", 0))

    def add_coin(self, baziness_user_id: int, amount: int, grant_key: str, meta: dict | None
= None) -> int:
        digest = hashlib.sha1(grant_key.encode()).hexdigest()[:24]
        j = self._call("POST", "/client/add-coin", {
            "userId": baziness_user_id, "amount": amount,
            "transactionId": f"{baziness_user_id}_{amount}_{digest}", "metaData": meta})
        return int(_pick(j, "amount", 0))  # the player's new balance
```

```
platform = BazinessPlatform()
```

Telling the client the settings

`GET /api/platform/config`: whether Baziness is on, and where its pages and API are. It has no side effects and never includes the secret.

TypeScript (Node)

```
import express from "express";
import { platformClientConfig, platformEnabled, WEB_BASE } from "./platform/baziness";

const GAME_NAME = process.env.BAZINESS_GAME_NAME ?? "example"; // your game's name on the
platform
const GAME_OFFER_URL = process.env.BAZINESS_GAME_OFFER_URL || `${WEB_BASE}/game-offer?
game_name=${encodeURIComponent(GAME_NAME)}`;

const app = express();
app.use(express.json());

// Side-effect free: tells the client whether Baziness is on and where its pages live. Never
the secret.
app.get("/api/platform/config", (_req, res) => {
  res.json(platformEnabled()
    ? { enabled: true, ...platformClientConfig(), gameName: GAME_NAME, gameOfferUrl:
GAME_OFFER_URL }
    : { enabled: false });
});
```

C# (.NET)

```
// Side-effect free: tells the client whether Baziness is on and where its pages live. Never
the secret.
app.MapGet("/api/platform/config", (PlatformSettings s, IConfiguration cfg) =>
{
    if (!s.IsOn) return Results.Ok(new { enabled = false });
    var gameName = cfg["Baziness:GameName"] ?? "example";
    var gameOfferUrl = cfg["Baziness:GameOfferUrl"] is { Length: > 0 } custom
        ? custom
        : $"{s.WebBase}/game-offer?game_name={Uri.EscapeDataString(gameName)}";
    return Results.Ok(new { enabled = true, webBase = s.WebBase, apiBase = s.ApiBase,
packageName = s.PackageName, gameName, gameOfferUrl });
});
```

Python

```
import os
from urllib.parse import quote
from fastapi import FastAPI
from baziness.platform import WEB_BASE, platform_client_config, platform_enabled

GAME_NAME = os.getenv("BAZINESS_GAME_NAME", "example")  # your game's name on the platform
```

```
GAME_OFFER_URL = os.getenv("BAZINESS_GAME_OFFER_URL") or f"{WEB_BASE}/game-offer?game_name=
{quote(GAME_NAME)}"

app = FastAPI()


@app.get("/api/platform/config")
def platform_config():
    """Side-effect free: tells the client whether Baziness is on and where its pages live.
Never the secret."""
    if not platform_enabled():
        return {"enabled": False}
    return {"enabled": True, **platform_client_config(), "gameName": GAME_NAME,
"gameOfferUrl": GAME_OFFER_URL}
```

Login endpoints

`POST /api/platform/token` and `POST /api/platform/login`, as described in Login → what your server provides.

TypeScript (Node)

```
import { createUser, getUserAccessToken, platformClientConfig, platformEnabled,
validateUserAccessToken } from "./platform/baziness";
import { db } from "./db"; // your own data layer

app.post("/api/platform/token", async (req, res) => {
  const deviceId = String(req.body.deviceId ?? "").trim();
  const linkedToken: string | null = req.body.linkedToken ?? null;
  if (!deviceId) return res.status(400).json({ error: "deviceId required" });
  if (!platformEnabled()) return res.json({ enabled: false });

  const config = { enabled: true, ...platformClientConfig() };
  // Pass first-touch attribution so a player who signs in before your normal login is still
credited.
  const user = await db.ensureUserByDevice(deviceId, { utm: req.body.utm, tag: req.body.tag
});
  try {
    if (!user.bazinessUserId) {
      const created = await createUser(deviceId, "Web", "Direct", linkedToken);
      const owner = await db.findUserByBazinessId(created.userId);
      if (owner && owner.id !== user.id) {
        return res.json({ ...config, token: created.token, bazinessUserId: created.userId,
          isLinkedToAccount: created.isLinkedToAccount, switchToDeviceId: owner.deviceId });
      }
      await db.linkBaziness(user.id, created.userId, created.isLinkedToAccount);
      return res.json({ ...config, token: created.token, bazinessUserId: created.userId,
isLinkedToAccount: created.isLinkedToAccount });
    }
    const fresh = await getUserAccessToken(user.bazinessUserId);
    if (fresh.isLinkedToAccount !== user.isBazinessLinked) await db.setLinked(user.id,
fresh.isLinkedToAccount);
    return res.json({ ...config, token: fresh.token, bazinessUserId: fresh.userId,
isLinkedToAccount: fresh.isLinkedToAccount });
```

```
  } catch (err) {
    console.error("[platform] token failed:", (err as Error).message);
    return res.json({ ...config, error: "platform_unavailable" }); // never block the boot
  }
});

app.post("/api/platform/login", async (req, res) => {
  const deviceId = String(req.body.deviceId ?? "").trim();
  const token = String(req.body.token ?? "").trim();
  if (!deviceId || !token) return res.status(400).json({ error: "deviceId and token required"
});
  if (!platformEnabled()) return res.json({ ok: false, reason: "disabled" });
  try {
    const bazinessUserId = await validateUserAccessToken(token); // the server-side gate
    const owner = await db.findUserByBazinessId(bazinessUserId);
    const me = await db.ensureUserByDevice(deviceId);
    if (owner && owner.id !== me.id) return res.json({ ok: true, switchToDeviceId:
owner.deviceId });
    await db.linkBaziness(me.id, bazinessUserId, true);
    return res.json({ ok: true });
  } catch {
    return res.json({ ok: false, reason: "invalid" });
  }
});
```

C# (.NET)

```
app.MapPost("/api/platform/token", async (TokenRequest body, BazinessPlatformClient platform,
    IUserStore users, PlatformSettings pub, ILogger<Program> log) =>
{
    if (string.IsNullOrWhiteSpace(body.DeviceId)) return Results.BadRequest(new { error =
"deviceId required" });
    if (!pub.IsOn) return Results.Ok(new { enabled = false });

    var user = await users.EnsureByDeviceAsync(body.DeviceId);
    try
    {
        PlatformUser auth;
        if (user.BazinessUserId is null)
        {
            auth = await platform.CreateUserAsync(body.DeviceId, "Web", "Direct",
body.LinkedToken);
            var owner = await users.FindByBazinessUserIdAsync(auth.UserId);
            if (owner is not null && owner.Id != user.Id)
                return Results.Ok(new { enabled = true, pub.WebBase, pub.ApiBase,
pub.PackageName,
                    token = auth.Token, bazinessUserId = auth.UserId, auth.IsLinkedToAccount,
                    switchToDeviceId = owner.DeviceId });
            await users.LinkBazinessAsync(user.Id, auth.UserId, auth.IsLinkedToAccount);
        }
        else
        {
            auth = await platform.GetUserAccessTokenAsync(user.BazinessUserId.Value);
            if (auth.IsLinkedToAccount != user.IsBazinessLinked)
                await users.SetLinkedAsync(user.Id, auth.IsLinkedToAccount);
        }
        return Results.Ok(new { enabled = true, pub.WebBase, pub.ApiBase, pub.PackageName,
            token = auth.Token, bazinessUserId = auth.UserId, auth.IsLinkedToAccount });
    }
```

```
    catch (Exception ex)
    {
        log.LogError(ex, "platform token failed");
        return Results.Ok(new { enabled = true, pub.WebBase, pub.ApiBase, pub.PackageName,
            error = "platform_unavailable" }); // never block the boot
    }
});

app.MapPost("/api/platform/login", async (LoginRequest body, BazinessPlatformClient platform,
IUserStore users, PlatformSettings pub) =>
{
    if (string.IsNullOrWhiteSpace(body.DeviceId) || string.IsNullOrWhiteSpace(body.Token))
        return Results.BadRequest(new { error = "deviceId and token required" });
    if (!pub.IsOn) return Results.Ok(new { ok = false, reason = "disabled" });
    try
    {
        long bazinessUserId = await platform.ValidateUserAccessTokenAsync(body.Token); //
throws when invalid
        var owner = await users.FindByBazinessUserIdAsync(bazinessUserId);
        var me = await users.EnsureByDeviceAsync(body.DeviceId);
        if (owner is not null && owner.Id != me.Id)
            return Results.Ok(new { ok = true, switchToDeviceId = owner.DeviceId });
        await users.LinkBazinessAsync(me.Id, bazinessUserId, isLinked: true);
        return Results.Ok(new { ok = true });
    }
    catch
    {
        return Results.Ok(new { ok = false, reason = "invalid" });
    }
});

public record TokenRequest(string DeviceId, string? LinkedToken);
public record LoginRequest(string DeviceId, string Token);
```

Python

```
from fastapi import FastAPI
from pydantic import BaseModel
from baziness.platform import platform, platform_client_config, platform_enabled
from app import db  # your own data layer

app = FastAPI()


class TokenRequest(BaseModel):
    deviceId: str
    linkedToken: str | None = None


class LoginRequest(BaseModel):
    deviceId: str
    token: str


@app.post("/api/platform/token")
def platform_token(body: TokenRequest):
    if not platform_enabled():
        return {"enabled": False}
    config = {"enabled": True, **platform_client_config()}
```

```
    user = db.ensure_user_by_device(body.deviceId)
    try:
        if not user.baziness_user_id:
            created = platform.create_user(body.deviceId, "Web", "Direct", body.linkedToken)
            owner = db.find_user_by_baziness_user_id(created["userId"])
            if owner and owner.id != user.id:
                return {**config, "token": created["token"], "bazinessUserId":
created["userId"],
                        "isLinkedToAccount": created["isLinkedToAccount"],
"switchToDeviceId": owner.device_id}
            db.link_baziness(user.id, created["userId"], created["isLinkedToAccount"])
            auth = created
        else:
            auth = platform.get_user_access_token(user.baziness_user_id)
            if auth["isLinkedToAccount"] != user.is_baziness_linked:
                db.set_linked(user.id, auth["isLinkedToAccount"])
        return {**config, "token": auth["token"], "bazinessUserId": auth["userId"],
                "isLinkedToAccount": auth["isLinkedToAccount"]}
    except Exception as exc:  # platform down: never block the boot
        print("[platform] token failed:", exc)
        return {**config, "error": "platform_unavailable"}


@app.post("/api/platform/login")
def platform_login(body: LoginRequest):
    if not platform_enabled():
        return {"ok": False, "reason": "disabled"}
    try:
        baziness_user_id = platform.validate_user_access_token(body.token)  # the server-side
gate
    except Exception:
        return {"ok": False, "reason": "invalid"}
    owner = db.find_user_by_baziness_user_id(baziness_user_id)
    me = db.ensure_user_by_device(body.deviceId)
    if owner and owner.id != me.id:
        return {"ok": True, "switchToDeviceId": owner.device_id}
    db.link_baziness(me.id, baziness_user_id, True)
    return {"ok": True}
```

Giving Gita coins

A payout that can never be paid twice: the grant key fixes the `transactionId`, and a `400` means it was already paid.

TypeScript (Node)

```
// grants.ts — a payout that can never be paid twice.
import { addCoin } from "./platform/baziness";

export async function payTournamentReward(claimId: number, bazinessUserId: number, amount:
number) {
  try {
    // Same claim → same key → same transactionId → the platform refuses a second credit.
    await addCoin(bazinessUserId, amount, `reward_claim:${claimId}`, { source: "tournament",
```

```
claimId });
  } catch (err) {
    // 400 = this transactionId was already credited (e.g. we crashed after the platform paid
but
    // before we saved): the coins are there, so carry on instead of failing forever.
    if ((err as { status?: number }).status !== 400) throw err;
  }
  await db.claims.markPaid(claimId);                        // your own bookkeeping
  await db.ledger.insert({ claimId, currency: "GITA_COIN", amount, external: true });
}
```

C# (.NET)

```
// Grants.cs — a payout that can never be paid twice.
public sealed class Grants(BazinessPlatformClient platform, AppDb db)
{
    public async Task PayTournamentRewardAsync(long claimId, long bazinessUserId, long
amount)
    {
        try
        {
            // Same claim → same key → same transactionId → the platform refuses a second
credit.
            await platform.AddCoinAsync(bazinessUserId, amount, $"reward_claim:{claimId}",
new { source = "tournament", claimId });
        }
        catch (HttpRequestException e) when (e.StatusCode ==
System.Net.HttpStatusCode.BadRequest)
        {
            // 400 = already credited by an earlier attempt: the coins are there, carry on.
        }
        await db.Claims.MarkPaidAsync(claimId);               // your own bookkeeping
        await db.Ledger.InsertAsync(new LedgerRow(claimId, "GITA_COIN", amount, External:
true));
    }
}
```

Python

```
# grants.py — a payout that can never be paid twice.
import requests
from baziness.platform import platform


def pay_tournament_reward(claim_id: int, baziness_user_id: int, amount: int) -> None:
    try:
        # Same claim → same key → same transactionId → the platform refuses a second credit.
        platform.add_coin(baziness_user_id, amount, f"reward_claim:{claim_id}", {"source":
"tournament", "claimId": claim_id})
    except requests.HTTPError as err:
        if err.response is None or err.response.status_code != 400:
            raise
        # 400 = already credited by an earlier attempt: the coins are there, carry on.
    db.claims.mark_paid(claim_id)  # your own bookkeeping
    db.ledger.insert(claim_id=claim_id, currency="GITA_COIN", amount=amount, external=True)
```

Claims, held coins and moving them on sign-in

The rules from Gita coins → players who haven't signed in: a claim requires a signed-in player, a paid purchase never fails, and held coins move to Baziness when the player signs in.

TypeScript (Node)

```
// grants.ts: Gita coins go to the platform; COIN/GEM stay local.
export async function grantCurrency(tx: Tx, change: WalletChange & { idempotencyKey: string
},
                                    opts: { requireLink: boolean; metaData?: object }) {
  if (change.currency !== "GITA_COIN" || !platformEnabled() || change.amount < 0) {
    return applyWalletChange(tx, change);                       // local wallet
  }
  const { baziness_user_id: bazinessUserId } = await tx.one("select baziness_user_id from
users where id = $1", [change.userId]);
  if (!bazinessUserId) {
    if (opts.requireLink) throw new ApiError("PLATFORM_NOT_LINKED", "برای دریافت گیتا کوین
وارد حساب بازینس شوید");
    return applyWalletChange(tx, { ...change, note: "held until a Baziness account is linked"
}); // escrow
  }
  try {
    await addCoin(bazinessUserId, change.amount, change.idempotencyKey, { source:
change.type, ...opts.metaData });
  } catch (err) {
    throw new ApiError("PLATFORM_UNAVAILABLE", "ارتباط با بازینس برقرار نشد، دوباره تلاش کنید");
  }
  await recordExternalCredit(tx, change);                       // ledger row, balance
unchanged locally
}

/** Called right after a successful link (POST /api/platform/login or /token). */
export async function flushGitaEscrow(userId: number) {
  await withTx(async (tx) => {
    const user = await tx.one("select baziness_user_id from users where id = $1", [userId]);
    const wallet = await tx.oneOrNone(
      "select balance from user_wallets where user_id = $1 and currency = 'GITA_COIN' for
update", [userId]);
    if (!user.baziness_user_id || !wallet || wallet.balance <= 0) return;
    const last = await tx.one("select max(id) as id from wallet_transactions where user_id =
$1 and currency = 'GITA_COIN'", [userId]);
    const key = `escrow:${userId}:${last.id ?? 0}`;
    await addCoin(user.baziness_user_id, wallet.balance, key, { source: "escrow" });
    await applyWalletChange(tx, { userId, currency: "GITA_COIN", amount: -wallet.balance,
      type: "REWARD_SELECTION", idempotencyKey: key, note: "moved to the Baziness account"
});
  });
}
```

C# (.NET)

```
// Inside a service with: BazinessPlatformClient platform, PlatformSettings settings,
IUserStore users, IWalletStore wallets.
public async Task GrantAsync(DbTransaction tx, WalletChange change, bool requireLink)
{
    if (change.Currency != "GITA_COIN" || !settings.IsOn || change.Amount < 0)
    { await wallets.ApplyAsync(tx, change); return; }                       // local wallet
```

```
    long? bazinessUserId = await users.GetBazinessUserIdAsync(tx, change.UserId);
    if (bazinessUserId is null)
    {
        if (requireLink) throw new ApiException("PLATFORM_NOT_LINKED", "برای دریافت گیتا کوین
وارد حساب بازینس شوید");
        await wallets.ApplyAsync(tx, change with { Note = "held until a Baziness account is
linked" }); // escrow
        return;
    }
    await platform.AddCoinAsync(bazinessUserId.Value, change.Amount, change.IdempotencyKey,
new { source = change.Type });
    await wallets.RecordExternalCreditAsync(tx, change);
}

public async Task FlushEscrowAsync(long userId)
{
    await using var tx = await db.BeginTransactionAsync();
    var bazinessUserId = await users.GetBazinessUserIdAsync(tx, userId);
    var balance = await wallets.LockBalanceAsync(tx, userId, "GITA_COIN");   // SELECT … FOR
UPDATE
    if (bazinessUserId is null || balance <= 0) return;
    var key = $"escrow:{userId}:{await wallets.LastTransactionIdAsync(tx, userId,
"GITA_COIN")}";
    await platform.AddCoinAsync(bazinessUserId.Value, balance, key, new { source = "escrow"
});
    await wallets.ApplyAsync(tx, new WalletChange(userId, "GITA_COIN", -balance,
"REWARD_SELECTION", key));
    await tx.CommitAsync();
}
```

Python

```
def grant_currency(tx, change: dict, require_link: bool, meta: dict | None = None) -> None:
    if change["currency"] != "GITA_COIN" or not platform_enabled() or change["amount"] < 0:
        return apply_wallet_change(tx, change)                        # local wallet
    baziness_user_id = tx.scalar("select baziness_user_id from users where id = %s",
[change["userId"]])
    if not baziness_user_id:
        if require_link:
            raise ApiError("PLATFORM_NOT_LINKED", "برای دریافت گیتا کوین وارد حساب بازینس
شوید")
        return apply_wallet_change(tx, {**change, "note": "held until a Baziness account is
linked"})  # escrow
    platform.add_coin(baziness_user_id, change["amount"], change["idempotencyKey"],
{"source": change["type"], **(meta or {})})
    record_external_credit(tx, change)


def flush_gita_escrow(user_id: int) -> None:
    with db.transaction() as tx:
        baziness_user_id = tx.scalar("select baziness_user_id from users where id = %s",
[user_id])
        balance = tx.scalar("select balance from user_wallets where user_id = %s and currency
= 'GITA_COIN' "
                            "for update", [user_id]) or 0
        if not baziness_user_id or balance <= 0:
            return
        last = tx.scalar("select max(id) from wallet_transactions where user_id = %s and
```

```
currency = 'GITA_COIN'",
                          [user_id]) or 0
        key = f"escrow:{user_id}:{last}"
        platform.add_coin(baziness_user_id, balance, key, {"source": "escrow"})
        apply_wallet_change(tx, {"userId": user_id, "currency": "GITA_COIN", "amount": -
balance,
                                 "type": "REWARD_SELECTION", "idempotencyKey": key})
```

Baziness Pay client

Steps 1–5 of Baziness Pay: the payment page URL, `get_payment`, `verify_payment` (a **form** body) and `consume`.

TypeScript (Node)

```
// platform/baziness-pay.ts
const BASE = (process.env.BAZINESS_PAY_BASE_URL ??
"https://pay.baziness.com").replace(/\/+$/, "");
const PNAME = process.env.BAZINESS_PAY_PACKAGE_NAME ?? "";
const PREFIX = process.env.BAZINESS_SKU_PREFIX ?? "";
export const payEnabled = () => process.env.BAZINESS_PAY_ENABLED === "true";

export const skuFor = (productSku: string, override?: string | null) => override ||
`${PREFIX}${productSku}`;

export const buildBuyUrl = (uid: string, sku: string) =>
  `${BASE}/buy?${new URLSearchParams({ sku, uid, pname: PNAME })}`;

/** The verification token of a completed, unconsumed payment — or null while unpaid. */
export async function getPayment(uid: string, sku: string): Promise<{ token: string; sku:
string } | null> {
  try {
    const res = await fetch(`${BASE}/get_payment`, {
      method: "POST", headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ uid, sku, pname: PNAME }),
    });
    if (!res.ok) return null;
    const j = (await res.json()) as { success?: boolean; token?: string; sku?: string };
    return j.success && j.token ? { token: j.token, sku: j.sku ?? sku } : null;
  } catch { return null; }
}

/** The authoritative check. Never trust a token from the client. */
export async function verifyPayment(uid: string, token: string, sku: string):
Promise<boolean> {
  try {
    const res = await fetch(`${BASE}/verify_payment`, {
      method: "POST", headers: { "Content-Type": "application/x-www-form-urlencoded" },   //
form, not JSON
      body: new URLSearchParams({ uid, token, sku, pname: PNAME }).toString(),
    });
    if (!res.ok) return false;
    const j = (await res.json()) as { success?: boolean; error?: string };
    if (!j.success && j.error) console.warn("[pay] verify_payment rejected:", j.error);
```

```
    return j.success === true;
  } catch { return false; }
}

/** Let the SKU be bought again. Best effort, 3 attempts; never blocks an already-granted
purchase. */
export async function consumePayment(uid: string, sku: string): Promise<boolean> {
  for (let attempt = 1; attempt <= 3; attempt++) {
    try {
      const res = await fetch(`${BASE}/consume`, {
        method: "POST", headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ uid, sku, pname: PNAME }),
      });
      if (res.ok && ((await res.json()) as { success?: boolean }).success) return true;
    } catch { /* retry */ }
  }
  console.error("[pay] consume failed uid=%s sku=%s", uid, sku);
  return false;
}
```

C# (.NET)

```
// BazinessPay.cs — register with builder.Services.AddHttpClient<BazinessPay>();
using System.Net.Http.Json;

public sealed class BazinessPay(HttpClient http, IConfiguration cfg, ILogger<BazinessPay>
log)
{
    string Base => (cfg["BazinessPay:BaseUrl"] ?? "https://pay.baziness.com").TrimEnd('/');
    string PName => cfg["BazinessPay:PackageName"]!;
    public bool Enabled => cfg.GetValue<bool>("BazinessPay:Enabled");
    public string SkuFor(string productSku, string? overrideSku = null)
        => string.IsNullOrEmpty(overrideSku) ? $"{cfg["BazinessPay:SkuPrefix"]}{productSku}"
: overrideSku;

    public string BuildBuyUrl(string uid, string sku) =>
        $"{Base}/buy?sku={Uri.EscapeDataString(sku)}&uid={Uri.EscapeDataString(uid)}&pname=
{Uri.EscapeDataString(PName)}";

    record PaymentDto(bool Success, string? Token, string? Sku);
    record ResultDto(bool Success, string? Error);

    public async Task<(string Token, string Sku)?> GetPaymentAsync(string uid, string sku)
    {
        try
        {
            var res = await http.PostAsJsonAsync($"{Base}/get_payment", new { uid, sku, pname
= PName });
            if (!res.IsSuccessStatusCode) return null;
            var j = await res.Content.ReadFromJsonAsync<PaymentDto>();
            return j is { Success: true, Token: { Length: > 0 } t } ? (t, j.Sku ?? sku) :
null;
        }
        catch (Exception ex) { log.LogError(ex, "get_payment failed"); return null; }
    }

    public async Task<bool> VerifyPaymentAsync(string uid, string token, string sku)
    {
        try
```

```
        {
            var form = new FormUrlEncodedContent(new Dictionary<string, string>   // form,
not JSON
                { ["uid"] = uid, ["token"] = token, ["sku"] = sku, ["pname"] = PName });
            var res = await http.PostAsync($"{Base}/verify_payment", form);
            if (!res.IsSuccessStatusCode) return false;
            var j = await res.Content.ReadFromJsonAsync<ResultDto>();
            if (j is { Success: false, Error: not null }) log.LogWarning("verify_payment
rejected: {Error}", j.Error);
            return j?.Success == true;
        }
        catch (Exception ex) { log.LogError(ex, "verify_payment failed"); return false; }
    }

    public async Task<bool> ConsumeAsync(string uid, string sku)
    {
        for (var attempt = 1; attempt <= 3; attempt++)
        {
            try
            {
                var res = await http.PostAsJsonAsync($"{Base}/consume", new { uid, sku, pname
= PName });
                if (res.IsSuccessStatusCode && (await
res.Content.ReadFromJsonAsync<ResultDto>())?.Success == true) return true;
            }
            catch (Exception ex) { log.LogWarning(ex, "consume attempt {Attempt} failed",
attempt); }
        }
        log.LogError("consume failed uid={Uid} sku={Sku}", uid, sku);
        return false;
    }
}
```

Python

```
# baziness/pay.py
import os
import requests
from urllib.parse import urlencode

BASE = os.getenv("BAZINESS_PAY_BASE_URL", "https://pay.baziness.com").rstrip("/")
PNAME = os.getenv("BAZINESS_PAY_PACKAGE_NAME", "")
PREFIX = os.getenv("BAZINESS_SKU_PREFIX", "")


def pay_enabled() -> bool:
    return os.getenv("BAZINESS_PAY_ENABLED", "false").lower() == "true"


def sku_for(product_sku: str, override: str | None = None) -> str:
    return override or f"{PREFIX}{product_sku}"


def build_buy_url(uid: str, sku: str) -> str:
    return f"{BASE}/buy?{urlencode({'sku': sku, 'uid': uid, 'pname': PNAME})}"


def get_payment(uid: str, sku: str) -> dict | None:
    try:
```

```
        r = requests.post(f"{BASE}/get_payment", json={"uid": uid, "sku": sku, "pname":
PNAME}, timeout=10)
        j = r.json() if r.ok else {}
        return {"token": j["token"], "sku": j.get("sku", sku)} if j.get("success") and
j.get("token") else None
    except requests.RequestException:
        return None


def verify_payment(uid: str, token: str, sku: str) -> bool:
    try:  # form-encoded, not JSON
        r = requests.post(f"{BASE}/verify_payment", data={"uid": uid, "token": token, "sku":
sku, "pname": PNAME},
                          timeout=10)
        return r.ok and r.json().get("success") is True
    except requests.RequestException:
        return False


def consume_payment(uid: str, sku: str) -> bool:
    for _ in range(3):
        try:
            r = requests.post(f"{BASE}/consume", json={"uid": uid, "sku": sku, "pname":
PNAME}, timeout=10)
            if r.ok and r.json().get("success"):
                return True
        except requests.RequestException:
            pass
    return False
```

The payments table

Schema (PostgreSQL)

```
create table payments (
  id                       bigserial primary key,
  user_id                  bigint not null references users(id),
  shop_product_id          bigint not null references shop_products(id),
  provider                 text not null default 'BAZINESS',
  provider_sku             text not null,
  -- The Baziness payment token; unique so the same payment can never be fulfilled twice.
  provider_transaction_id  text,
  amount                   bigint not null,
  currency_code            text not null,
  status                   text not null default 'CREATED' check (status in

('CREATED','REDIRECTED','PAID','VERIFIED','FAILED','CANCELLED','FULFILLED')),
  reward_snapshot          jsonb not null,          -- what the player was promised when they
clicked
  failure_reason           text,
  created_at               timestamptz not null default now(),
  redirected_at            timestamptz,
  verified_at              timestamptz,
  fulfilled_at             timestamptz,
  updated_at               timestamptz not null default now(),
```

```
  unique (provider, provider_transaction_id)
);
create index payments_user on payments(user_id, created_at desc);
```

Purchase endpoints

`POST /api/shop/purchase/start` and `POST /api/shop/purchase/verify`: save the order, then verify and give the item exactly once.

TypeScript (Node)

```
// services/shop.ts
export async function startPurchase(userId: number, productId: number) {
  if (!payEnabled()) throw new ApiError("PAYMENT_DISABLED", "پرداخت به‌زودی فعال می‌شود");
  const product = await db.one("select * from shop_products where id = $1 and enabled",
[productId]);
  const sku = skuFor(product.sku, product.baziness_product_id);
  const payment = await db.one(
    `insert into payments (user_id, shop_product_id, provider_sku, amount, currency_code,
status, reward_snapshot, redirected_at)
     values ($1, $2, $3, $4, $5, 'REDIRECTED', $6, now()) returning id`,
    [userId, product.id, sku, product.price, product.currency_code,
JSON.stringify(productGrants(product))]);
  return { paymentId: String(payment.id), url: buildBuyUrl(String(userId), sku) };
}

export async function verifyPurchase(userId: number, paymentId: number) {
  const p = await db.oneOrNone("select * from payments where id = $1", [paymentId]);
  if (!p || p.user_id !== userId) throw notFound("payment");
  if (p.status === "FULFILLED" || p.status === "FAILED" || p.status === "CANCELLED") {
    return { status: p.status, wallets: await getBalances(userId), granted: [] };
  }
  const uid = String(userId);
  const found = await getPayment(uid, p.provider_sku);
  if (!found) return { status: p.status, wallets: await getBalances(userId), granted: [] };
// not paid yet
  if (!(await verifyPayment(uid, found.token, p.provider_sku))) {
    await db.none("update payments set failure_reason = 'verify_payment rejected' where id =
$1 and status <> 'FULFILLED'", [paymentId]);
    throw new ApiError("PAYMENT_FAILED", "پرداخت تأیید نشد");
  }
  const granted = await fulfilPayment(paymentId, found.token);
  void consumePayment(uid, p.provider_sku);                        // after commit, best
effort
  return { status: "FULFILLED", wallets: await getBalances(userId), granted };
}

export function fulfilPayment(paymentId: number, providerToken: string) {
  return withTx(async (tx) => {
    const p = await tx.one("select * from payments where id = $1 for update", [paymentId]);
    if (p.status === "FULFILLED") return [];
    const dup = await tx.oneOrNone(
      "select id from payments where provider = 'BAZINESS' and provider_transaction_id = $1
and id <> $2", [providerToken, paymentId]);
```

```
    if (dup) {
      await tx.none("update payments set status = 'FAILED', failure_reason = 'duplicate
provider token' where id = $1", [paymentId]);
      return [];
    }
    await tx.none("update payments set status = 'VERIFIED', provider_transaction_id = $2,
verified_at = now() where id = $1",
      [paymentId, providerToken]);
    for (const g of p.reward_snapshot) {
      await grantCurrency(tx, { userId: p.user_id, currency: g.currency, amount: g.amount,
type: "SHOP_PURCHASE",
        idempotencyKey: `payment:${paymentId}:${g.currency}` }, { requireLink: false }); //
Gita coins: escrow if not linked
    }
    await tx.none("update payments set status = 'FULFILLED', fulfilled_at = now() where id =
$1", [paymentId]);
    return p.reward_snapshot;
  });
}
```

C# (.NET)

```
app.MapPost("/api/shop/purchase/start", async (StartRequest body, ClaimsPrincipal me,
BazinessPay pay, ShopDb db) =>
{
    if (!pay.Enabled) return Results.Ok(new { error = "PAYMENT_DISABLED" });
    var userId = me.UserId();
    var product = await db.GetEnabledProductAsync(body.ProductId);
    if (product is null) return Results.NotFound();
    var sku = pay.SkuFor(product.Sku, product.BazinessProductId);
    var paymentId = await db.InsertPaymentAsync(userId, product, sku, status: "REDIRECTED");
    return Results.Ok(new { paymentId = paymentId.ToString(), url =
pay.BuildBuyUrl(userId.ToString(), sku) });
});

app.MapPost("/api/shop/purchase/verify", async (VerifyRequest body, ClaimsPrincipal me,
BazinessPay pay, ShopDb db) =>
{
    var userId = me.UserId();
    var p = await db.GetPaymentAsync(body.PaymentId);
    if (p is null || p.UserId != userId) return Results.NotFound();
    if (p.Status is "FULFILLED" or "FAILED" or "CANCELLED")
        return Results.Ok(new { status = p.Status, wallets = await db.BalancesAsync(userId),
granted = Array.Empty<object>() });

    var uid = userId.ToString();
    var found = await pay.GetPaymentAsync(uid, p.ProviderSku);
    if (found is null)                                          // not paid yet — keep
waiting
        return Results.Ok(new { status = p.Status, wallets = await db.BalancesAsync(userId),
granted = Array.Empty<object>() });
    if (!await pay.VerifyPaymentAsync(uid, found.Value.Token, p.ProviderSku))
    {
        await db.SetFailureReasonAsync(p.Id, "verify_payment rejected");
        return Results.Ok(new { error = "PAYMENT_FAILED" });
    }

    var granted = await db.FulfilAsync(p.Id, found.Value.Token);   // one tx: FOR UPDATE,
dup-token check, grants, FULFILLED
```

```
    _ = pay.ConsumeAsync(uid, p.ProviderSku);                    // after commit, best
effort
    return Results.Ok(new { status = "FULFILLED", wallets = await db.BalancesAsync(userId),
granted });
});

public record StartRequest(long ProductId);
public record VerifyRequest(long PaymentId);
```

Python

```
from fastapi import Depends, HTTPException
from baziness import pay


@app.post("/api/shop/purchase/start")
def start_purchase(body: dict, user=Depends(current_user)):
    if not pay.pay_enabled():
        raise HTTPException(400, "PAYMENT_DISABLED")
    product = db.fetch_one("select * from shop_products where id = %s and enabled",
[body["productId"]])
    if not product:
        raise HTTPException(404)
    sku = pay.sku_for(product["sku"], product["baziness_product_id"])
    payment_id = db.scalar(
        """insert into payments (user_id, shop_product_id, provider_sku, amount,
currency_code, status,
                                 reward_snapshot, redirected_at)
           values (%s, %s, %s, %s, %s, 'REDIRECTED', %s, now()) returning id""",
        [user.id, product["id"], sku, product["price"], product["currency_code"],
json.dumps(product_grants(product))])
    return {"paymentId": str(payment_id), "url": pay.build_buy_url(str(user.id), sku)}


@app.post("/api/shop/purchase/verify")
def verify_purchase(body: dict, user=Depends(current_user)):
    p = db.fetch_one("select * from payments where id = %s", [body["paymentId"]])
    if not p or p["user_id"] != user.id:
        raise HTTPException(404)
    if p["status"] in ("FULFILLED", "FAILED", "CANCELLED"):
        return {"status": p["status"], "wallets": balances(user.id), "granted": []}
    uid = str(user.id)
    found = pay.get_payment(uid, p["provider_sku"])
    if not found:  # not paid yet
        return {"status": p["status"], "wallets": balances(user.id), "granted": []}
    if not pay.verify_payment(uid, found["token"], p["provider_sku"]):
        db.execute("update payments set failure_reason = 'verify_payment rejected' where id =
%s "
                   "and status <> 'FULFILLED'", [p["id"]])
        raise HTTPException(400, "PAYMENT_FAILED")
    granted = fulfil_payment(p["id"], found["token"])   # one tx: FOR UPDATE, dup-token
check, grants, FULFILLED
    threading.Thread(target=pay.consume_payment, args=(uid, p["provider_sku"]),
daemon=True).start()
    return {"status": "FULFILLED", "wallets": balances(user.id), "granted": granted}
```

Local fake gateway

The example game includes a stand-in for Baziness Pay that speaks the same protocol, for working without gateway access. It is switched on with these settings, and never in production:

.env (local only)

```
DEV_FAKE_BAZINESS_PAY=true
BAZINESS_PAY_ENABLED=true
BAZINESS_PAY_BASE_URL=http://localhost:3390/dev/baziness
```

First-touch attribution

The rules from Links & UTM. The first tab is the **client** side, reading the tags and removing them from the address. The others save them on the **server** when the player is created.

TypeScript (client)

```
// api/client.ts — the FIRST module the app imports.
const UTM_KEY = "mygame_utm";
const TAG_KEY = "mygame_tag";
const UTM_FIELDS = ["source", "medium", "campaign", "content"] as const;
type Utm = Partial<Record<(typeof UTM_FIELDS)[number], string>>;

const clean = (raw: unknown): string | null => {
  if (typeof raw !== "string") return null;
  const v = raw.trim().toLowerCase().replace(/\s+/g, "_").slice(0, 64);
  return /^[a-z0-9][a-z0-9._-]*$/.test(v) ? v : null;      // ASCII only
};

export const FIRST_TOUCH_UTM: Utm = (() => {
  let stored: Utm = {};
  try { stored = JSON.parse(localStorage.getItem(UTM_KEY) ?? "{}"); } catch {}
  try {
    const query = new URLSearchParams(location.search);
    const keys = [...query.keys()].filter((k) => k.toLowerCase().startsWith("utm_"));
    const fromUrl: Utm = {};
    for (const k of keys) {
      const field = k.toLowerCase().slice(4) as keyof Utm;
      const v = clean(query.get(k));
      if (UTM_FIELDS.includes(field as never) && v) fromUrl[field] = v;
    }
    if (keys.length) {                                      // take them out of the address
      keys.forEach((k) => query.delete(k));
      const search = query.toString();
      history.replaceState(history.state, "", `${location.pathname}${search ? `?${search}` :
""}${location.hash}`);
    }
    if (Object.keys(fromUrl).length && !Object.keys(stored).length) {   // first touch wins
      stored = fromUrl;
      localStorage.setItem(UTM_KEY, JSON.stringify(stored));
```

```
    }
  } catch { /* storage blocked: the visit is simply untagged */ }
  return stored;
})();

export function firstTouchTag(): string | null {
  try {
    const fromUrl = clean(new URLSearchParams(location.search).get("tag"));
    if (fromUrl && !localStorage.getItem(TAG_KEY)) localStorage.setItem(TAG_KEY, fromUrl);
    return localStorage.getItem(TAG_KEY) ?? fromUrl;
  } catch { return null; }
}

// Sent with the device login AND with /api/platform/token:
//   { deviceId, tag: firstTouchTag(), utm: FIRST_TOUCH_UTM }
```

C# (server)

```
using System.Text.RegularExpressions;

// Save attribution ONLY when the account is created (first touch). Four separate columns.
public async Task<User> EnsureUserByDeviceAsync(string deviceId, string? tag, UtmParams? utm)
{
    const string sql = """
        insert into users (device_id, acq_tag, utm_source, utm_medium, utm_campaign,
utm_content)
        values (@deviceId, @tag, @source, @medium, @campaign, @content)
        on conflict (device_id) do update set last_seen = now()   -- existing account:
attribution untouched
        returning *;
        """;
    return await db.QuerySingleAsync<User>(sql, new {
        deviceId, tag = Utm.Clean(tag),
        source = Utm.Clean(utm?.Source), medium = Utm.Clean(utm?.Medium),
        campaign = Utm.Clean(utm?.Campaign), content = Utm.Clean(utm?.Content),
    });
}

public static class Utm
{
    static readonly Regex Valid = new("^[a-z0-9][a-z0-9._-]*$", RegexOptions.Compiled);
    public static string? Clean(string? raw)
    {
        if (raw is null) return null;
        var v = Regex.Replace(raw.Trim().ToLowerInvariant(), @"\s+", "_");
        if (v.Length > 64) v = v[..64];
        return Valid.IsMatch(v) ? v : null;
    }
}
public record UtmParams(string? Source, string? Medium, string? Campaign, string? Content);
```

Python (server)

```
import re

_VALID = re.compile(r"^[a-z0-9][a-z0-9._-]*$")


def clean_utm(raw: str | None) -> str | None:
```

```python
    if not isinstance(raw, str):
        return None
    v = re.sub(r"\s+", "_", raw.strip().lower())[:64]
    return v if _VALID.match(v) else None


def ensure_user_by_device(device_id: str, tag: str | None, utm: dict | None):
    """Attribution is saved ONLY when the account is created (first touch)."""
    utm = utm or {}
    return db.fetch_one(
        """
        insert into users (device_id, acq_tag, utm_source, utm_medium, utm_campaign,
utm_content)
        values (%s, %s, %s, %s, %s, %s)
        on conflict (device_id) do update set last_seen = now()
        returning *
        """,
        [device_id, clean_utm(tag), *(clean_utm(utm.get(k) or utm.get(f"utm_{k}"))
                                      for k in ("source", "medium", "campaign", "content"))],
    )
```

Schema (PostgreSQL)

```sql
alter table users
  add column if not exists acq_tag      text,
  add column if not exists utm_source   text,
  add column if not exists utm_medium   text,
  add column if not exists utm_campaign text,
  add column if not exists utm_content  text;

create index if not exists idx_users_utm_combo on users (utm_source, utm_medium,
utm_campaign, utm_content)
  where utm_source is not null or utm_medium is not null or utm_campaign is not null or
utm_content is not null;
```

Building links

Baziness page links (parameters):

TypeScript

```typescript
type PageOptions = { redirect?: "shop" | "profile"; allowGuest?: boolean; gameName?: string;
buildStore?: string };

export function bazinessPageUrl(webBase: string, path: string, packageName: string, token:
string | null, o: PageOptions = {}) {
  const u = new URL(webBase.replace(/\/+$/, "") + (path || "/"));
  if (o.gameName) u.searchParams.set("game_name", o.gameName);
  u.searchParams.set("packageName", packageName);
  u.searchParams.set("sdkType", "js");                      // never optional
  if (token) u.searchParams.set("linkedToken", token);
  if (o.redirect) u.searchParams.set("redirect", o.redirect);
```

```
  if (o.allowGuest === false) u.searchParams.set("allowGuest", "false");
  if (o.buildStore) u.searchParams.set("buildStore", o.buildStore);
  return u.toString();
}

// bazinessPageUrl(webBase, "/shop", pkg, token, { gameName: "example", redirect: "shop" })
// bazinessPageUrl(webBase, "/", pkg, token, { allowGuest: false })
```

C#

```
using System.Web; // or Microsoft.AspNetCore.WebUtilities.QueryHelpers

public static string BazinessPageUrl(string webBase, string path, string packageName, string?
token,
    string? redirect = null, bool allowGuest = true, string? gameName = null, string?
buildStore = null)
{
    var q = HttpUtility.ParseQueryString(string.Empty);     // encodes values for you
    if (gameName != null) q["game_name"] = gameName;
    q["packageName"] = packageName;
    q["sdkType"] = "js";                                    // never optional
    if (!string.IsNullOrEmpty(token)) q["linkedToken"] = token;
    if (redirect != null) q["redirect"] = redirect;
    if (!allowGuest) q["allowGuest"] = "false";
    if (buildStore != null) q["buildStore"] = buildStore;
    return $"{webBase.TrimEnd('/')}{(string.IsNullOrEmpty(path) ? "/" : path)}?{q}";
}

// BazinessPageUrl(webBase, "/shop", pkg, token, redirect: "shop", gameName: "example")
```

Python

```
from urllib.parse import urlencode


def baziness_page_url(web_base: str, path: str, package_name: str, token: str | None, *,
                      redirect: str | None = None, allow_guest: bool = True,
                      game_name: str | None = None, build_store: str | None = None) -> str:
    params: dict[str, str] = {}
    if game_name:
        params["game_name"] = game_name
    params["packageName"] = package_name
    params["sdkType"] = "js"                                # never optional
    if token:
        params["linkedToken"] = token
    if redirect:
        params["redirect"] = redirect
    if not allow_guest:
        params["allowGuest"] = "false"
    if build_store:
        params["buildStore"] = build_store
    return f"{web_base.rstrip('/')}{path or '/'}?{urlencode(params)}"

# baziness_page_url(web_base, "/shop", pkg, token, game_name="example", redirect="shop")
```

Campaign links for marketing:

TypeScript

```typescript
export function campaignLink(gameUrl: string, utm: { source: string; medium: string;
campaign: string; content?: string }) {
  const u = new URL(gameUrl);
  u.searchParams.set("utm_source", utm.source);
  u.searchParams.set("utm_medium", utm.medium);
  u.searchParams.set("utm_campaign", utm.campaign);
  if (utm.content) u.searchParams.set("utm_content", utm.content);
  return u.toString();
}
// campaignLink("https://yourgame.example.com/", { source: "baziness", medium: "inapp",
campaign: "shahrivar_1405", content: "banner_home" })
```

C#

```csharp
public static string CampaignLink(string gameUrl, string source, string medium, string
campaign, string? content = null)
{
    var q = System.Web.HttpUtility.ParseQueryString(string.Empty);
    q["utm_source"] = source; q["utm_medium"] = medium; q["utm_campaign"] = campaign;
    if (content != null) q["utm_content"] = content;
    return $"{gameUrl.TrimEnd('/')}/?{q}";
}
```

Python

```python
from urllib.parse import urlencode


def campaign_link(game_url: str, source: str, medium: str, campaign: str, content: str | None
= None) -> str:
    params = {"utm_source": source, "utm_medium": medium, "utm_campaign": campaign}
    if content:
        params["utm_content"] = content
    return f"{game_url.rstrip('/')}/?{urlencode(params)}"
```