

Migrating to link codes & short-lived tokens

The platform is replacing long-lived user tokens with 15-minute tokens and one-time link codes. What changes for your server and your client, step by step, and by when.

What is changing and why

Until now, a Baziness **user token** was valid for **10 years**. It also travelled in addresses:

- your game opened Baziness pages with `?linkedToken=<token>` ;
- the Baziness app opened your game with `?linkedToken=<token>` .

Addresses end up in server logs, analytics, browser history and Referer headers. A token found there gave access to the player's coins and account for a decade, and signing out couldn't stop it.

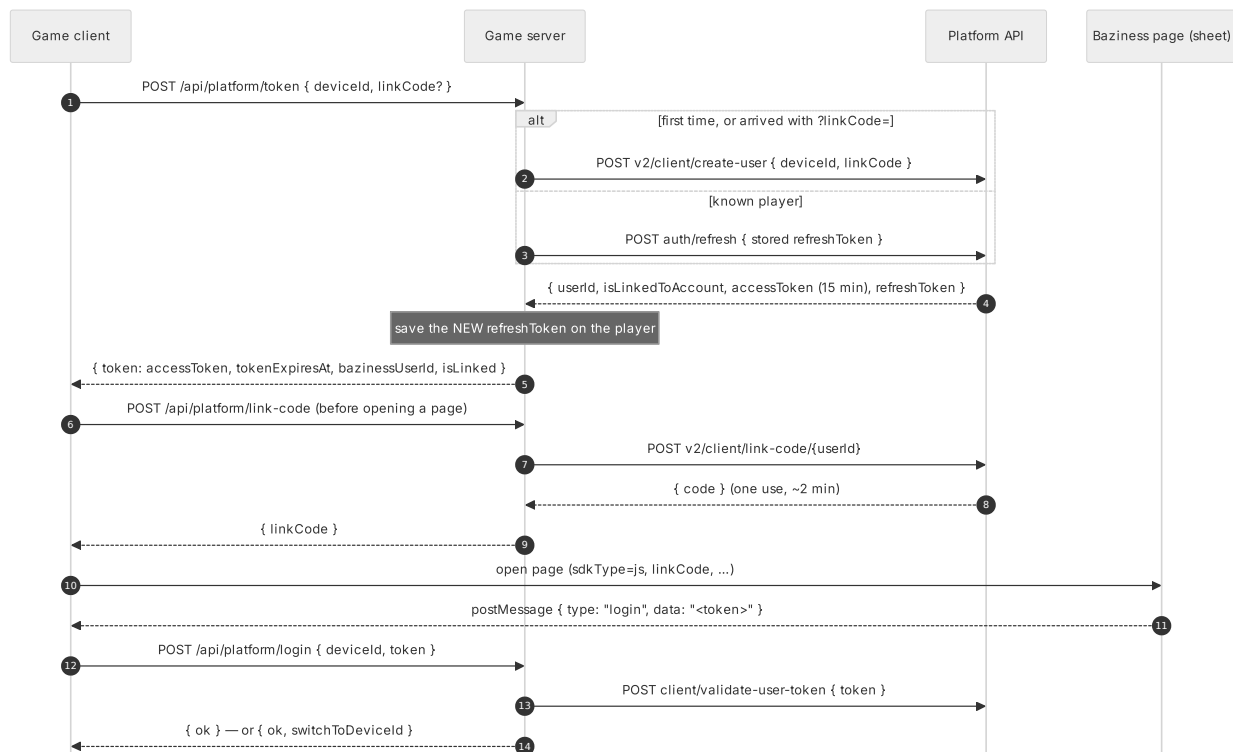
The new model:

	Before	Now
Access token (what your client sends as <code>Bearer</code>)	valid 10 years	valid 15 minutes
Refresh token	—	renews the access token; kept on your server , never on the device; single use, so every renewal returns a new one
Moving a player between apps	<code>?linkedToken=<token></code>	<code>?linkCode=<code></code> : a one-time code , valid about 2 minutes , useless after first use

Important

Both models work side by side during the migration. The old endpoints and `?linkedToken=` keep working, and the Baziness app sends both `?linkCode=` and `?linkedToken=` to games. The platform team will announce a **cut-off date**: after it, old tokens and `linkedToken` stop working. Plan the migration before then.

The flow after the migration:



What changes at a glance

Today	After the migration	Where
<code>client/create-user</code> with <code>linkedToken</code>	<code>v2/client/create-user</code> with <code>linkCode</code> ; save <code>tokens.refreshToken</code>	server
<code>client/get-user-token/{id}</code> on every boot	<code>auth/refresh</code> with the saved refresh token; <code>v2/client/get-user-token/{id}</code> only when there is no session	server
open pages with <code>linkedToken=<token></code>	open pages with <code>linkCode=<code></code> , a new code from <code>v2/client/link-code/{id}</code> for each page	server + client
the Baziness app opens your game with <code>?linkedToken=</code>	<code>?linkCode=</code> : forward it to your server like the token before	client
<code>GET client/validate-user-token/{token}</code>	<code>POST client/validate-user-token { token }</code> (the token is no longer in the address)	server
the client keeps its token for years	the client keeps the access token only; when it expires, it asks your server again	client

Nothing else changes: the page messages (`login` , `token` , `close` , `logout`), coins, rewards and payments work as before.

Server, step by step

1. Keep a refresh token per player

Add a column to your player record, for example `baziness_refresh_token` . Treat it like a password:

- never log it;
- never send it to the client;
- encrypt it if you encrypt other secrets.

Always overwrite it with the newest one. Every `create-user` , `get-user-token` and `refresh` returns a new refresh token, and the previous one stops working.

2. `/api/platform/token` : refresh instead of `get-user-token`

This is the endpoint your client calls on boot. The client now sends `linkCode` instead of `linkedToken`.

1. **The client sent a `linkCode` , or the player has no Baziness user yet:** call `v2/client/create-user` with `deviceId` and `linkCode` . Then handle the returned `userId` as you do today:
 - The `userId` belongs to **another** of your players → answer `switchToDeviceId` (same person, another device).
 - The player **already has a different** `userId` → a different Baziness account arrived. Start it as its own player and answer `switchToDeviceId` (the old "clear local data and start again" rule).
 - Otherwise → save `userId` and the refresh token on this player.
 - If the code was **expired or already used** (`400 invalid_link_code`), continue as if no code was sent.
2. **A known player:** call `auth/refresh` with the saved refresh token, and save the new one. The answer also contains `isLinkedToAccount` , so you still learn when the player signed in.
 - `401 refresh_token_rotated` : another request of the same player refreshed a moment earlier. **Reload** the refresh token from your database and try once more.
 - Any other `401` (`session_revoked` , `refresh_token_expired` , `invalid_refresh_token` , `refresh_token_reused`): the session is over. Call `v2/client/get-user-token/{userId}` for a new one and save its refresh token.
 - `403 account_suspended` : the account is blocked.
3. **Answer the client** with the **access token** and its expiry: `{ token, tokenExpiresAt, bazinessUserId, isLinkedToAccount }` . Never include the refresh token.

Tip

Players you already have don't hold a refresh token yet. Step 2 handles them on its own: no saved refresh token → `v2/client/get-user-token` once, and from then on `auth/refresh` .

3. New `/api/platform/link-code` : a code for each page

When the client is about to open a Baziness page, it asks your server for a code. Your server calls `v2/client/link-code/{userId}` and returns `{ linkCode }` .

- A code works **once** and for about **two minutes**: get a new one for every page you open.
- It only works for Baziness pages. Even if it leaks, nobody can use it to log in to your game.
- Protect the endpoint like your other player endpoints: only the player themselves may ask for their code.

4. `/api/platform/login` : verify with POST

Nothing changes in the logic. Only the call does: `POST client/validate-user-token` with `{ token }` in the body.

The token in the `login` message is now a **15-minute** token. Verify it right away, as you already do.

If the token belongs to a different Baziness user than the player's current one, clear the player's saved refresh token; the next `/token` call starts a new session.

5. New `/api/platform/logout`

When a page sends `logout`, the client calls this before starting over. Your server ends the session (`auth/logout`) and clears the saved refresh token.

Server code

Additions to your platform client (next to `createUser` / `getUserAccessToken`):

TypeScript (Node)

```
// platform/baziness.ts - ADD these next to createUser / getUserAccessToken (they reuse
API_BASE and platformFetch).

/** A player's session: the access token goes to your client, the refresh token stays in your
database. */
export interface PlatformSession {
  userId: number;
  isLinkedToAccount: boolean;
  accessToken: string;
  accessTokenExpiresAt: string;
  refreshToken: string;
}

const readSession = (j: Record<string, any>): PlatformSession => {
  const t = j.tokens ?? j; // v2 create-user / get-user-token nest the pair in `tokens`;
  auth/refresh returns it flat
  return {
    userId: Number(j.userId),
    isLinkedToAccount: Boolean(j.isLinkedToAccount),
    accessToken: String(t.accessToken),
    accessTokenExpiresAt: String(t.accessTokenExpiresAt),
    refreshToken: String(t.refreshToken),
  };
};

/** First time, or when the client brought a ?linkCode=. Throws (status 400) for an expired
or used code. */
export const createUserSession = async (deviceId: string, platform: string, store: string,
linkCode: string | null) =>
  readSession(await platformFetch("POST", "/v2/client/create-user", { deviceId, platform,
store, linkCode }));
```

```

/** A new session for a player you know – only when you have no refresh token or the session
ended. */
export const startUserSession = async (businessUserId: number) =>
  readSession(await platformFetch("POST", `/v2/client/get-user-token/${businessUserId}`));

/**
 * Renews a session. Returns the new session, "rotated" when a parallel request refreshed
first
 * (reload the stored token and retry once), or null when the session is over (start a new
one).
 */
export async function refreshUserSession(refreshToken: string): Promise<PlatformSession |
"rotated" | null> {
  const res = await fetch(`${API_BASE}/auth/refresh`, { // no client token: the refresh
token is the credential
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ refreshToken }),
  });
  if (res.ok) return readSession(await res.json());
  const error = ((await res.json().catch(() => ({}))) as { error?: string }).error;
  if (res.status === 401 && error === "refresh_token_rotated") return "rotated";
  if (res.status === 403) throw Object.assign(new Error("account_suspended"), { status: 403
});
  if (res.status === 401) return null;
  throw new Error(`auth/refresh failed (${res.status})`); // 5xx / network: keep the old
session, try later
}

/** Ends a session (player signed out). Never throws. */
export async function endUserSession(refreshToken: string): Promise<void> {
  await fetch(`${API_BASE}/auth/logout`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ refreshToken }),
  }).catch(() => {});
}

/** One-time code for opening ONE Baziness page as this player (~2 minutes). */
export const createPageLinkCode = async (businessUserId: number) =>
  String((await platformFetch("POST", `/v2/client/link-code/${businessUserId}`)).code);

/** REPLACES the old GET version: the token now goes in the body, not in the URL. */
export async function validateUserAccessToken(token: string): Promise<number> {
  const j = await platformFetch("POST", "/client/validate-user-token", { token });
  if (!(j.isValid ?? j.IsValid)) throw new Error("invalid user token");
  return Number(j.userId ?? j.UserId ?? 0);
}

```

C# (.NET)

```

// BazinessPlatformClient.cs – ADD these members (they reuse settings, http, SendAsync and
Json).

public record PlatformSession(long UserId, bool IsLinkedToAccount, string AccessToken,
    DateTimeOffset AccessTokenExpiresAt, string RefreshToken);

public enum RefreshOutcome { Ok, Rotated, Ended }

```

```

static PlatformSession ReadSession(JsonElement j)
{
    // v2 create-user / get-user-token nest the pair in "tokens"; auth/refresh returns it
    flat.
    var t = j.TryGetProperty("tokens", out var nested) ? nested : j;
    return new PlatformSession(
        j.GetProperty("userId").GetInt64(),
        j.GetProperty("isLinkedToAccount").GetBoolean(),
        t.GetProperty("accessToken").GetString()!,
        t.GetProperty("accessTokenExpiresAt").GetDateTimeOffset(),
        t.GetProperty("refreshToken").GetString()!);
}

/// <summary>First time, or when the client brought a ?linkCode=. 400 = expired or used code.
</summary>
public async Task<PlatformSession> CreateUserSessionAsync(string deviceId, string platform,
string store, string? linkCode) =>
    ReadSession(await SendAsync(HttpMethod.Post, "/v2/client/create-user", new { deviceId,
platform, store, linkCode }));

/// <summary>A new session for a known player – only when there is no refresh token or the
session ended.</summary>
public async Task<PlatformSession> StartUserSessionAsync(long bazinessUserId) =>
    ReadSession(await SendAsync(HttpMethod.Post, $"/v2/client/get-user-
token/{bazinessUserId}"));

/// <summary>Renews a session. Rotated = a parallel request refreshed first (reload, retry
once); Ended = start a new one.</summary>
public async Task<(RefreshOutcome Outcome, PlatformSession? Session)>
RefreshUserSessionAsync(string refreshToken)
{
    // No client token here: the refresh token itself is the credential.
    var res = await http.PostAsJsonAsync($"{settings.ApiBase}/auth/refresh", new {
refreshToken }, Json);
    if (res.IsSuccessStatusCode)
        return (RefreshOutcome.Ok, ReadSession(await
res.Content.ReadFromJsonAsync<JsonElement>(Json)));

    string? error = null;
    var text = await res.Content.ReadAsStringAsync();
    if (!string.IsNullOrEmpty(text))
    {
        try { error = JsonDocument.Parse(text).RootElement.TryGetProperty("error", out var e)
? e.GetString() : null; }
        catch (JsonException) { /* not JSON */ }
    }
    return res.StatusCode switch
    {
        HttpStatusCode.Unauthorized when error == "refresh_token_rotated" =>
(RefreshOutcome.Rotated, null),
        HttpStatusCode.Unauthorized => (RefreshOutcome.Ended, null),
        HttpStatusCode.Forbidden => throw new InvalidOperationException("account_suspended"),
        _ => throw new HttpRequestException($"auth/refresh failed ({(int)res.StatusCode}")
    };
    // keep the old session
};
}

/// <summary>Ends a session (player signed out). Never throws.</summary>
public async Task EndUserSessionAsync(string refreshToken)

```

```

{
    try { await http.PostAsJsonAsync($"{settings.ApiBase}/auth/logout", new { refreshToken },
Json); }
    catch { /* best effort */ }
}

/// <summary>One-time code for opening ONE Baziness page as this player (~2 minutes).
</summary>
public async Task<string> CreatePageLinkCodeAsync(long bazinessUserId) =>
    (await SendAsync(HttpMethod.Post, $"/v2/client/link-
code/{bazinessUserId}")).GetProperty("code").GetString()!;

/// <summary>REPLACES the old GET version: the token now goes in the body, not in the URL.
</summary>
public async Task<long> ValidateUserAccessTokenAsync(string token)
{
    var j = await SendAsync(HttpMethod.Post, "/client/validate-user-token", new { token });
    if (!j.GetProperty("isValid").GetBoolean()) throw new InvalidOperationException("invalid
user token");
    return j.GetProperty("userId").GetInt64();
}

```

Python

```

# baziness/platform.py – ADD these methods to BazinessPlatform (they reuse API_BASE,
self._http and self._call).

@staticmethod
def _session(j: dict) -> dict:
    t = j.get("tokens") or j # v2 create-user / get-user-token nest the pair;
auth/refresh returns it flat
    return {"userId": int(j["userId"]), "isLinkedToAccount":
bool(j["isLinkedToAccount"]),
        "accessToken": t["accessToken"], "accessTokenExpiresAt":
t["accessTokenExpiresAt"],
        "refreshToken": t["refreshToken"]}

def create_user_session(self, device_id: str, platform: str, store: str, link_code: str |
None) -> dict:
    """First time, or when the client brought a ?linkCode=. HTTPError 400 = expired or
used code."""
    return self._session(self._call("POST", "/v2/client/create-user", {
        "deviceId": device_id, "platform": platform, "store": store, "linkCode":
link_code}))

def start_user_session(self, baziness_user_id: int) -> dict:
    """A new session for a known player – only when there is no refresh token or the
session ended."""
    return self._session(self._call("POST", f"/v2/client/get-user-
token/{baziness_user_id}"))

def refresh_user_session(self, refresh_token: str):
    """Returns the new session, "rotated" (reload the stored token, retry once) or None
(session over)."""
    r = self._http.post(f"{API_BASE}/auth/refresh", json={"refreshToken": refresh_token},
timeout=10)
    if r.ok:
        return self._session(r.json())
    error = (r.json() if r.content else {}).get("error")

```

```

    if r.status_code == 401:
        return "rotated" if error == "refresh_token_rotated" else None
    if r.status_code == 403:
        raise PermissionError("account_suspended")
    r.raise_for_status() # 5xx: keep the old session, try later

def end_user_session(self, refresh_token: str) -> None:
    """Ends a session (player signed out). Never raises."""
    try:
        self._http.post(f"{API_BASE}/auth/logout", json={"refreshToken": refresh_token},
            timeout=10)
    except requests.RequestException:
        pass

def create_page_link_code(self, baziness_user_id: int) -> str:
    """One-time code for opening ONE Baziness page as this player (~2 minutes)."""
    return str(self._call("POST", f"/v2/client/link-code/{baziness_user_id}")["code"])

def validate_user_access_token(self, token: str) -> int:
    """REPLACES the old GET version: the token now goes in the body, not in the URL."""
    j = self._call("POST", "/client/validate-user-token", {"token": token})
    if not _pick(j, "isValid", False):
        raise ValueError("invalid user token")
    return int(_pick(j, "userId", 0))

```

The four endpoints:

TypeScript (Node)

```

import {
    createUserSession, startUserSession, refreshUserSession, endUserSession,
    createPageLinkCode,
    validateUserAccessToken, platformClientConfig, platformEnabled, type PlatformSession,
} from "../platform/baziness";
import { db } from "../db"; // your own data layer. New column on the player:
baziness_refresh_token (a secret)

// What the client gets. The refresh token never leaves your server.
const forClient = (s: PlatformSession) => ({
    token: s.accessToken, tokenExpiresAt: s.accessTokenExpiresAt,
    bazinessUserId: s.userId, isLinkedToAccount: s.isLinkedToAccount,
});

async function keep(playerId: number, s: PlatformSession) {
    await db.setRefreshToken(playerId, s.refreshToken); // always the NEWEST one: the old one
    stops working
    return s;
}

/** A valid session for a player who already has a Baziness user. */
async function sessionFor(player: { id: number; bazinessUserId: number; bazinessRefreshToken:
string | null }) {
    if (player.bazinessRefreshToken) {
        let r = await refreshUserSession(player.bazinessRefreshToken);
        if (r === "rotated") { // another request of this player refreshed
            a moment ago
            const latest = await db.getRefreshToken(player.id);
            r = latest ? await refreshUserSession(latest) : null;
        }
    }
}

```

```

    if (r === "rotated") r = null;
  }
  if (r) return keep(player.id, r);
}
// No refresh token yet (players from before the migration) or the session ended.
return keep(player.id, await startUserSession(player.bazinessUserId));
}

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

  const config = { enabled: true, ...platformClientConfig() };
  const me = await db.ensureUserByDevice(deviceId, { utm: req.body.utm, tag: req.body.tag });
  try {
    if (linkCode || !me.bazinessUserId) {
      // First time, or arrived from the Baziness app with ?linkCode=: which Baziness user is
this?
      const s = await createUserSession(deviceId, "Web", "Direct", linkCode)
        .catch((e) => (linkCode && e.status === 400 ? null : Promise.reject(e))); // expired
or used code
      if (s) {
        const owner = await db.findUserByBazinessId(s.userId);
        if (owner && owner.id !== me.id) { // the same person, another
player of yours
          await keep(owner.id, s);
          return res.json({ ...config, switchToDeviceId: owner.deviceId });
        }
        if (me.bazinessUserId && me.bazinessUserId !== s.userId) {
          // A different Baziness account arrived on a device that already has a player:
          // it becomes its own player (the old "clear local data and start again" rule).
          const fresh = await db.createPlayerForBaziness(s.userId, s.isLinkedToAccount);
          await keep(fresh.id, s);
          return res.json({ ...config, switchToDeviceId: fresh.deviceId });
        }
        await db.linkBaziness(me.id, s.userId, s.isLinkedToAccount);
        await keep(me.id, s);
        return res.json({ ...config, ...forClient(s) });
      }
      if (!me.bazinessUserId) { // stale code and no Baziness
user yet
        const s2 = await createUserSession(deviceId, "Web", "Direct", null);
        await db.linkBaziness(me.id, s2.userId, s2.isLinkedToAccount);
        await keep(me.id, s2);
        return res.json({ ...config, ...forClient(s2) });
      }
    }
    const s = await sessionFor(me);
    if (s.isLinkedToAccount !== me.isBazinessLinked) await db.setLinked(me.id,
s.isLinkedToAccount);
    return res.json({ ...config, ...forClient(s) });
  } catch (err) {
    console.error("[platform] token failed:", (err as Error).message);
    const suspended = (err as { status?: number }).status === 403;
    return res.json({ ...config, error: suspended ? "account_suspended" :
"platform_unavailable" }); // never block the boot
  }
}

```

```

});

// Protect it like your other player endpoints: only the player themself may ask for their code.
app.post("/api/platform/link-code", async (req, res) => {
    const me = await db.findUserByDevice(String(req.body.deviceId ?? ""));
    if (!me?.bazinessUserId) return res.status(409).json({ error: "call /api/platform/token first" });
    try {
        return res.json({ linkCode: await createPageLinkCode(me.bazinessUserId) });
    } catch {
        return res.status(503).json({ error: "platform_unavailable" });
    }
});

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); // POST now; verify right away (15-min token)
        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 });
        if (me.bazinessUserId !== bazinessUserId) await db.setRefreshToken(me.id, null); // belonged to the old user
        await db.linkBaziness(me.id, bazinessUserId, true);
        return res.json({ ok: true }); // the client then calls /token again
    } catch {
        return res.json({ ok: false, reason: "invalid" });
    }
});

// The page sent `logout`: end the session before the client starts over as a new visitor.
app.post("/api/platform/logout", async (req, res) => {
    const me = await db.findUserByDevice(String(req.body.deviceId ?? ""));
    if (me?.bazinessRefreshToken) {
        await endUserSession(me.bazinessRefreshToken);
        await db.setRefreshToken(me.id, null);
    }
    return res.json({ ok: true });
});

```

C# (.NET)

```

// IUserStore gains: SetRefreshTokenAsync(id, token?), GetRefreshTokenAsync(id), FindByDeviceAsync(deviceId),
// CreatePlayerForBazinessAsync(bazinessUserId, isLinked). Store the refresh token like a password.

// What the client gets. The refresh token never leaves your server.
static object ForClient(PlatformSettings pub, PlatformSession s) => new
{
    enabled = true, pub.WebBase, pub.ApiBase, pub.PackageName,

```

```

        token = s.AccessToken, tokenExpiresAt = s.AccessTokenExpiresAt,
        bazinessUserId = s.UserId, isLinkedToAccount = s.IsLinkedToAccount
    };

    static async Task<PlatformSession> SessionForAsync(Player player, BazinessPlatformClient
platform, IUserStore users)
    {
        var refreshToken = player.BazinessRefreshToken;
        for (var attempt = 0; refreshToken is not null && attempt < 2; attempt++)
        {
            var (outcome, session) = await platform.RefreshUserSessionAsync(refreshToken);
            if (outcome == RefreshOutcome.Ok)
            {
                await users.SetRefreshTokenAsync(player.Id, session!.RefreshToken); // always the
newest one
                return session;
            }
            if (outcome == RefreshOutcome.Ended) break;
            refreshToken = await users.GetRefreshTokenAsync(player.Id); // Rotated: a
parallel request won
        }
        // No refresh token yet (players from before the migration) or the session ended.
        var fresh = await platform.StartUserSessionAsync(player.BazinessUserId!.Value);
        await users.SetRefreshTokenAsync(player.Id, fresh.RefreshToken);
        return fresh;
    }

    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 me = await users.EnsureByDeviceAsync(body.DeviceId);
        try
        {
            if (body.LinkCode is not null || me.BazinessUserId is null)
            {
                PlatformSession? s = null;
                try { s = await platform.CreateUserSessionAsync(body.DeviceId, "Web", "Direct",
body.LinkCode); }
                catch (HttpRequestException e) when (body.LinkCode is not null && e.StatusCode ==
HttpStatusCode.BadRequest) { }

                if (s is not null)
                {
                    var owner = await users.FindByBazinessUserIdAsync(s.UserId);
                    if (owner is not null && owner.Id != me.Id)
                    {
                        await users.SetRefreshTokenAsync(owner.Id, s.RefreshToken);
                        return Results.Ok(new { enabled = true, pub.WebBase, pub.ApiBase,
pub.PackageName, switchToDeviceId = owner.DeviceId });
                    }
                    if (me.BazinessUserId is not null && me.BazinessUserId != s.UserId)
                    {
                        var created = await users.CreatePlayerForBazinessAsync(s.UserId,
s.IsLinkedToAccount);
                        await users.SetRefreshTokenAsync(created.Id, s.RefreshToken);
                        return Results.Ok(new { enabled = true, pub.WebBase, pub.ApiBase,

```

```

pub.PackageName, switchToDeviceId = created.DeviceId });
    }
    await users.LinkBazinessAsync(me.Id, s.UserId, s.IsLinkedToAccount);
    await users.SetRefreshTokenAsync(me.Id, s.RefreshToken);
    return Results.Ok(ForClient(pub, s));
}
if (me.BazinessUserId is null) // stale code and no Baziness
user yet
{
    var s2 = await platform.CreateUserSessionAsync(body.DeviceId, "Web",
"Direct", null);
    await users.LinkBazinessAsync(me.Id, s2.UserId, s2.IsLinkedToAccount);
    await users.SetRefreshTokenAsync(me.Id, s2.RefreshToken);
    return Results.Ok(ForClient(pub, s2));
}
}

var session = await SessionForAsync(me, platform, users);
if (session.IsLinkedToAccount != me.IsBazinessLinked) await
users.SetLinkedAsync(me.Id, session.IsLinkedToAccount);
return Results.Ok(ForClient(pub, session));
}
catch (Exception ex)
{
    log.LogError(ex, "platform token failed");
    var error = ex.Message == "account_suspended" ? "account_suspended" :
"platform_unavailable";
    return Results.Ok(new { enabled = true, pub.WebBase, pub.ApiBase, pub.PackageName,
error }); // never block the boot
}
});

// Protect it like your other player endpoints: only the player themself may ask for their
code.
app.MapPost("/api/platform/link-code", async (DeviceRequest body, BazinessPlatformClient
platform, IUserStore users) =>
{
    var me = await users.FindByDeviceAsync(body.DeviceId);
    if (me?.BazinessUserId is null) return Results.Conflict(new { error = "call
/api/platform/token first" });
    try { return Results.Ok(new { linkCode = await
platform.CreatePagelinkCodeAsync(me.BazinessUserId.Value) }); }
    catch { return Results.StatusCode(503); }
});

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); //
POST now; verify right away
        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 });
        if (me.BazinessUserId != bazinessUserId) await users.SetRefreshTokenAsync(me.Id,

```

```

null); // belonged to the old user
    await users.LinkBazinessAsync(me.Id, bazinessUserId, isLinked: true);
    return Results.Ok(new { ok = true }); //
the client then calls /token again
}
catch { return Results.Ok(new { ok = false, reason = "invalid" }); }
});

// The page sent `logout`: end the session before the client starts over as a new visitor.
app.MapPost("/api/platform/logout", async (DeviceRequest body, BazinessPlatformClient
platform, IUserStore users) =>
{
    var me = await users.FindByDeviceAsync(body.DeviceId);
    if (me?.BazinessRefreshToken is { } refreshToken)
    {
        await platform.EndUserSessionAsync(refreshToken);
        await users.SetRefreshTokenAsync(me.Id, null);
    }
    return Results.Ok(new { ok = true });
});

public record TokenRequest(string DeviceId, string? LinkCode);
public record DeviceRequest(string DeviceId);
public record LoginRequest(string DeviceId, string Token);

```

Python

```

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from requests import HTTPError
from baziness.platform import platform, platform_client_config, platform_enabled
from app import db # your own data layer. New column on the player: baziness_refresh_token
(a secret)

app = FastAPI()

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

class DeviceRequest(BaseModel):
    deviceId: str

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

def for_client(s: dict) -> dict: # the refresh token never leaves your server
    return {"token": s["accessToken"], "tokenExpiresAt": s["accessTokenExpiresAt"],
            "bazinessUserId": s["userId"], "isLinkedToAccount": s["isLinkedToAccount"]}

def keep(player_id: int, s: dict) -> dict:
    db.set_refresh_token(player_id, s["refreshToken"]) # always the newest one
    return s

```

```

def session_for(player) -> dict:
    token = player.baziness_refresh_token
    for _ in range(2):
        if not token:
            break
        r = platform.refresh_user_session(token)
        if isinstance(r, dict):
            return keep(player.id, r)
        if r is None:
            # the session ended
            break
        token = db.get_refresh_token(player.id) # "rotated": a parallel request won
        # No refresh token yet (players from before the migration) or the session ended.
        return keep(player.id, platform.start_user_session(player.baziness_user_id))

@app.post("/api/platform/token")
def platform_token(body: TokenRequest):
    if not platform_enabled():
        return {"enabled": False}
    config = {"enabled": True, **platform_client_config()}
    me = db.ensure_user_by_device(body.deviceId)
    try:
        if body.linkCode or not me.baziness_user_id:
            try:
                s = platform.create_user_session(body.deviceId, "Web", "Direct",
body.linkCode)
            except HTTPError as e:
                if not (body.linkCode and e.response.status_code == 400):
                    raise
                s = None
                # expired or used code
            if s:
                owner = db.find_user_by_baziness_user_id(s["userId"])
                if owner and owner.id != me.id:
                    # the same person, another player of
yours
                    keep(owner.id, s)
                    return {**config, "switchToDeviceId": owner.device_id}
                if me.baziness_user_id and me.baziness_user_id != s["userId"]:
                    fresh = db.create_player_for_baziness(s["userId"],
s["isLinkedToAccount"])
                    keep(fresh.id, s)
                    return {**config, "switchToDeviceId": fresh.device_id}
                db.link_baziness(me.id, s["userId"], s["isLinkedToAccount"])
                return {**config, **for_client(keep(me.id, s))}
            if not me.baziness_user_id:
                # stale code and no Baziness user yet
                s2 = platform.create_user_session(body.deviceId, "Web", "Direct", None)
                db.link_baziness(me.id, s2["userId"], s2["isLinkedToAccount"])
                return {**config, **for_client(keep(me.id, s2))}
        s = session_for(me)
        if s["isLinkedToAccount"] != me.is_baziness_linked:
            db.set_linked(me.id, s["isLinkedToAccount"])
        return {**config, **for_client(s)}
    except PermissionError:
        return {**config, "error": "account_suspended"}
    except Exception as err: # never block the boot
        print("[platform] token failed:", err)
        return {**config, "error": "platform_unavailable"}

```

```

@app.post("/api/platform/link-code") # protect it like your other player endpoints
def platform_link_code(body: DeviceRequest):
    me = db.find_user_by_device(body.deviceId)
    if not me or not me.baziness_user_id:
        raise HTTPException(409, "call /api/platform/token first")
    return {"linkCode": platform.create_page_link_code(me.baziness_user_id)}

@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) # POST now;
        verify right away
    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}
    if me.baziness_user_id != baziness_user_id:
        db.set_refresh_token(me.id, None) # belonged to
        the old user
    db.link_baziness(me.id, baziness_user_id, True)
    return {"ok": True} # the client
    then calls /token again

@app.post("/api/platform/logout") # the page sent `logout`
def platform_logout(body: DeviceRequest):
    me = db.find_user_by_device(body.deviceId)
    if me and me.baziness_refresh_token:
        platform.end_user_session(me.baziness_refresh_token)
        db.set_refresh_token(me.id, None)
    return {"ok": True}

```

Client, step by step

1. **On boot**, send `linkCode` (from `?linkCode=` in the address, or from your deep link) to `/api/platform/token` instead of `linkedToken`. Remove both `linkCode` and `linkedToken` from the address afterwards: a code works once.
2. **Stop storing the token** for later boots (`localStorage`, `PlayerPrefs`). Your server keeps the session and gives you a fresh token on every `/api/platform/token`.
3. **Before using the token** (`user/get-coin`), check `tokenExpiresAt`. If it expires within a minute, or a call answers `401`, call `/api/platform/token` again.
4. **To open a Baziness page**, get a code from `/api/platform/link-code` and put `linkCode=<code>` in the page address instead of `linkedToken`. All other parameters stay the same (pages).
5. **On the page's logout message**, call `/api/platform/logout`, then start over as before.

TypeScript

```
// PlatformController – the parts that change.

// 1. Boot: forward the one-time ?linkCode= (it replaces ?linkedToken=). Don't keep the token
in
// localStorage any more: your server keeps the session and gives you a fresh token on
demand.
async init(): Promise<boolean> {
  const params = new URLSearchParams(location.search);
  const linkCode = params.get("linkCode");
  const res = await fetch("/api/platform/token", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ deviceId: this.deviceId, linkCode }),
  });
  const data = await res.json();
  if (data.switchToDeviceId) return this.switchTo(data.switchToDeviceId); // unchanged
  this.token = data.token ?? null;
  this.tokenExpiresAt = data.tokenExpiresAt ? Date.parse(data.tokenExpiresAt) : 0;
  if (linkCode || params.has("linkedToken")) { // a code works
    once
    params.delete("linkCode"); params.delete("linkedToken");
    const rest = params.toString();
    history.replaceState(null, "", rest ? `${location.pathname}?${rest}` :
location.pathname);
  }
  return !!this.token;
}

// 2. The token lives 15 minutes: ask your server again shortly before it expires, or after a
401.
async accessToken(): Promise<string | null> {
  if (!this.token || this.tokenExpiresAt - Date.now() < 60_000) await this.init();
  return this.token;
}

async getCoin(): Promise<number> {
  const call = async () => fetch(`${this.apiBase}/user/get-coin`, {
    headers: { Authorization: `Bearer ${await this.accessToken()}` },
  });
  let res = await call();
  if (res.status === 401) { this.token = null; res = await call(); } // renew once,
then give up
  return res.ok ? Number((await res.json()).amount ?? 0) : 0;
}

// 3. Pages: a fresh one-time code for every page you open, instead of the token.
async pageUrl(path: string, o: PageOptions = {}): Promise<string> {
  const res = await fetch("/api/platform/link-code", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ deviceId: this.deviceId }),
  });
  const { linkCode } = await res.json();
  const url = new URL(bazinessPageUrl(this.webBase, path, this.packageName, null, o)); // no
token any more
  url.searchParams.set("linkCode", linkCode);
  return url.toString();
}
```

```

}

// 4. The page sent `logout`: tell your server first, then start over as before.
async onPageLogout() {
    await fetch("/api/platform/logout", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ deviceId: this.deviceId }),
    }).catch(() => {});
    this.resetAndRestart(); // unchanged: delete local data, new visitor
}

```

C# (Unity)

```

// BazinessLogin – the parts that change.
public DateTimeOffset TokenExpiresAt { get; private set; }

[Serializable] class LinkCodeResponse { public string linkCode; }

// 1. Boot: forward the one-time ?linkCode= / deep-link code (it replaces linkedToken).
// Stop saving the token in PlayerPrefs: your server keeps the session.
public IEnumerator Init(string deviceId, string linkCode = null)
{
    var body = $"{{\"deviceId\":\"{deviceId}\",\"linkCode\":{Quote(linkCode)}}}";
    using var req = Post("/api/platform/token", body);
    yield return req.SendWebRequest();
    var r = JsonUtility.FromJson<TokenResponse>(req.downloadHandler.text); // TokenResponse
    gains: string tokenExpiresAt
    if (!string.IsNullOrEmpty(r.switchToDeviceId)) { SwitchTo(r.switchToDeviceId); yield
break; } // unchanged
    Token = r.token;
    TokenExpiresAt = DateTimeOffset.TryParse(r.tokenExpiresAt, out var t) ? t :
DateTimeOffset.MinValue;
}

// 2. The token lives 15 minutes: renew it through your server before using it.
public IEnumerator EnsureFreshToken(string deviceId)
{
    if (string.IsNullOrEmpty(Token) || TokenExpiresAt - DateTimeOffset.UtcNow <
    TimeSpan.FromMinutes(1))
        yield return Init(deviceId);
}

// 3. Pages: a fresh one-time code for every page you open, instead of the token.
public IEnumerator OpenPage(string deviceId, string path, Action<string> open,
    string redirect = null, bool allowGuest = true, string extra = null)
{
    using var req = Post("/api/platform/link-code", $"{{\"deviceId\":\"{deviceId}\",\"linkCode\":{Quote(linkCode)}}}");
    yield return req.SendWebRequest();
    var linkCode = JsonUtility.FromJson<LinkCodeResponse>(req.downloadHandler.text).linkCode;
    open(PageUrl(path, linkCode, redirect, allowGuest, extra));
}

public string PageUrl(string path, string linkCode, string redirect = null, bool allowGuest =
true, string extra = null)
{
    var url = $"{webBase.TrimEnd('/')}{path}?packageName=
{Uri.EscapeDataString(packageName)}&sdkType=js";
    if (!string.IsNullOrEmpty(linkCode)) url += "&linkCode="

```

```
{Uri.EscapeDataString(linkCode)}}"; // was linkedToken
    if (!string.IsNullOrEmpty(redirect)) url += $"&redirect={redirect}";
    if (!allowGuest) url += "&allowGuest=false";
    if (!string.IsNullOrEmpty(extra)) url += $"&{extra}";
    return url;
}
```

The new platform endpoints

All are under the usual `{api}` base. The `client` endpoints need your **client token**, as before.

Create a user, with a session

HTTP

```
POST {api}/v2/client/create-user
Authorization: Bearer <client token>
Content-Type: application/json
```

```
{ "deviceId": "8f1c2a...", "platform": "Android", "store": "CafeBazaar", "linkCode": null }
```

→ 200 OK

```
{
  "userId": 239236,
  "token": "eyJhbGciOiJIUzI1NiIs...", // the access token (same as
tokens.accessToken)
  "isLinkedToAccount": false,
  "tokens": {
    "accessToken": "eyJhbGciOiJIUzI1NiIs...", // 15 minutes – the client uses this
    "accessTokenExpiresAt": "2026-10-22T10:15:00+00:00",
    "refreshToken": "rt1.dkUoqaDDG8YB15KAQN01q77FE...", // keep on YOUR server only
    "refreshTokenExpiresAt": "2026-11-21T10:00:00+00:00"
  }
}
```

```
→ 400 { "success": false, "error": "invalid_link_code" } // the code expired or was already
used
```

Renew a session

No client token: the refresh token itself is the credential. Call it **from your server**.

HTTP

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

```
{ "refreshToken": "rt1.dkUoqaDDG8YB15KAQN01q77FE..." }
```

```

→ 200 OK
{
  "userId": 239236,
  "isLinkedToAccount": true,
  "accessToken": "eyJhbGciOiJIUzI1NiIs... ",
  "accessTokenExpiresAt": "2026-10-22T10:30:00+00:00",
  "refreshToken": "rt1.Wn0QhZ...", // store THIS one – the old one no longer
works
  "refreshTokenExpiresAt": "2026-11-21T10:15:00+00:00"
}

→ 401 { "success": false, "error": "refresh_token_rotated" } // another request refreshed
first: reload yours, retry once
→ 401 { "success": false, "error": "session_revoked" } // also invalid_refresh_token,
refresh_token_expired,
// refresh_token_reused: get a
new session
→ 403 { "success": false, "error": "account_suspended" }

```

A new session for a known player

HTTP

```

POST {api}/v2/client/get-user-token/239236
Authorization: Bearer <client token>

→ 200 OK
{ "userId": 239236, "token": "eyJhbGciOiJIUzI1NiIs...", "isLinkedToAccount": true,
  "tokens": { "accessToken": "...", "accessTokenExpiresAt": "...", "refreshToken": "rt1...",
"refreshTokenExpiresAt": "..." } }

→ 403 { "success": false, "error": "access denied" } // the user belongs to another game

```

A link code for opening a page

HTTP

```

POST {api}/v2/client/link-code/239236
Authorization: Bearer <client token>

→ 200 OK
{ "code": "q3Jc9yX2tL0...", "expiresAt": "2026-10-22T10:02:00+00:00" } // one page, once, ~2
minutes

→ 403 { "success": false, "error": "access denied" } // the user belongs to another game

```

Verify a token (body instead of address)

HTTP

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

```
{ "token": "eyJhbGciOiJIUzI1NiIs..." }
```

→ 200 OK

```
{ "userId": 239236, "isValid": true }
```

→ 200 OK (invalid or expired: not an HTTP error)

```
{ "userId": 0, "isValid": false }
```

End a session

POST {api}/auth/logout with { "refreshToken": "rt1..." }. It always answers 200 .

Errors

Answer	Where	Meaning	What to do
401 refresh_token_rotated	auth/refresh	a parallel request of this player refreshed first	reload the saved refresh token, retry once
401 session_revoked , refresh_token_expired , invalid_refresh_token , refresh_token_reused	auth/refresh	the session is over	v2/client/get-user-token/{id} , save the new refresh token
403 account_suspended	auth/refresh	the account is blocked	show the blocked message
400 invalid_link_code	v2/client/create-user	the code expired, was used, or wasn't meant for your game	continue without the code
403	v2/client/link-code/{id} , v2/client/get-user-token/{id}	the user belongs to another game	check the userId you saved
401	user/get-coin (client)	the 15-minute token expired	ask /api/platform/token again, retry once

Checklist

Test on the [development platform](#) (any phone number, code 111111):

- ☐ A new player boots: the client gets a token; your database has a refresh token for them.

- ☐ Boot again: `auth/refresh` is used (not `get-user-token`), and the stored refresh token **changed**.
- ☐ Two boots of the same player at the same moment: both succeed (one through the `rotated` retry).
- ☐ An existing player from before the migration boots once and gets a refresh token.
- ☐ Every Baziness page opens with `linkCode=...` and **no** `linkedToken`; the sign-in, profile, reward shop and offer pages all work.
- ☐ Signing in on a page with a phone that already belongs to another of your players still switches to that player.
- ☐ Opening your game from the Baziness app with `?linkCode=` lands in the right account; open the same link twice → the second time the code is ignored and the game still boots.
- ☐ After 15 minutes of play, the coin balance still loads (token renewed through your server).
- ☐ `logout` from a page ends the session; the next boot is a new visitor.
- ☐ Your logs contain no refresh tokens, and no tokens in addresses.

Note

Until the cut-off date, the Baziness app also sends `?linkedToken=`. Once your game reads `linkCode`, ignore `linkedToken`. After the cut-off it won't be sent any more, and old tokens stop working.