iBunnyApp for Developers

A plain-English handbook for studios adding Continue with iBunnyApp to a Capygram blockchain game: what it is, why it helps, how the pieces fit together, and what to do first.

Who this is for: founders, producers and engineers who want to understand the platform before writing code. It explains the concepts and the decisions you need to make.

Looking for the exact specification? The companion document, iBunnyApp AI Integration Guide, is written to be handed straight to an AI coding assistant. Read this one first, then give that one to your assistant.

Base URL: https://www.ibunnyapp.com | API version: v1

1. The problem iBunnyApp solves

A player who wants to try five Capygram games today creates five accounts, invents five passwords, and starts every one of them alone in an empty lobby. Most of them never make it past the sign-up screen. That is where your funnel leaks.

iBunnyApp gives the Capygram ecosystem one shared account layer. A player signs up once, and from then on every participating game is one button away. Their friends and their messages follow them in, so your game feels populated from the very first session.

Without iBunnyApp	With iBunnyApp
A new sign-up form and password for every game.	One tap: “Continue with iBunnyApp”.
Empty friends list on day one.	Their existing friends are already there.
You build and maintain auth, resets and support.	Identity is handled for you.
Each game holds its own isolated currency.	One Bunnycoin balance across the network.
Player-vs-player betting means holding funds.	Wagers are escrowed by iBunnyApp.

2. What you get, in six pieces

- Single sign-on. Standard OAuth 2.0. Players press one button and land in your game with a verified Capygram identity.
- Account linking. Already have players with their own accounts? They can attach an existing account to their iBunnyApp profile and keep all of their progress.
- Shared friends graph. Read the player's iBunnyApp friends so your lobby, invites and matchmaking work immediately.
- Cross-game messaging. Players read and send their iBunnyApp messages without leaving your game.

- Bunnycoin. One balance that travels with the player across every game on the network. You credit and debit it through the API.
- Escrowed wagers. Friend-versus-friend wagers are held and paid out by iBunnyApp, so your game never has to custody funds.

Every piece is optional. Plenty of studios start with sign-on only and add the social and economy features later. Nothing in the platform forces an all-or-nothing adoption.

3. How sign-in works, without the jargon

Think of it as a visitor pass. Your game does not see the player's iBunnyApp password. It asks iBunnyApp to confirm who the player is, and gets a temporary pass back.

1. A player clicks Continue with iBunnyApp in your game.
2. Your game sends them to the iBunnyApp consent screen.
3. The player sees your game's name and approves the connection. If they say no, nothing happens.
4. iBunnyApp sends them back to your game with a one-time code.
5. Your server swaps that code for an access token — the visitor pass.
6. Your server calls the iBunnyApp API with that pass to read the player's profile, friends, messages and balance.

The player stays in control the whole time. They approve your game explicitly, and they can disconnect it from their iBunnyApp dashboard whenever they like. Your engineers will recognise this as the Authorization Code flow with PKCE — the same mechanism behind “Sign in with Google”.

4. The two keys, and why the difference matters

Almost every support question we get comes down to mixing these up.

	Studio API key	Player access token
Looks like	`bnk_…`	`a long OAuth token`
Answers the question	`Which game is asking?`	`Which player is this?`
Where it comes from	`The /studio dashboard, once per game.`	`The OAuth sign-in, once per player session.`
Sent as the header	`x-bunnycoin-key`	`Authorization: Bearer …`
Lifetime	`Until you revoke it.`	`Short-lived; refreshed as needed.`
Lives where	`Server-side environment variable only.`	`Your server, for the length of the session.`

Most calls send both: your key identifies the game, the player's token identifies the person. Reads then run as that player, so a game can only ever see data the player already has access to.

Never put the studio API key in a browser, a mobile binary, or a public repository. It is shown to you exactly once when you create it, because iBunnyApp only stores a hash of it. If it leaks, revoke it in /studio and create a new one — that takes seconds and instantly kills the old key.

5. Your first afternoon: a realistic order of work

1. Create a studio account at ibunnyapp.com and register your game in /studio: name, slug, URL, tagline, emoji. This is what players see on the consent screen, so use your real game name.
2. Create an API key for that game and store it as a server-side secret.
3. Register your redirect URI — the page in your game that iBunnyApp returns players to.
4. Add the sign-in button and complete the OAuth round-trip on your server.
5. Store the player's iBunnyApp user id (a UUID) on your own user record. This is the permanent link between the two accounts; everything else can be re-fetched.
6. Call /api/public/v1/me to confirm the whole chain works. A profile and a Bunnycoin balance come back, and your game appears in the player's iBunnyApp dashboard.
7. Add the social features you want — friends, then messages, then wagers or Bunnycoin.

Steps 1 through 3 take a few minutes in the dashboard. Steps 4 through 6 are the real work, and they are the same work as adding any OAuth provider. Hand the AI Integration Guide to your coding assistant and it will write them for you.

6. What each endpoint is for

All of these live under `/api/public/v1/` and are called from your server, never the browser.

Endpoint	Use it when you want to…
`me`	Confirm who the player is and read their profile, avatar and Bunnycoin balance. This is the call to make right after sign-in.
`friends`	Populate a friends list, show who is online-capable, or restrict invites and wagers to real friends.
`messages`	Show the player's inbox, open a single conversation, or let them reply from inside your game.
`wagers`	Create, accept, decline, cancel or report the result of a friend-vs-friend wager. iBunnyApp holds the stakes.
`bunnycoin`	Read the balance, or credit and debit it for rewards, entry fees, purchases and payouts.

7. Money rules worth understanding before you write code

Always send a unique reference with a balance change. Networks drop responses. If your server retries a “credit 50 Bunnycoin” call and does not tag it with a unique `external_ref`, the player could be paid twice. With a reference, iBunnyApp recognises the repeat and applies it once. Use your own transaction id.

Wagers settle by agreement. Both players report the outcome. When they agree, the winner is paid automatically. When they disagree, the wager is marked disputed and can be re-reported — no stake is ever stranded, and your game never touches the funds.

You can only touch players who connected your game. A game cannot read or move the balance of someone who never approved it. Server-to-server jobs such as leaderboard payouts can act on a player id without a live session, but only for players who connected.

8. Common mistakes

What you'll see	What it usually means
`invalid_api_key`	The x-bunnycoin-key header is missing, mistyped, or the key was revoked.
`invalid_access_token`	The player's token expired or was never obtained. Send them through sign-in again.
`player_not_connected`	This player has not approved your game. Complete the OAuth flow for them first.
`insufficient_balance`	The debit is larger than the player's Bunnycoin balance. Check before charging.
`duplicate_external_ref`	You reused a reference. Generate a fresh one per transaction.
`Unsupported provider`	Sign-in was attempted before the provider was configured. Finish setup in /studio.

9. Questions studios ask us

Do I have to drop my own login? No. Run both. Players who already have an account with you can link it and keep their progress; new players can take the one-tap route.

What happens if a player disconnects my game? Their iBunnyApp data stops being available to you. Their progress inside your game is yours and is unaffected.

Do I need a special SDK? No. It is standard OAuth 2.0 plus plain JSON over HTTPS, so any language or framework works.

Can I test without real players? Yes. Create an iBunnyApp account for yourself and run the full flow against your own profile.

Who owns the player relationship? You do. iBunnyApp is an identity and social layer, not a storefront, and it does not sit between you and your players.

10. Ready to start

- Create your studio account and register a game: ibunnyapp.com/studio
- Skim the platform overview: ibunnyapp.com/developers
- Read how the economy works: ibunnyapp.com/bunnycoin
- Hand the iBunnyApp AI Integration Guide to your AI coding assistant and ask it to integrate exactly as specified.

One account, every Capygram game. Welcome to the network.