Avatar unlock system — take-home case

Serhii Afanasiev · Technical Product Owner, Product Madness · 9 November 2026

AI software: Claude Code, ChatGPT

MAIN DOCUMENT

Part 1. The problem, the design on one page, and how we would judge it

Recommendation. Make avatars server-owned rewards: GameOps (the team that runs the game live) publishes reviewed rules, the server credits progress from settled spins (spins whose payout is recorded) after the coins are paid, and the game shows the unlock and lets the player equip it. Players choose from the pictures they own: the default picture everyone has, avatars earned by play, and, rarely, a replacement GameOps grants when an avatar is pulled.

Where the brief is answered.

- Back-office UI: Part 3 (Figures 2–4a), Story 1 in Part 7, A2, and the A3 lifecycle actions.
- APIs: Parts 4 and 5, and the reference in A1.
- Game and server interaction: Parts 2 and 6, Stories 2 and 3, and the A3 edge cases.

Terms. A *settled spin* is one spin the spin system has recorded and paid. A *bar* is a player's progress toward one *target* avatar. A *revision* is a numbered published catalogue.

The problems. Player problem (assumed): players lack an identity earned through play. GameOps: catalogue changes need no game release, and a wrong threshold must not reach players unchecked.

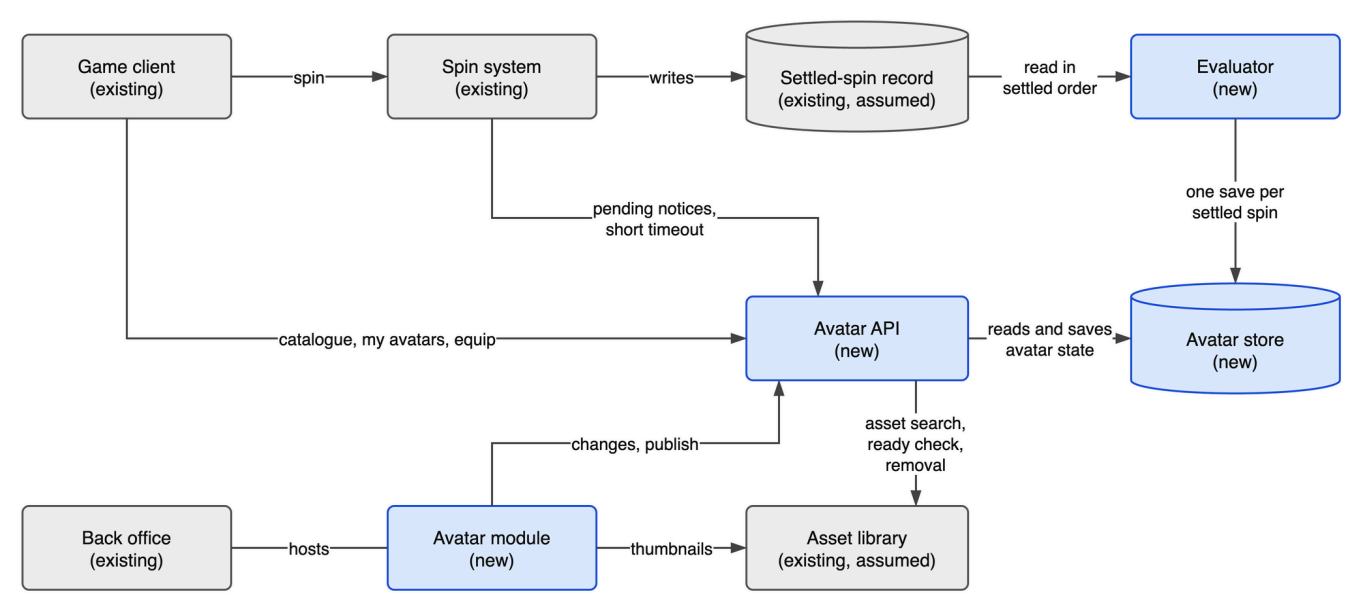


Figure 1. System picture: new parts blue, existing grey; components explained in Part 4

Six decisions commit the design: the ladder (#1), what counts (#2), pinned terms (#3), where the award is decided (#4), taking an avatar out (#5) and publication (#6). Part 2 gives each one with what it rests on, what it costs and how hard it is to reverse.

New avatar path. GameOps publishes a new avatar, Golden Cat (win 2,000,000) → a settled spin of Ana, a player, completes her bar → the server records ownership and the notice → the game shows "Golden Cat unlocked" → Equip now.

Goals. Player: show who I am and earn something lasting from play. Business: more players returning and picking an avatar, from a reward that costs no coins, measured against a holdout (players kept without the feature); what we watch is in Part 7.

Part 2. Progress rule and the six consequential decisions

The rule. Each earnable avatar has one rule: a mode (`bet` or `win`) and an `unlockThreshold` in coins. The player's bar adds up what counts (table below). When it reaches the threshold (inclusive), the player owns the avatar, once. The hiring team left the meaning of "win X coins" to me; the table is my answer. I also chose that spins on every slot game in the app count (assumed). The spin system already records what kind each spin was (assumed); I call that field `spinKind` (dependency 1).

Settled spin (`spinKind`)	Bet mode adds	Win mode adds the *credited win* (gross, not net of stake)
Paid spin (`paid`)	the paid stake, once	its credited win
Free spin or feature it triggered (`triggered_feature`)	nothing	each spin's credited win
Bought feature (`bought_feature`)	the purchase price, once	each free game's credited win
Promotional free spin (`promotional`)	nothing	nothing (assumed)

Increments are read per spin, not also as a round summary; a bought feature's price sits on its purchase settlement only. Golden Cat (win 2,000,000), bar at 1,200,000: a credited win of 799,999 leaves it unowned; 800,000 or 900,000 owns it. The extra 100,000 is discarded: it does not carry to the next avatar, whose bar starts from zero (#1, ladder).

The ladder. A player has at most two bars, one per mode.

1. **Next target:** the live, earnable, not paused, not-yet-owned avatar with the lowest `unlockThreshold` in that mode (tie: earliest published). Until the player's next settled spin it shows as "next up": a locked tile with its rule.
2. **Assignment:** that spin assigns the target, pins its terms and credits it at once, using the revision that was live when the spin settled, not when it is processed. The ring appears from that spin.
3. **Fresh bars:** a new bar starts at zero, except progress carried over when GameOps removes the current target. So one spin completes at most one avatar per mode.

Six decisions that commit the design (cited elsewhere by number, #1–#6)

Decision	Chosen	Rejected	Why, and what it rests on	Cost	Hard to reverse?
#1 Ladder	Players chase one avatar per mode at a time; each new bar starts from zero	All rules at once; lifetime counter	One big win does not sweep the catalogue	Overflow discarded	Yes: progress
#2 What counts	A bet counts the paid stake once; a win counts each spin's credited win	Round total; net win	No overlap, no round boundary; progress is cumulative, on every slot game	`spinKind` on every record	Yes: contributions
#3 Pinned terms	A player's target and threshold are fixed when the chase starts	Latest rule	A goal does not move while a player chases it; replays match	Fixes need an audited correction	Yes: pins
#4 Where it is decided	Awards are decided after the spin has paid, from the settled-spin record	Inside the spin call	Spin settlement does not wait on the avatar service; needs replayable records (dependency 1)	Award lags the coins	Medium: integration
#5 Taking an avatar out	Retire leaves it with its owners; a rare Pull and replace gives them a named replacement	Taking it from owners without a replacement; coins	Owners keep what they earned; only legal or offensive content justifies a pull	Replacement art	Pull: yes
#6 Publication	Reviewed revisions, a second approver in production, a *kill switch* (feature off) for emergencies	Live edits	Checked changes with a preview; uses the existing GameOps role and flag tool	Slower changes	No: workflow

Product assumption. Every account starts with the default avatar (today's silhouette), owned and equipped; all others are earned, and earning does not equip ("Equip now").

Part 3. Back-office UI and the player screens it drives

Who is who. Maria (GameOps) edits a rule, Jordi (GameOps) approves and publishes it, and Ana, a player, plays; the same people and numbers run through Parts 5–7. Back-office screens show full numbers (2,000,000), player screens compact ones (1.4M / 2M); a progress ring takes its mode's colour, amber for win and blue for bet.

Change workflow. Maria opens a change on the live revision (Draft) → submits it (In review; checks run) → a second GameOps user approves it (in production, not its author) → it is published as revision N+1. An edit clears the approval; Send back returns it to Draft. The three screens follow c-311 through these steps.

Step 1. Maria edits the rule (rule editor). An image from the asset library, availability, and the unlock rule only for Earnable; the right column states the rule in Part 7's words. She submits it. With c-311 in review, "Save changes" keeps it In review and clears its approval (#6, publication).

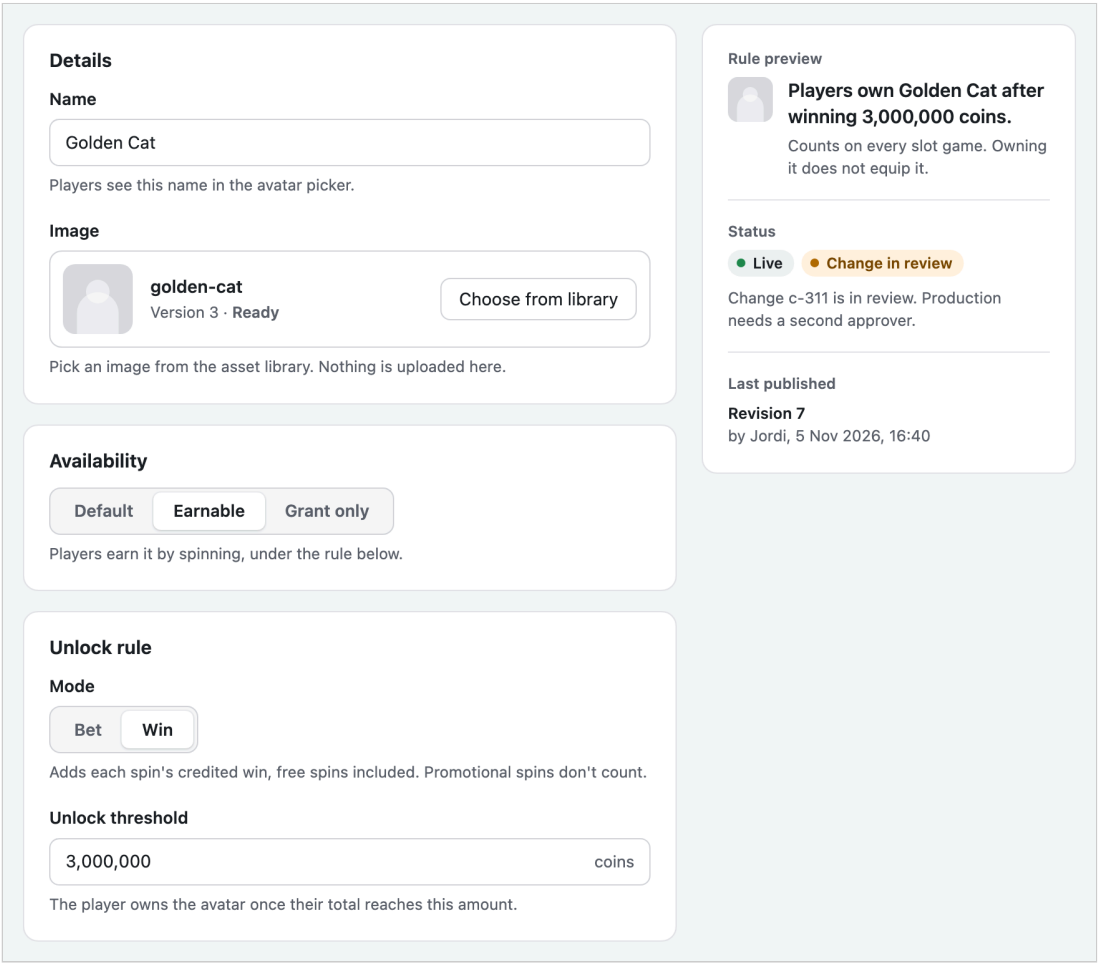


Figure 2. Rule editor: Maria's change c-311 raises Golden Cat from 2,000,000 to 3,000,000

Step 2. Jordi finds it in the catalogue (production). Last updated first. Golden Cat stays Live while Maria's change c-311 is in review, with a "Change in review · Review" chip; each row's menu holds Retire and Pull and replace (#5, taking an avatar out).

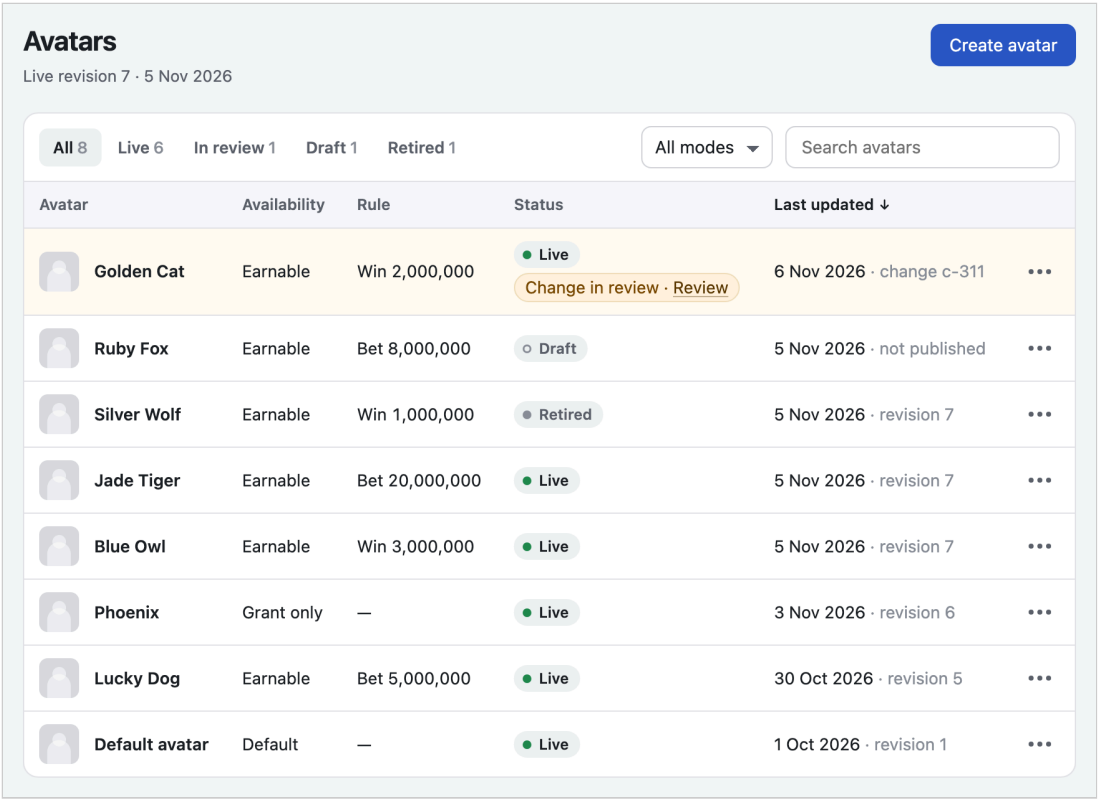


Figure 3. Catalogue

Step 3. Jordi reviews and publishes (review). Diff against revision 7, checks, and what players see: those already chasing Golden Cat keep their pinned 2,000,000 (#3, pinned terms). "Approve and publish" is A1's one control; the author's approval is refused, and the second approver applies in production only. Version history and audit log: Figure 4b in A2.

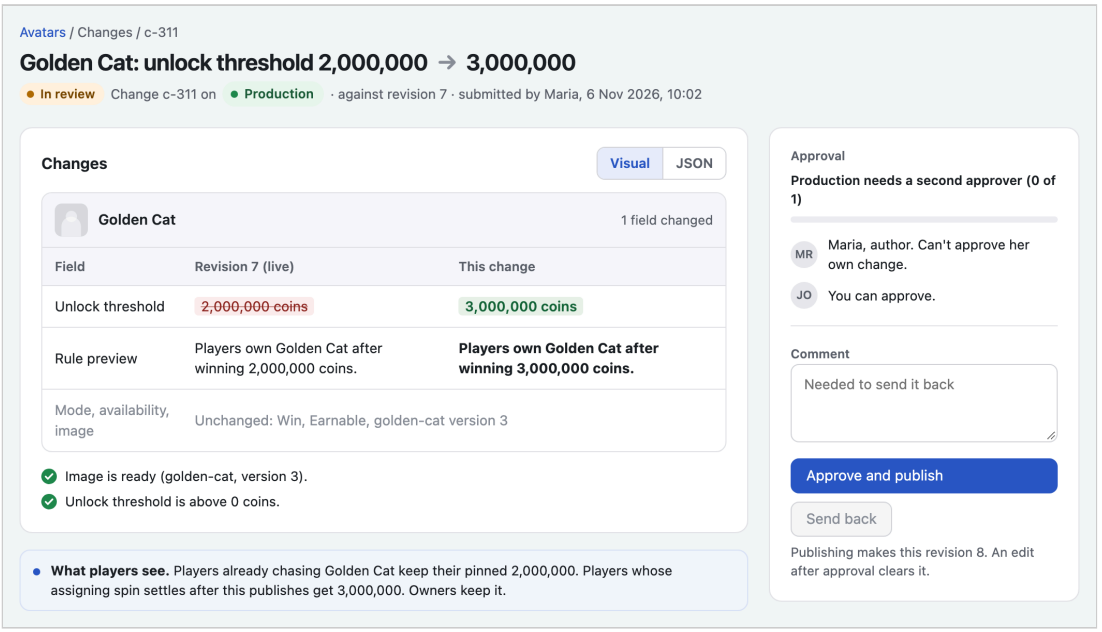


Figure 4a. Review: diff and approval

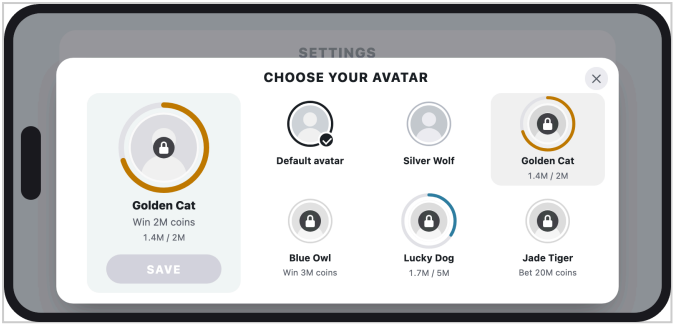


Figure 5. Avatar picker: Ana chasing two targets

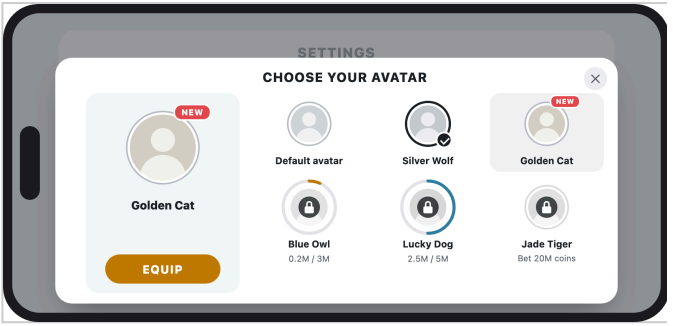


Figure 6. Avatar picker a few spins after Golden Cat unlocked

Avatar picker (Ana; a dialog over Settings; full rules in A2). It shows what Ana owns and the live earnable avatars, so grant-only Phoenix (given only by GameOps; in this version only as a Pull and replace replacement) is hidden. A ring marks each current target, one per mode; other earnable avatars show their rule ("Win 3M coins", "Bet 20M coins"). An avatar whose rule is paused (`pausedAvatarIds`, A1) is greyed with "Paused" until resume.

- Figure 5: Golden Cat 1.4M / 2M (win) and Lucky Dog 1.7M / 5M (bet).
- Figure 6, a few spins later (after A1's sample state): Golden Cat is owned, with a NEW badge. Blue Owl is the new win target; its bar started from zero at the next settled spin and reads 0.2M / 3M (#1, ladder). Lucky Dog reads 2.5M / 5M.

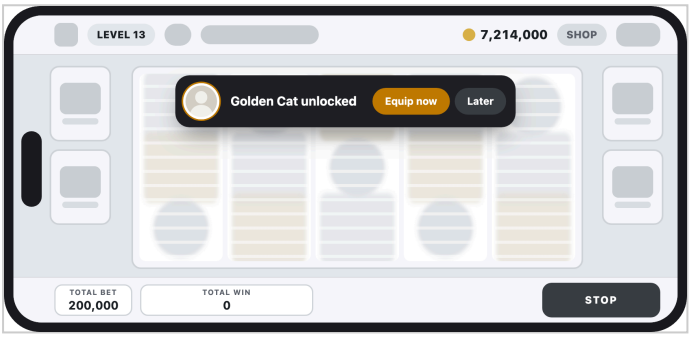


Figure 7. Slot screen at the unlock

Slot screen. One addition to the existing screen: the toast "Golden Cat unlocked" with Equip now / Later, once the server has recorded the grant (Part 6). The toast does not block play.

Part 4. Logical components and small data model

Figure 1 (Part 1) shows these components: new parts blue, existing grey. Catalogue changes need no game release: the game reads the catalogue from the Avatar API. The design builds on the existing spin system; the notice read adds a bounded overhead to the spin reply.

- **Avatar module (new):** the back-office screens GameOps uses for the catalogue, the rule editor and the review; it sends changes and publish requests to the Avatar API, which searches the asset library, checks that images are ready and stops serving pulled images.
- **Avatar API (new):** the player, admin and internal calls of Part 5.
- **Evaluator (new):** reads settled-spin records in each player's settled order and makes one all-or-nothing save per spin: credits, assignments, grants and notices (#4, where it is decided).
- **Avatar store (new):** where catalogue revisions, bars, ownership and the equipped choice are saved, each update in one all-or-nothing save.

Entity	Key	Main fields	Rule
Avatar	`avatarId`	`status`, `availability`, `mode`, `unlockThreshold`, `assetId`, `pullReason`	Rule fields only when earnable; a pulled avatar is `retired` with a `pullReason`
Change → revision	`changeId`; `revision`	`changeVersion`, `baseRevision`, `approvedBy`, `items`	In production the approver is not the author; a revision is not edited and keeps its effective time
Ownership	(`playerId`, `avatarId`)	`source`, `grantedAt`	One row per player and avatar, kept after Retire and Pull and replace
Selection	`playerId`	`equippedAvatarId`, `selectionVersion`	Owned and equippable (live, or retired and not pulled); +1 per change

Entity	Key	Main fields	Rule
Bar	(`playerId`, `mode`)	`targetAvatarId`, pinned `unlockThreshold` and `revision`, `amount`	Pinned at assignment; only an audited correction may lower it
Spin processing	(`gameId`, `settledSpinId`)	increments, assignments, grants	One all-or-nothing save with bars, grants and notices; kept while the record can be replayed (period agreed with the spin team, dependency 1)

Notices (`owned`, `replacement`, `progress_moved`) stay pending until acknowledged; **audit entries** (who, what, when, before, after) cover every change, approval, publish and pause; kill-switch use is audited in the feature-flag tool. API values are lowercase; the back office shows Draft, In review, Live and Retired, with the actions Create, Edit, Retire and Pull and replace. All fields are in A1.

Part 5. APIs: contracts with representative requests, responses and errors

The Avatar API has three call classes: player calls take the player from the session; admin calls use the back-office login and the existing GameOps role; internal calls use a service identity. No call lets the game report an award: the server decides (#4, where it is decided). The main calls at a glance (every endpoint, payload and error in A1):

Method	Path	Who	Purpose
GET	`/v1/catalogue`	Game	Live and retired avatars of the current revision (`ETag` = revision)
GET	`/v1/me/avatars`	Game	The player's state: owned, equipped, bars, pending notices
PUT	`/v1/me/equipped-avatar`	Game	Equip an owned, equippable avatar (below)
POST	`/v1/me/notices/{noticeId}/ack`	Game	Mark a notice as shown
GET	`/internal/v1/players/{playerId}/avatar-notices`	Spin system	Pending notices for the spin reply, under a short timeout
PUT	`/admin/v1/changes/{changeId}`	Back office	Edit a change's items (`If-Match: changeVersion`); clears an approval
POST	`/admin/v1/changes/{changeId}/approve`	Back office	Record the approver, in production not the author
POST	`/admin/v1/changes/{changeId}/publish`	Back office	Validate and publish the approved version as the next revision

`PUT /v1/me/equipped-avatar` · Game · Session (the player comes from the session, not the body)

Purpose. Equip an avatar the player owns and may equip. The server decides; the reply is the full player state.

Field	Type	Required	Meaning
`avatarId`	number	yes	Avatar to equip
`selectionVersion`	number	yes	Version the device last saw; stale → `SELECTION_CONFLICT`

Response 200: player state (A1): `owned`, `equipped` {`avatarId`, `selectionVersion` +1}, `bars`, `notices`, `pausedAvatarIds`, `processedThrough`, `catalogueRevision`.

Errors, checked in this order:

HTTP	Code	When	Game does
404	`AVATAR_NOT_FOUND`	Unknown `avatarId`	Refetch the catalogue
403	`AVATAR_NOT_OWNED`	Not owned	Show it locked
409	`AVATAR_NOT_EQUIPPABLE`	Owned, but pulled	Refetch catalogue and state
409	`SELECTION_CONFLICT`	Stale `selectionVersion`; `details.equipped` = {`avatarId`, `selectionVersion`}	Redraw from `details`

Also: 400 `INVALID_REQUEST`, 401 `UNAUTHENTICATED`, 503 `SERVICE_UNAVAILABLE` (A1).

Retry. Safe. A retry runs the usual checks in order. If the first call was saved and the avatar is still equippable, the retry returns `SELECTION_CONFLICT`, treated as success when `details.equipped` matches the request (Figure 9, step 4); if the avatar was pulled in between, it returns `AVATAR_NOT_EQUIPPABLE`. If the first call was not saved, the retry succeeds only if its `selectionVersion` is still current and the avatar is still equippable. Any other `SELECTION_CONFLICT` redraws from `details`.

Example.

```
Phone:  PUT /v1/me/equipped-avatar   { "avatarId": 9, "selectionVersion": 12 }
200     full player state (A1): equipped avatar 9, selectionVersion 13
Tablet, still on 12:   { "avatarId": 17, "selectionVersion": 12 }
409 { "error": { "code": "SELECTION_CONFLICT",
      "message": "The equipped avatar changed since you last loaded it", "requestId": "r-5501",
      "details": { "equipped": { "avatarId": 9, "selectionVersion": 13 } } } }
```

`requestId` is for diagnosis.

Admin: publish a change. Retire and Pull and replace are change items (`retire`; `pull_and_replace` with `replacementAvatarId` and `pullReason`), published the same way. Approve and publish are separate calls; an edit between them clears the approval. A publish sent with the old `changeVersion` then returns 412 `PRECONDITION_FAILED`, and one sent with the new version returns `CHANGE_CONFLICT`, because that version was never approved. Maria authored c-311; Jordi approved it and publishes it:

```
POST /admin/v1/changes/c-311/publish   Idempotency-Key: k-8812   If-Match: "5"
200 { "changeId": "c-311", "revision": 8, "publishedBy": "jordi", "approvedBy": "jordi" }
If a check fails, nothing is published:
422 { "error": { "code": "VALIDATION_FAILED",
      "message": "A publish check failed; nothing was published", "requestId": "r-7730",
      "details": [ { "avatarId": 17, "field": "assetId",
                     "reason": "IMAGE_NOT_READY" } ] } }
```

Internal: the settled-spin record and the notice read. The Evaluator consumes the record (an assumed contract, dependency 1 in Part 7); the spin system adds the pending notices to its reply as the new optional `avatarNotices`.

```
Record:  gameId slot-42 · settledSpinId s-123 · playerId p-882 · settledAt 2026-11-06T10:15:02.120Z
         spinKind paid · paidStake 200000 · creditedWin 900000
GET /internal/v1/players/p-882/avatar-notices
200 { "notices": [ { "noticeId": "n-41", "type": "owned", "avatarId": 17 } ] }
503 { "error": { "code": "SERVICE_UNAVAILABLE",
      "message": "The avatar service is unavailable", "requestId": "r-9120" } }
    on 503 or no answer within the timeout, the spin reply goes without avatarNotices
```

Errors. Domain codes: `SELF_APPROVAL_NOT_ALLOWED`, `AVATAR_NOT_FOUND`, `AVATAR_NOT_OWNED`, `AVATAR_NOT_EQUIPPABLE` (owned but pulled), `SELECTION_CONFLICT`, `CHANGE_CONFLICT`, `PRECONDITION_FAILED` (412, stale `If-Match`), `VALIDATION_FAILED` (each detail names `field` and `reason`) and `SERVICE_UNAVAILABLE`.

Retry keys and versioning: A1.

Part 6. Game and server interaction: earning an avatar and equipping one, with timeout and retry

Earning. Ana has Golden Cat at 1,400,000 of 2,000,000 (win) and Lucky Dog at 1,700,000 of 5,000,000 (bet). The spin reply goes back as today; the award is decided afterwards (#4, where it is decided).

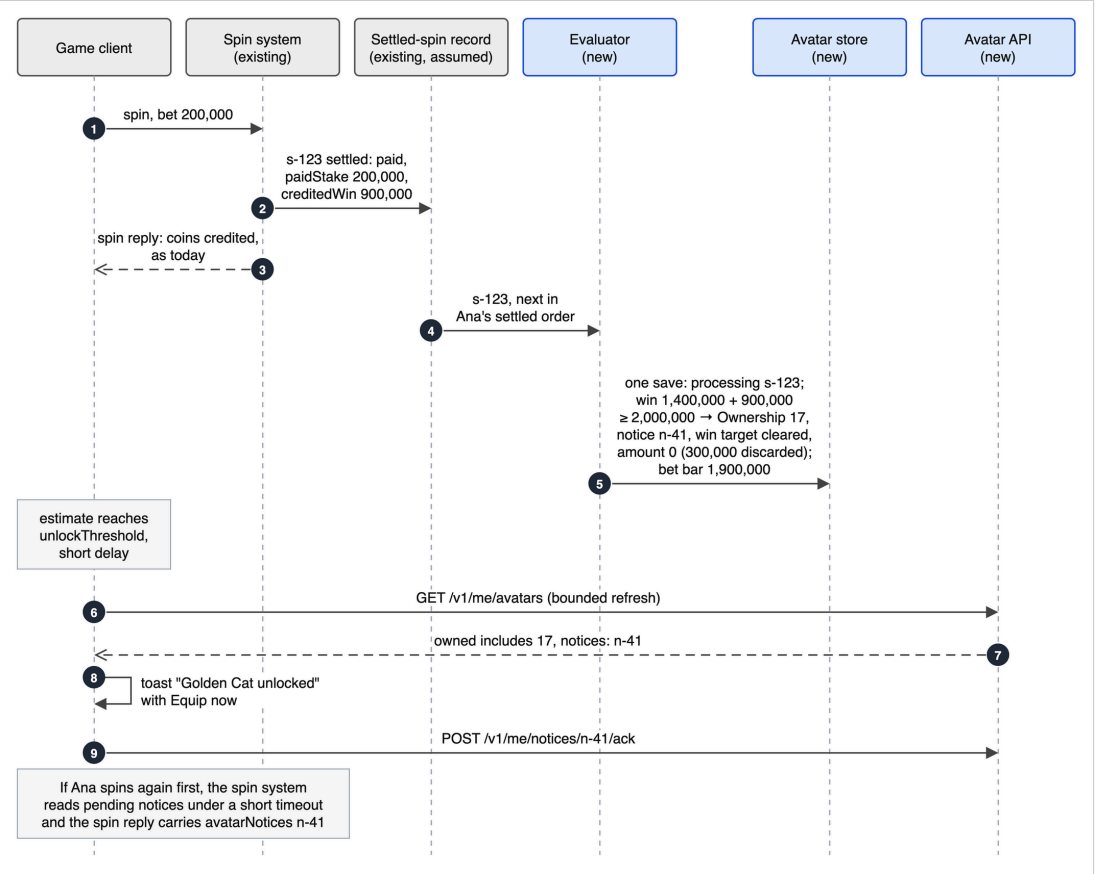


Figure 8. Earning an avatar

The game does not announce an award before the server has recorded it; how it estimates the bar and when it refreshes are in A1. The 300,000 overflow is not carried; Blue Owl is assigned at Ana's next settled spin. The game estimates each bar from the spins it has seen, so it asks the server only when an unlock is likely, with no new connection. The award can reach it two ways because either may come first: the short-delay refresh once the estimate reaches the threshold, or `avatarNotices` on the next spin reply if Ana spins again before that.

Equipping. Ana uses a phone and a tablet, both at `selectionVersion` 12.

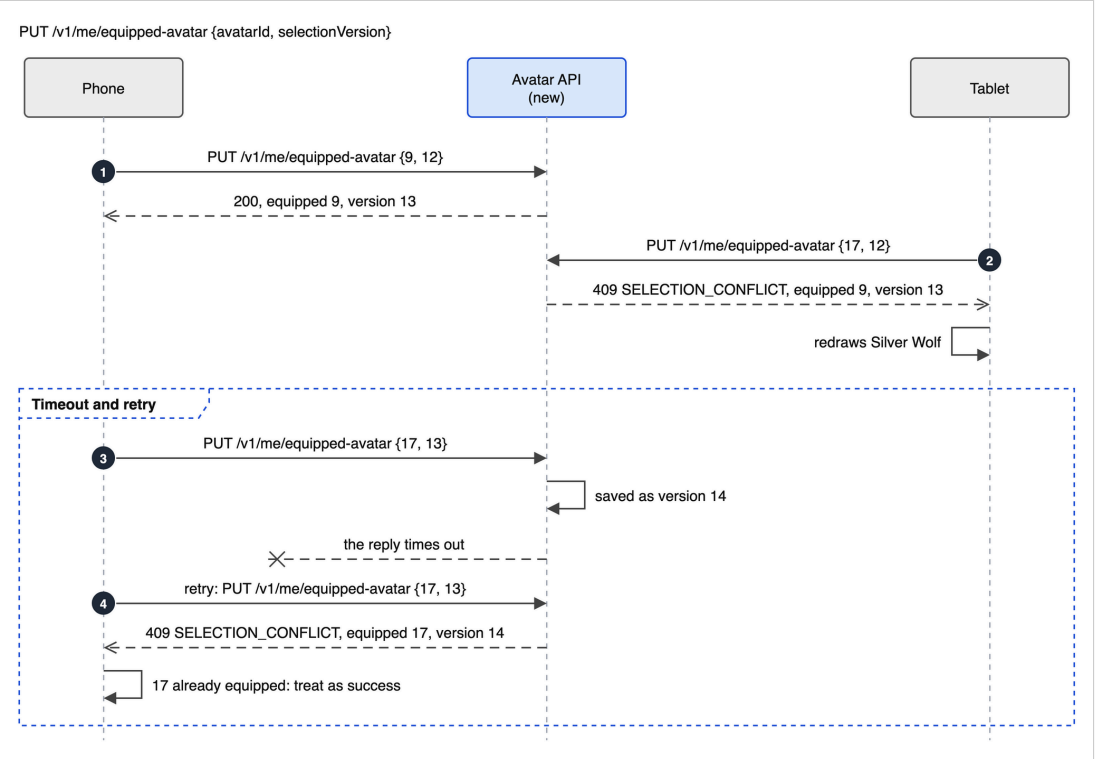


Figure 9. Equipping on two devices

When things go wrong

Case	Server	Player sees
Same settled spin twice	The processing record stops a second credit	Repeated `noticeId` skipped
Evaluator down; notice read times out	Spins pay as today; records processed in settled order on recovery; lag alarm	Toast with a later reply, refresh or fetch
Final spin completes a target	Grant and notice stored	The refresh shows the toast, or else the next fetch
Fetch fails, or the catalogue is empty, older, or lacks an avatar that `owned`, `equipped` or a bar names (checked against the state returned with or after it)	Keep the last valid cache; retry	Cached or default picture; Save waits

A rule change, a Retire and a Pull and replace, as players see them: A3.

Part 7. Acceptance criteria, release controls, measures and open dependencies

Story 1. As a GameOps user, I publish an avatar rule, and a wrong value does not reach players unchecked (#6, publication).

Given	When	Then
Maria raises Golden Cat's `unlockThreshold` from 2,000,000 to 3,000,000 in production	she submits	diff against revision 7; "Players own Golden Cat after winning 3,000,000 coins"
Maria is the author	she approves	`SELF_APPROVAL_NOT_ALLOWED`
Jordi approved	Maria edits	approval cleared; the change stays In review
Jordi approved, then Maria edits	the publish call arrives	`PRECONDITION_FAILED` with the version Jordi approved, `CHANGE_CONFLICT` with the new one; revision 7 stays
image not ready, or `unlockThreshold` missing or at or below 0 coins	anyone publishes	`VALIDATION_FAILED` names the field; revision 7 stays
Jordi approved	he publishes	revision 8 live, audited; `CHANGE_CONFLICT` if a newer revision went live

Story 2. As a player, I earn an avatar by spinning (#1, ladder; #2, what counts).

Given	When	Then
separate fixtures: Golden Cat pinned at 2,000,000, bar 1,200,000	credited win 799,999 / 800,000 / 900,000	not owned / owned / owned; overflow discarded
Lucky Dog pinned at 5,000,000, bar 0	24 / 25 / 26 paid spins of 200,000	not owned / owned / owned; spin 26 assigns Jade Tiger (44), whose bar reads 200,000
win bar 1,900,000, bet bar 4,900,000	a 200,000 paid spin pays 100,000	both owned in one save, two notices
s-123 was processed	it arrives again	no change

Story 3. As a player, I equip an avatar I own (#5, taking an avatar out, in the last row).

Given	When	Then
Ana owns retired Silver Wolf, version 12	she equips it	equipped, version 13
she does not own Blue Owl (21)	she equips it	`AVATAR_NOT_OWNED`
the tablet holds 12 after the phone saved 13	it saves	`SELECTION_CONFLICT` with the current choice; redraw
Golden Cat is equipped	its Pull and replace by Phoenix is published	Phoenix owned and equipped (`selectionVersion` + 1), one `replacement` notice and no `owned` notice; the cat leaves the catalogue and `owned` and is `AVATAR_NOT_EQUIPPABLE`

Release controls.

Control	Run by	Effect
Staged rollout: internal, 1%, 10%, 50%, then everyone except the holdout (Part 1)	The existing feature-flag tool (assumed)	Advances while the health signals hold (A3); bars start at a player's first settled spin inside it
Rule pause	A senior GameOps user, audited	Progress keeps counting, but the rule grants nothing and is not assigned as a new target until resumed (A3)
Kill switch (feature off)	The game's on-call engineer, in the existing feature-flag tool (assumed)	Hides the avatar screen, bars and toasts and shows the default picture; ownership, selection and processing carry on
Notice read off	The game's on-call engineer, in the existing feature-flag tool (assumed)	The spin reply goes without `avatarNotices`; notices stay pending until the next player-state fetch

Measures. Watched against the holdout: (1) share of active players with a non-default avatar at day 30; (2) median paid spins to first unlock, per threshold; (3) day-7 return. Guardrails: spend and session length not above the holdout; progress never shown as a near-win. A threshold with under 1% unlocks in two weeks is reviewed.

Dependencies, in order. The server team builds the Avatar API, Evaluator and Avatar store once dependency 1 is agreed; the rest work as in dependency 4.

1. Spin-system team: durable, replayable records in a deterministic per-player order (`settledSpinId` unique per game, `spinKind`, per-spin increments), `avatarNotices`, latency and freshness targets. The spin reply also gives the game the spin's `gameId` and `settledSpinId`, so it can match `processedThrough` (assumed). Fallback: the same evaluation inside the spin service after payout, one player at a time.
2. Back-office owner: GameOps role, second approver, senior permission for pause.
3. Asset library: ready, versioned, removable images; default and replacement art.
4. Client teams: the game's avatar screen (Part 3) and the back-office Avatar module, built against a stub of the Part 5 contracts. GameOps with the economy and responsible-play owner: first catalogue, pacing, and every threshold agreed before the change is submitted. QA: boundaries, duplicate and late spins, replay order, two devices, load, on a settled-spin test feed.
5. Legal: the Pull and replace message, and whether a won avatar counts as digital content.

With more time: a player-chosen goal and ladder order; a starter set; VIP gifts via `grant_only`.

APPENDIX

A1. API reference: payloads, endpoints, errors

Payloads and errors behind Part 5. "Session" = the signed-in player; "GameOps" = the existing back-office role; "senior" = the senior GameOps permission; "service" = service identity.

The shared catalogue. `GET /v1/catalogue` is the same for everyone and cached on the device; its `ETag` is the revision, so an unchanged catalogue costs a `304`. It lists every live and retired avatar; pulled ones are absent.

```
{ "revision": 7,
  "avatars": [
    { "avatarId": 17, "name": "Golden Cat", "status": "live", "availability": "earnable",
      "mode": "win", "unlockThreshold": 2000000, "imageUrl": "…/golden-cat?v=3" },
    { "avatarId": 9, "name": "Silver Wolf", "status": "retired", "availability": "earnable",
      "mode": "win", "unlockThreshold": 1000000, "imageUrl": "…/silver-wolf?v=1" },
    { "avatarId": 44, "name": "Jade Tiger", "status": "live", "availability": "earnable",
      "mode": "bet", "unlockThreshold": 20000000, "imageUrl": "…/jade-tiger?v=1" } ] }
```

Revision 7, the live catalogue before c-311 is published, holds seven avatars, three shown above: the default avatar (1), Silver Wolf (9, retired), Golden Cat (17, win 2,000,000), Blue Owl (21, win 3,000,000), Lucky Dog (30, bet 5,000,000), Phoenix (40, grant only) and Jade Tiger (44, bet 20,000,000).

Player state (the 200 of `GET /v1/me/avatars` and of equip in Part 5):

```
{ "owned": [ { "avatarId": 1, "source": "default" },
             { "avatarId": 9, "source": "rule_completion" },
             { "avatarId": 17, "source": "rule_completion" } ],
  "equipped": { "avatarId": 9, "selectionVersion": 13 },
  "bars": [ { "mode": "win", "targetAvatarId": 21, "unlockThreshold": 3000000,
              "revision": 8, "amount": 200000 },
            { "mode": "bet", "targetAvatarId": 30, "unlockThreshold": 5000000,
              "revision": 6, "amount": 2100000 } ],
  "notices": [],
  "pausedAvatarIds": [],
  "processedThrough": { "gameId": "slot-42", "settledSpinId": "s-140" },
  "catalogueRevision": 8 }
```

`owned` leaves out pulled avatars. `equipped.avatarId` is the stored `equippedAvatarId`. `pausedAvatarIds` lists the avatars whose rule is paused (A3); it is in the player state, not the catalogue, because a pause creates no revision and the catalogue is cached by revision. A mode with no target returns, for example, `{ "mode": "win", "nextUpAvatarId": 21, "amount": 0 }`; `nextUpAvatarId` is derived at read time and copied into a `progress_moved` notice. `processedThrough` is the last settled spin, in the player's order, that the state includes; the game keeps the contributions it saw after it. If that spin is not in the game's local history (for example, it was played on another device), the game counts every eligible contribution since it received that state. A newer `catalogueRevision` than the cached one makes the game refetch the catalogue.

Client estimate and refresh (behind Part 6). The game estimates each mode as the confirmed `amount` (the bar's progress in the last state) plus eligible contributions after `processedThrough`. At the pinned threshold, or the first eligible spin of a mode with no target, it refreshes from the Avatar API after a short delay and a few times more with growing gaps; each answer rebases the estimate. It skips a repeated `noticeId` while its local history lasts. Login, app return and the avatar screen fetch too; no new connection is added, and an existing push channel may notify a closed app (assumed).

Notices. Each has a stable `noticeId`, player, `type` and avatar IDs, is saved with the change that caused it and stays pending until acknowledged:

Type	Fields	Keyed by
`owned`	`avatarId`	the grant (a replacement grant creates the `replacement` notice instead of an `owned` one)
`replacement`	`avatarId`, `replacedAvatarId`	player, pulled avatar and pull revision, also when the replacement was already owned
`progress_moved`	`removedAvatarId`, `mode`, `amount`, `nextUpAvatarId`	player, removed target and revision

Settled-spin record. `paidStake`, like `creditedWin`, is made of non-overlapping increments: a triggered free spin carries 0; a bought feature's price appears only on its purchase settlement (`bought_feature`), its free games are `bought_feature` records with `paidStake` 0, and no paid spin includes it.

All model fields.

Entity	Fields
Avatar	`avatarId` (number, not reused), `name`, `status` (`draft`, `in_review`, `live`, `retired`), `availability` (`default`, `earnable`, `grant_only`; shown as Default, Earnable, Grant only), `mode` (`bet`, `win`), `unlockThreshold` (coins, inclusive), `assetId`, `imageVersion`, `firstPublishedAt`, `pullReason` (`legal`, `offensive`), `paused` (rule pause; sent to players as `pausedAvatarIds` in the player state)
Change	`changeId`, `status` (`draft`, `in_review`, `published`, `discarded`), `changeVersion`, `baseRevision`, `authorId`, `approvedBy`, `publishedBy`, published at, `items` (`create`, `edit`, `retire`, `pull_and_replace` with `replacementAvatarId` and `pullReason`); published as `revision` (whole number per environment) with its effective time
Ownership	`playerId`, `avatarId`, `source` (`default`, `rule_completion`, `replacement`), `grantedAt`, `gameId` and `settledSpinId` of the completing spin
Selection	`playerId`, `equippedAvatarId`, `selectionVersion`
Bar	`playerId`, `mode`, `targetAvatarId`, pinned `unlockThreshold` and `revision`, `amount` (with no target, carried progress)
Spin processing	`gameId`, `settledSpinId`, `playerId`, `processedAt`, increments, assignments, grants

Change lifecycle. `changeVersion` +1 on every `PUT` edit only; submit, approve and send-back keep it; a published change is closed and read-only (status `published`). A change in draft or in review can be discarded, with `If-Match` like an edit, which closes the change (status `discarded`, audited). The back office's one "Approve and publish" control makes two calls: approve, then publish.

Game endpoints

Method	Path	Auth	Safe to retry?	Purpose
GET	`/v1/catalogue`	Session	Yes (read, `ETag` = revision)	Live and retired avatars of the current revision; pulled ones left out
GET	`/v1/me/avatars`	Session	Yes (read)	Player state (above)
PUT	`/v1/me/equipped-avatar`	Session	Yes, by `selectionVersion`	Equip an owned, equippable avatar; returns the player state
POST	`/v1/me/notices/{noticeId}/ack`	Session	Yes, by `noticeId`	Mark a notice as shown

Back-office endpoints (`If-Match` carries the change's `changeVersion`, as `GET /admin/v1/changes/{changeId}` returns it (also as its `ETag`), sent quoted: `If-Match: "5"`; a stale version returns 412 `PRECONDITION_FAILED`; lifecycle conflicts return 409 `CHANGE_CONFLICT`)

Method	Path	Auth	Safe to retry?	Purpose
GET	`/admin/v1/avatars`	GameOps	Yes (read)	Catalogue list, newest first: status, availability, mode, `unlockThreshold`, `updatedAt` (derived from the latest change or revision touching it), the open `changeId` with its status, live revision
GET	`/admin/v1/assets`	GameOps	Yes (read)	Search the existing asset library; returns `assetId`, `imageVersion`, ready state
POST	`/admin/v1/changes`	GameOps	`Idempotency-Key`	Open a change on the live revision (`baseRevision`)
GET	`/admin/v1/changes/{changeId}`	GameOps	Yes (read)	The change with `changeVersion`, items and status
PUT	`/admin/v1/changes/{changeId}`	GameOps	`If-Match: changeVersion`	Edit items: `create`, `edit`, `retire`, `pull_and_replace` (with `replacementAvatarId`, `pullReason`); clears an approval

Method	Path	Auth	Safe to retry?	Purpose
POST	`/admin/v1/changes/{changeId}/submit`	GameOps	`Idempotency-Key`, `If-Match`	`draft` → `in_review`; runs validation
GET	`/admin/v1/changes/{changeId}/review`	GameOps	Yes (read)	Diff against live, preview sentences, validation results
POST	`/admin/v1/changes/{changeId}/approve`	GameOps, not the author (production)	`Idempotency-Key`, `If-Match`	Record `approvedBy`
POST	`/admin/v1/changes/{changeId}/send-back`	GameOps	`Idempotency-Key`, `If-Match`	`in_review` → `draft`, with a comment
POST	`/admin/v1/changes/{changeId}/publish`	GameOps	`Idempotency-Key`, `If-Match: changeVersion`	Validate and publish as the next revision; publishes only the version that was approved, otherwise `CHANGE_CONFLICT`
POST	`/admin/v1/changes/{changeId}/discard`	GameOps	`Idempotency-Key`, `If-Match`	From `draft` or `in_review`: closes the change; status `discarded`, audited; a stale version returns 412
GET	`/admin/v1/revisions`	GameOps	Yes (read)	Version history (A2)
GET	`/admin/v1/revisions/{revision}`	GameOps	Yes (read)	One revision's full content
POST	`/admin/v1/revisions/{revision}/republish`	GameOps	`Idempotency-Key`	Open a change that restores that revision; normal review and lifecycle validation (A2)
POST	`/admin/v1/avatars/{avatarId}/pause` and `/resume`	Senior	`Idempotency-Key`	Rule pause: the rule keeps crediting progress, grants nothing and is not assigned as a new target, until resumed
POST	`/admin/v1/corrections`	Senior	`Idempotency-Key`	Propose an audited player-state correction for one rule, run while it is paused
POST	`/admin/v1/corrections/{correctionId}/approve`	Senior, not the proposer	`Idempotency-Key`	Record the second approval; the correction then runs
GET	`/admin/v1/audit`	GameOps	Yes (read)	Audit log, filtered by avatar, user, action or time (A2)

Internal

Method	Path	Auth	Safe to retry?	Purpose
GET	`/internal/v1/players/{playerId}/avatar-notices`	Service	Yes (read)	Pending notices for the spin system's reply, under a short timeout
—	Settled-spin record	Service	Keyed by (`gameId`, `settledSpinId`)	Assumed contract (Part 5): the spin system writes it, the Evaluator reads it; consumed, not called
—	`avatarNotices` in the spin reply	Existing session	By `noticeId`	New optional field on the spin system's existing reply to the game

Errors. Every error is `{ "error": { "code", "message", "requestId", "details"? } }`; `message` is a short English sentence for logs and support, never shown to players.

Code	HTTP	When	Caller does
`INVALID_REQUEST`	400	Malformed body or unknown field value	Fix the call
`UNAUTHENTICATED`	401	No or expired session	Sign in again
`FORBIDDEN`	403	Role lacks the action (for example pause without the senior role)	Ask someone with the role
`SELF_APPROVAL_NOT_ALLOWED`	403	The author approves their own change in production	Ask another GameOps user

Code	HTTP	When	Caller does
AVATAR_NOT_OWNED	403	Equip of an avatar the player does not own	Show it locked
AVATAR_NOT_FOUND	404	Unknown avatarId	Refetch the catalogue
NOT_FOUND	404	Unknown changeId, noticeId, revision or correctionId; an already-acknowledged notice is not unknown, and its acknowledgement succeeds again	Reload the list
AVATAR_NOT_EQUIPPABLE	409	Owned, but pulled	Refetch catalogue and state
SELECTION_CONFLICT	409	Stale selectionVersion	Redraw from details
CHANGE_CONFLICT	409	A newer revision went live after baseRevision, the version being published is not the approved one (for example, the change was edited after approval), or the call is a new edit or action on a closed change (published, discarded). Reads of a closed change still work, and a repeated publish with the same Idempotency-Key returns its first result	Reload, reapply, review again; if it was never approved, get it approved; if it is already published, reload the live revision
PRECONDITION_FAILED	412	If-Match names a stale changeVersion (the change was edited since the caller loaded it)	Reload the change and re-check
VALIDATION_FAILED	422	Failing checks on submit or publish; each detail names field and reason: IMAGE_NOT_READY, THRESHOLD_OUT_OF_BOUNDS (missing or not above 0 coins), MODE_MISSING, AVAILABILITY_MISSING, REPLACEMENT_INVALID, DEFAULT_PROTECTED	Fix the fields
SERVICE_UNAVAILABLE	503	Avatar service down; Retry-After set	Keep cached state; retry later

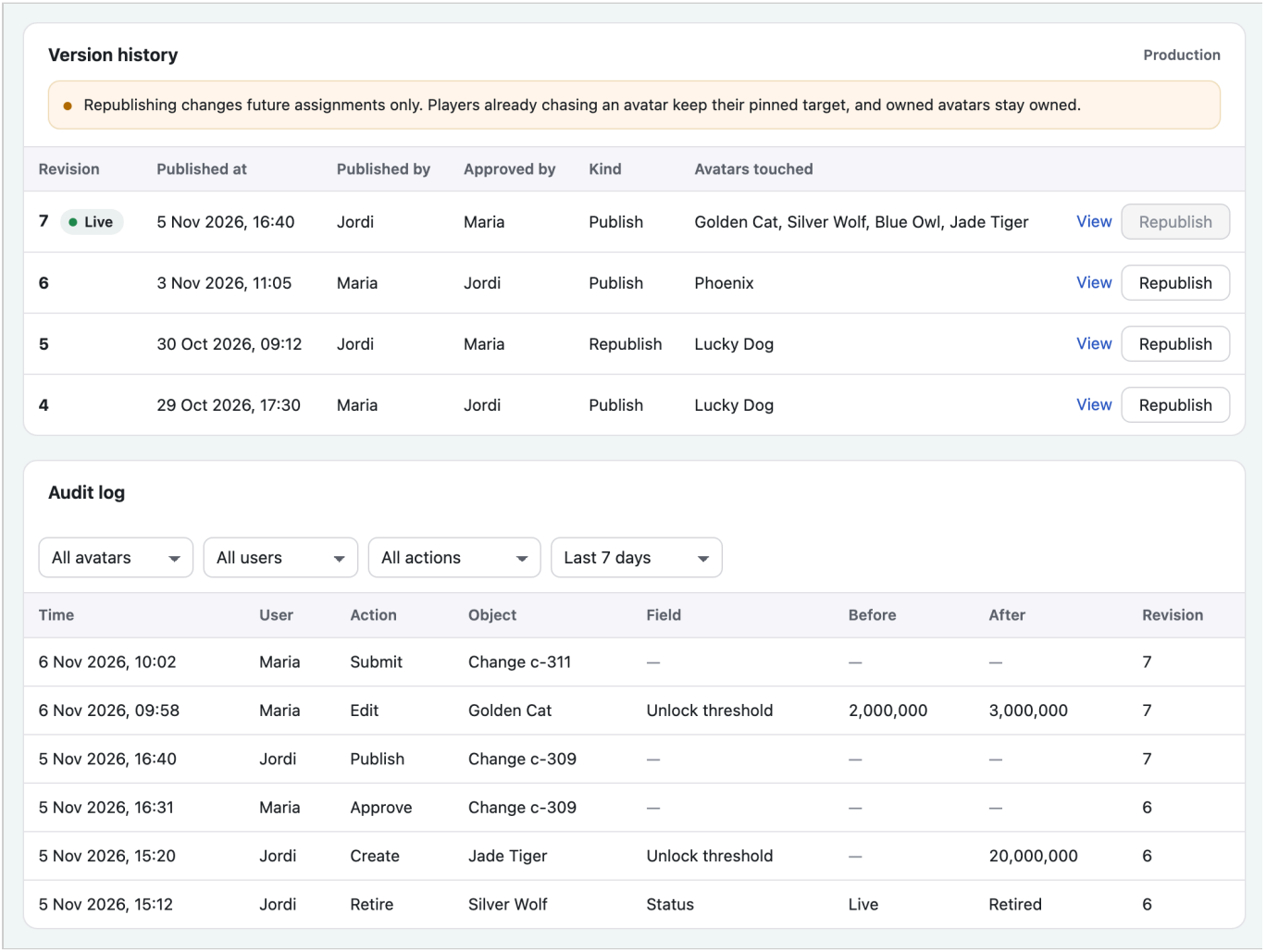
details: SELECTION_CONFLICT → equipped {avatarId, selectionVersion}; VALIDATION_FAILED → a list of {avatarId, field, reason}; other codes have none.

Retries. (1) Settled spins are keyed by (gameId, settledSpinId); a second copy changes nothing. (2) Equip is keyed by selectionVersion: a stale retry that passes the earlier checks returns SELECTION_CONFLICT and the current choice; if it equals the requested avatar, the game treats it as success, otherwise it redraws. (3) Acknowledgements are keyed by noticeId, admin edits by changeVersion. (4) Every admin POST takes an Idempotency-Key; a repeated key returns the first result; only publish creates a revision.

Versions. /v1 changes only for breaking changes; the game ignores unknown fields and renders an unknown equipped avatar as the default.

A2. Extra screens

The version history and the audit log are shown on the review screen, below its diff and approval (Figure 4a, Part 3).



Version history

Production

Republishing changes future assignments only. Players already chasing an avatar keep their pinned target, and owned avatars stay owned.

Revision	Published at	Published by	Approved by	Kind	Avatars touched		
7 Live	5 Nov 2026, 16:40	Jordi	Maria	Publish	Golden Cat, Silver Wolf, Blue Owl, Jade Tiger	View	Republish
6	3 Nov 2026, 11:05	Maria	Jordi	Publish	Phoenix	View	Republish
5	30 Oct 2026, 09:12	Jordi	Maria	Republish	Lucky Dog	View	Republish
4	29 Oct 2026, 17:30	Maria	Jordi	Publish	Lucky Dog	View	Republish

Audit log

All avatars | All users | All actions | Last 7 days

Time	User	Action	Object	Field	Before	After	Revision
6 Nov 2026, 10:02	Maria	Submit	Change c-311	—	—	—	7
6 Nov 2026, 09:58	Maria	Edit	Golden Cat	Unlock threshold	2,000,000	3,000,000	7
5 Nov 2026, 16:40	Jordi	Publish	Change c-309	—	—	—	7
5 Nov 2026, 16:31	Maria	Approve	Change c-309	—	—	—	6
5 Nov 2026, 15:20	Jordi	Create	Jade Tiger	Unlock threshold	—	20,000,000	6
5 Nov 2026, 15:12	Jordi	Retire	Silver Wolf	Status	Live	Retired	6

Figure 4b. Review: history and audit log

Version history (one table per environment, newest first). Columns: `revision`, published at, `publishedBy`, `approvedBy`, kind (publish, republish), avatars touched, and a "View" link to the full revision with its diff against the one before. Each row has **Republish**, which opens a change restoring that revision; it shows the diff against the live revision and goes through the normal review. Republish passes the normal lifecycle validation and does not restore a pulled avatar or reverse its image removal; existing ownership and pinned targets remain unchanged. A banner states the consequence: "Republishing changes future assignments only. Players already chasing an avatar keep their pinned target, and owned avatars stay owned."

Audit log. Columns: time, user, action (create, edit, retire, pull and replace, submit, approve, send back, discard, publish, republish, pause, resume, correction; kill-switch use is audited in the feature-flag tool), object (avatar or change), field, before, after, `revision`. Filters: avatar, user, action, date range. Rows are read-only on the screen. A support lookup of one player's avatars is out of scope for this version.

Labels. The catalogue lists avatars last updated first. A live avatar with an edit under review keeps its Live label and shows "Change in review"; Draft and In review appear as labels only for an avatar not yet published.

Player screens. The picker shows the live earnable avatars and every avatar the player owns; a retired or grant-only avatar appears only to its owners. Only the current targets carry a progress ring, one per mode. Player screens show compact numbers (1.4M / 2M, "Win 2M coins"); the back office and the API keep full numbers. A newly owned avatar shows a NEW badge until the player equips it or saves a choice after seeing it (a display state kept by the game, assumed; no new field). The slot screen shows only the unlock toast with Equip now / Later; progress is shown in the picker. Win rings are amber; bet rings are blue. Tapping a current target repeats its ring in the preview. The Settings icon has no progress ring.

A3. Lifecycle actions and more edge cases

Retire (#5, taking an avatar out; a change item, reviewed). The avatar stays in the catalogue as `retired`; owners keep and can equip it; it is no longer assigned or credited. A Retire applies to spins settled after its revision went live; a spin that settled earlier but is processed later still credits, and can complete, the retired target (as for assignment, Part 2). A player's unfinished `amount` toward it moves to that mode's bar and seeds the next target, with a `progress_moved` notice. The evaluator makes the move in the player's settled order, after the spins settled before the Retire went live; a Pull applies to every spin processed after it commits, so it never completes the pulled avatar.

Pull and replace (rare: legal or offensive content; a change item with a `pullReason` and a live `grant_only` `replacementAvatarId`, reviewed by a second approver like any production change, audited). Once published, the avatar is `retired` with its `pullReason`, it leaves the catalogue for everyone, its image is no longer served and controlled copies are removed through the assumed asset process; clients receiving the revision stop rendering it. ID, name, status, ownership and audit history are kept. After publish, a per-owner step keyed by player, pulled avatar and revision (as the notice is) grants the replacement once (one ownership row per player, so a retry grants nothing more, and an owner who already has it gets nothing more). Every affected owner gets that pull's `replacement` notice, keyed by player, pulled avatar and revision, also when the replacement was already owned; a replacement grant creates the `replacement` notice instead of an `owned` one, and the message does not name the pulled avatar. The same step equips the replacement, with `selectionVersion` + 1, only if the pulled avatar is still equipped at that step, so a newer choice stays. Unfinished progress moves as for Retire. No coin compensation; deletion of offline copies is not promised. For an emergency before the change is approved, the kill switch hides the whole feature.

Rule pause (senior, audited). A pause applies to spins the server processes after it takes effect. A paused rule keeps crediting progress but grants nothing and is not assigned as a new target; the picker greys it with "Paused" (`pausedAvatarIds` in the player state, A1). Grants that fall due during the pause are made at resume against the corrected record; audited corrections run while the rule is paused. At resume, a per-player step keyed by player, avatar and resume makes the due grants, so a retry never grants twice. The step runs in the player's processing order (dependency 1), and each grant is one save, as for a completing spin: ownership, the `owned` notice, the cleared target and the discarded overflow.

Rule changes and removals as players see them (from Part 6):

Case	Server	Player sees
Rule changed mid-chase	Existing pins kept (#3, pinned terms); later assignments use the new revision	Ana keeps 2,000,000; Ben (after revision 8) gets 3,000,000
Golden Cat retired	Owners keep and can equip it (#5, taking an avatar out); progress moves on, `progress_moved` notice	"Golden Cat has left; your 1.2M now counts toward Blue Owl"
Golden Cat pulled and replaced	Image removed; owners get Phoenix, equipped where the cat still is (above)	Owners: "Avatar removed and replaced with Phoenix"; players chasing it: "Your progress now counts toward Blue Owl" (wording from Legal, dependency 5)

Health signals (Part 7's release controls):

Health signal	Action
Spin reply time; notice-read timeouts	Notice read off; stop the rollout
Award lag, backlog	Alert; hold the rollout
Bars at or above their pinned threshold without ownership, outside a rule pause; unprocessed records	Pause the rule, replay, audited correction
Equip errors after a client release	Hold that release
Catalogue or image failures, avatar-screen crashes	Kill switch, fix, switch back on

More edge cases

Case	What happens
A settled spin arrives late or out of order	Only if dependency 1's per-player order is not met: a late spin may credit the target still pinned when it is processed, not its successor; results can differ from settled order by more than one spin; a replay keeps the recorded assignments; the processing record stops a second credit
Carried progress already meets the next target's threshold	That avatar is granted at the assigning spin; the remainder is discarded, as on completion
Every avatar in a mode owned, or none live	No bar; the game shows "all collected" or nothing to earn; the first settled spin after a new avatar goes live assigns it
A cheaper avatar is published mid-chase	It becomes the target at the next assignment, not now
A threshold is lowered	Pinned players keep the old target; no automatic sweep. If that is wrong for players, the correction "apply revision N to players in progress on this rule" only lowers pinned targets; it runs while the rule is paused, so anyone already at or above the lowered target is granted at resume
A mistyped threshold went live	Pause the rule, find the affected pins and grants, approve an audited correction, resume. A rollback fixes future assignments only and reverses no grant
Retire or Pull and replace of the default avatar	Refused: `VALIDATION_FAILED` with `DEFAULT_PROTECTED`
An approved change goes stale because another revision went live	Publish returns `CHANGE_CONFLICT`; the author reapplies the edit on the new revision and it is reviewed again
A jackpot win	Counts in win mode as credited; it completes at most one win-mode avatar
The player spins on two devices at once	Spins are processed one at a time per player, in settled order
An old app version without the avatar screen	Nothing changes for it; progress and ownership are recorded and appear after the update
A player leaves the rollout (rollout reduced)	Their progress and ownership are kept; the flag hides the feature
A guest signs in	Assumed: the account keeps the same player ID, so progress and ownership carry over
A notice is not acknowledged (for example, the app is uninstalled)	It stays pending; fetched ownership remains the authority on the next login