Enjin Governance

MANIFESTO · NOVEMBER 2026

Read from the chain. Prove every text. Show it honestly.



Enjin Governance is a web client for Enjin's on-chain governance: read referenda, vote, file proposals, follow the Fellowship. This manifesto says what the project believes, how it is built and what it does today. Where something is missing, it says so.

CONTENTS

01	What we believe	2	08	How it is built	14
02	The problem	5	09	What it does today	15
03	The idea	6	10	In production	16
04	Reading the chain	7	11	Moderation	18
05	The EGOV1 record	9	12	Security and privacy	19
06	Voting and filing	11	13	How it is made	20
07	The Fellowship	13	14	Limits today	22

Chris Zemmel (Zyric)

01

What we believe

Public money and changes to a blockchain are decided by people who read a text and then sign. A client for that is only as good as the rules it is built by. Enjin Governance follows eight principles. Each one is in the code, and each one has a real case behind it.

June 19, 2026 first commit in the repository, version 1.0 on June 28	**180+** commits on the main line	**15** merged pull requests
1,200+ automated tests, plus browser tests	**since 2026** public at gov.enjin.cloud, with EGOV1 records there from August 2026	**10+** referenda on Enjin Relaychain with an EGOV1 record from this app

1 The chain is the source of truth

What is on chain, the browser reads straight from the chain. The database, file storage and block explorer are caches or extras, never the source.

WHY

An indexer can lag, go down or be wrong. Reading straight from a node shows what holds, with no service standing between voters and the chain. No indexer is needed.

THE CASE

A decided referendum no longer carries its track on chain, so the treasury page counted 0 approved and 0 rejected proposals. Today the server reads the tracks of decided referenda from the archive node and keeps them. If the browser can’t get them either, the page shows “-”, not a zero.

2 The rules come from the runtime

Tracks, deposits, spend limits, lock periods and the way support is measured are read from the chain and tied to the runtime’s version.

WHY

The rules change with every runtime upgrade. Hardcoded values are wrong the day after, and nobody notices. So even a tab left open across an upgrade picks up the new values on its own.

THE CASE

The smallest treasury tier was offered up to 250 ENJ while mainnet allowed 100 ENJ at the time. Such a proposal could only fail at enactment, after the whole vote. Today the limits sit in a table per runtime version, and an unknown version gets a warning.

3 Every text can be checked

Every proposal filed here pins its text to the chain with a hash. Anyone can check whether it was swapped since, even without this app.

WHY

Voters decide on a call, but they read a text. If the text lives only on a server, it can change later and nobody would know. A change should be allowed, but visible.

THE CASE

Referendum #19 on Enjin Relaychain was edited by its proposer on November 4, four minutes after filing. Since then the hash on chain no longer matches the text, and that is exactly what the page shows: “EGOV1 · Edited”.

4 Honest status

The page shows what the chain proves. Estimates carry “≈”. What could not be read says so, instead of showing a zero or an empty list.

WHY

Approved is not enacted, and close is not lost. Whoever votes or expects a payout needs the real state. A false warning does as much harm as a missing one.

THE CASE

Mainnet referendum #12 was enacted a day after it was approved. Today the Enact step stays open until the scheduler has really run the call, with its block and result. And “short on support” now appears only when today’s votes can no longer meet both curves in time.

Proven

Read from the chain, with its block. What is shown is the state.

Estimated

A future moment, computed from the runtime’s rules and marked “≈”.

Unreadable

The source did not answer. The page shows “-” or the reason, never “0”.

5 No signature without notice

Before every signature, the page says what it reserves and costs. Nothing is filed twice, and a button only appears if the runtime will accept the action.

WHY

A signature binds real money: the submission deposit, preimage deposits, fees. Learning that afterward costs trust and sometimes money. Runtime errors are turned into plain sentences for the same reason.

THE CASE

When a proposal landed but its status updates were lost, Retry now finds the referendum it already filed instead of filing a second one with a second deposit. And Refund for a submission deposit appears only after approval or cancellation, since the runtime refuses it otherwise with `BadStatus`.

6 Moderation keeps the space clean, never the chain

Moderators decide what the app shows. They never touch data on chain or the pinned JSON, and every decision appears with its reason in the public log.

WHY

A client that quietly removes content becomes a player in governance itself. So moderation has to be justified, public and limited to what the app displays.

THE CASE

A hidden proposal could still be read, title and text, through two API routes. Today both leave it out and say why. The JSON whose hash is on chain stays public on purpose: otherwise nobody could check the text anymore.

7 Proof against reality

Browser tests run against the production build with real data from the Canary chain. According to the changelog, the runtime's rules were tried on copies of the real chains.

WHY

Mocks only know the bugs you taught them. The bugs that matter happen at the edge of the real chain: rounding, versions, data that disappears.

THE CASE

On mainnet, the app showed about 28 percent too little support. It divided by the total ENJ issuance, while the runtime at the time divided by the active issuance. Today the app picks the denominator by runtime version (chapter 4).

8 Open and easy to hand over

The code is licensed under the AGPL-3.0. Anyone can run their own instance, with their own accounts and none of the operator's credentials.

WHY

A client for public decisions must not depend on one person. The authoritative record, meaning referenda, votes, locks and every EGOV1 envelope, lives on chain anyway.

THE CASE

The handover document in the repository covers the maintainer stepping back: a new instance from the source, starting from an empty database. With a copy of the file storage, old EGOV1 links keep resolving. Without it, the hash on chain remains as proof.

02

The problem

Enjin Relaychain decides on its treasury and on changes to itself with OpenGov: referenda on tracks, votes with conviction, deposits and preimages. It is powerful, and hard to read.

On chain, a referendum is an entry with a hash and a call in encoded bytes. To know what is being voted on, you need the call decoded, the text that goes with it and the rules of its track. Once the referendum is decided, parts of that vanish from the current state again: the tally, the call, the block it was submitted in. What remains is found only by asking an archive node.

The text of a proposal, on the other hand, lives somewhere off chain. Whether it is still the text people voted on can usually not be checked. And the rules themselves move: spend limits, deposits and the way support is measured change with runtime upgrades.

Bytes, not sentences

An encoded call tells voters nothing. Without decoding it, they vote on a label.

Text without proof

If the text lives only on a server, it can change. Nobody sees whether it is still the one that was voted on.

Rules that move

Limits, deposits and measures depend on the runtime. A client with fixed values is wrong after an upgrade.

Signing on the go

Many voters sign on a phone. A request that never reaches the wallet looks like a wallet that doesn’t respond.

Money with deadlines

Deposits, locks and payouts follow their own deadlines. Not knowing them costs money or means waiting in vain.

An open space

Where anyone can post, scams show up too. Moderating must never mean quietly stepping into governance.

03

The idea

Enjin Governance rests on two ideas. First: what is on chain, the browser reads from the chain. Second: what doesn't fit on chain, the text of a proposal, gets pinned to the chain with a hash.

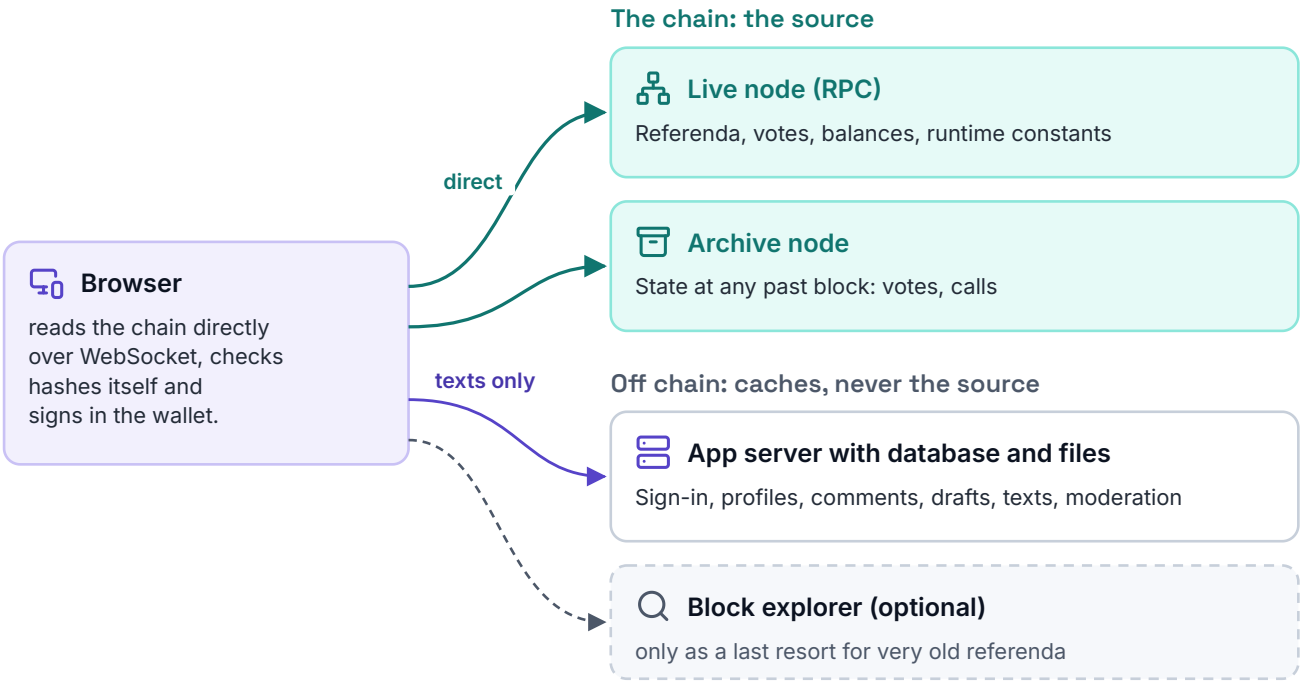


Fig. 1 Where the data comes from. The browser reads referenda, votes and rules straight from the live node and the archive node. The app server holds only what doesn't belong on chain and is never the source for the chain's state. A block explorer is an optional last resort.

The browser connects over WebSocket to an Enjin Relaychain node and, for past state, to an archive node. The list, the detail page, voting and the treasury are built from that. The app server reads the chain only before it acts itself, for example to check whether a draft really belongs to a referendum.

Then there is what the chain doesn't know: sign-in by signature, profiles, comments, drafts, uploaded images and PDFs, moderation. For those the app has a Postgres database and S3-compatible file storage. If both go down, reading and voting still work. With no settings at all, the app starts with public nodes and browser-extension wallets.

The network can be switched inside the app, with no redeploy: Enjin Relaychain or Canary Relaychain, the test network. A link with `?network=` opens on the network it names. A wallet session only ever covers one network.

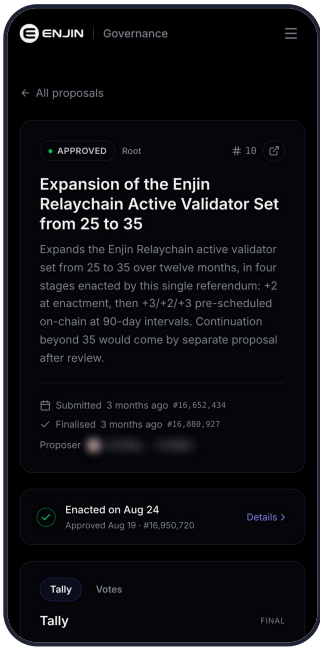
04

Reading the chain

An ongoing referendum is easy to read. The hard part comes after: the runtime drops the tally, the call and the submission block from the current state. Enjin Governance brings them back from the archive.

For a decided referendum, the app reads the state at the block before the decision. That keeps voters visible even if they removed their vote later. It decodes the call from the preimage store, live or, once the runtime has dropped the preimage after enactment, from the archive. Small calls sit inside the referendum itself and are decoded with the runtime of the block they were read at, since a later upgrade can renumber calls.

Every vote comes from the chain's voting storage, with its currency: ENJ, or sENJ from a staking pool. A chart shows each vote as a segment in proportion to its weight, along with the spread across conviction levels: one large holder or many small ones, at a glance.



A referendum on mainnet

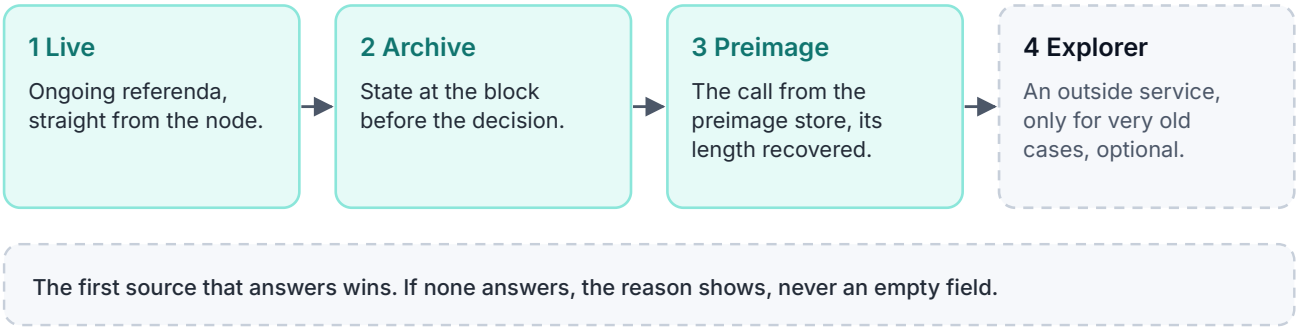


Fig. 2 The order of sources for a referendum's call. An explorer is used only when the chain and the archive return nothing. On Canary, the app works without one entirely.

The lifecycle

Every referendum goes through Prepare, Decide, Confirm and Enact, and treasury spends add a Payout. The card on the detail page gives a date for each step, leading with the time left while anything is still to come. Once everything has happened, it folds to one line such as "Paid out" or "Enacted on", with every event and block behind it.

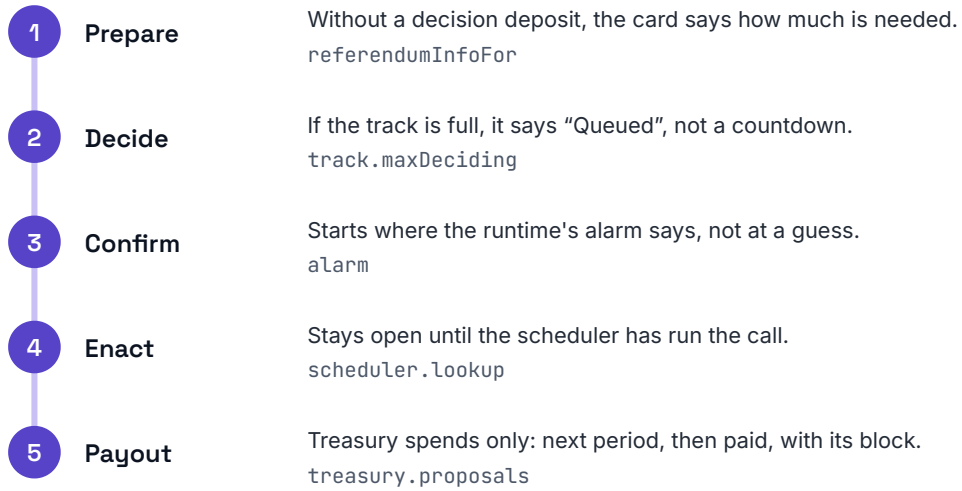


Fig. 3 Five steps, each from its own source on chain. What is still to come is marked as an estimate. What has happened shows with its block and result, and a failed call shows as failed.

How much support is enough

A referendum needs approval and support, both measured against curves that fall over time. The runtime measures support against the ENJ issuance, and which issuance counts changed between two versions: up to version 1070 the active issuance (total minus inactive), from 1080 the total. The app picks the denominator by the version it is connected to, and the chart caption says which one applies.

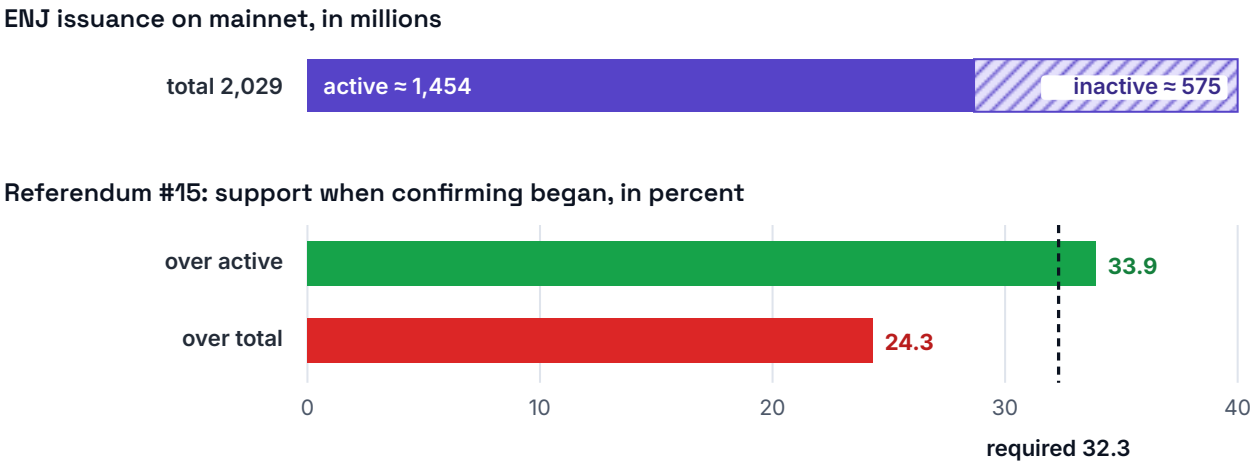


Fig. 4 Why the denominator matters. On mainnet, about a quarter of the issuance was inactive. Referendum #15 started confirming with 33.9 percent of the active issuance against a required 32.3 percent. Measured against the total, the app showed 24.3 percent, a referendum on course looking too weak. Values from the project documentation.

The outlook on the detail page builds on this. Using the runtime’s own rounding, it works out whether and when today’s votes meet both curves before the decision period ends. “On course” says when the referendum would pass if the votes hold. The yellow warning is kept for referenda that can’t make it, with what is missing and by when.

05

The EGOV1 record

EGOV1 is a small, open format. It binds a proposal's text to its referendum, so anyone can find and check it without this app.

The text of a proposal is a JSON file in the app's file storage, served from the app's own address. Only a short envelope goes on chain: `EGOV1:{"u":"…","h":"…"}`, with the JSON's URL `u` and its SHA-256 `h`. The hash is taken over a fixed form, with keys sorted at every level, so every program computes the same value.

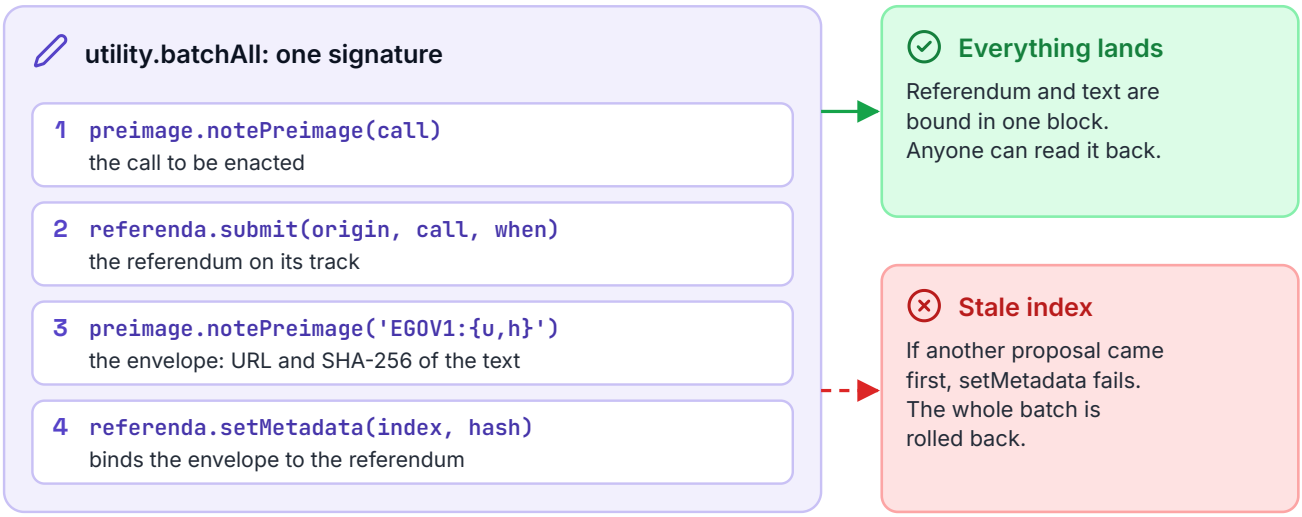


Fig. 5 One proposal, one signature. The call, the referendum, the envelope and the link to it go on chain in one batch. If the precomputed index is stale, the runtime refuses the link and the whole batch is rolled back. A stale index can never label someone else's referendum.

The runtime accepts the `setMetadata` link only from whoever filed the referendum, and only while it is ongoing. If another proposal gets in first, the batch fails as a whole. The app then rebuilds it and tries again. Older proposals from before this link carry the same envelope in a `system.remark` inside their batch.

How anyone can check

To check a text, read the referendum's metadata hash, resolve the hash through the preimage store to the envelope, fetch the JSON at `u` and compare its SHA-256 with `h`. That works with any node, in any browser, from any script. An indexer, an explorer or another client can build the whole corpus of proposal texts this way.

The app shows the result as a badge on the text. If a proposer edits the text of an ongoing referendum, the JSON is rewritten at the same address. Its hash then no longer matches the envelope on chain. That mismatch is the public sign of an edit. The call, the amount and the beneficiary can't change.

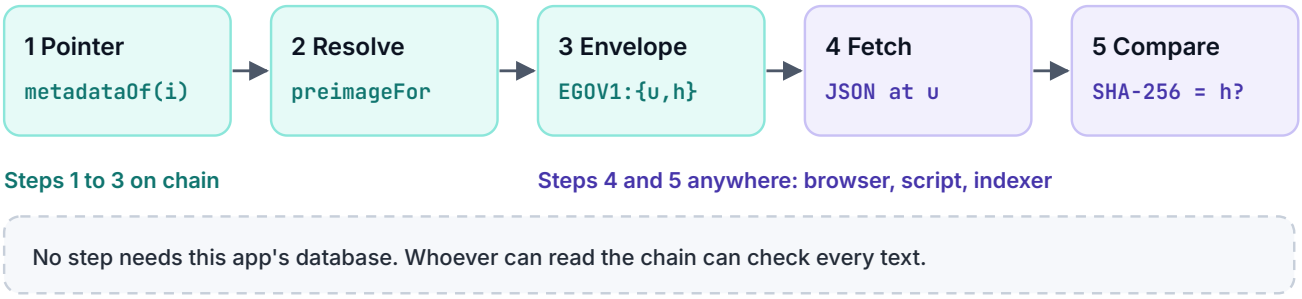


Fig. 6 Five steps from a referendum to a checked text. The first three are read from the chain; the last two anyone can run.

Verified
The fetched text has the hash the app checked against the chain when it linked the proposal.

Edited
The proposer changed it after filing. The mismatch with the chain is visible on purpose.

Unverified
The text doesn't match and no edit is recorded. It is shown, but marked as unproven.

```
{
  "schema": "enjin-governance-proposal", "version": "1.2.0",
  "network": "enjin-relay", "proposer": "en…", "track": "Root",
  "title": "…", "summary": "…", "body_markdown": "…",
  "attachments": [ { "name", "size", "content_type", "sha256" } ],
  "call": { "section": "system", "method": "authorizeUpgrade", "code_hash": "0x…" },
  "enactment": { … when the call runs … }, "preimage_hash": "0x…"
}
```

Fig. 7 An EGOV1 JSON, shortened. Version 1.1.0 describes treasury proposals. 1.2.0 adds the `call` and `enactment` sections, so upgrades, cancellations and other calls are documented next to their text. Readers of 1.1.0 keep working. The size, type and hash of each attachment are compared with the stored file when the draft is staged.

06

Voting and filing

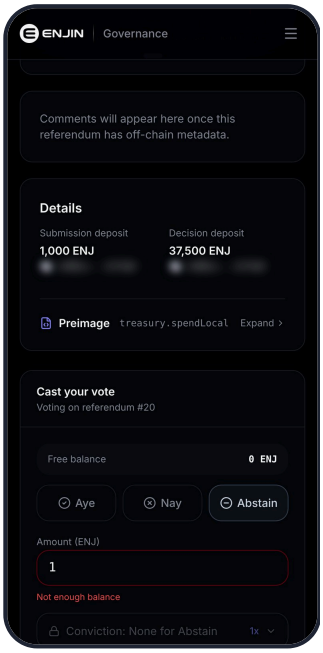
Reading needs no wallet. To vote or file, you connect a wallet and sign. Every signature goes through the same dialog, and each one says up front what it costs and locks.

Voting

Votes are Aye, Nay or Abstain. Conviction multiplies a vote's weight in exchange for a lock: with no lock it counts one tenth, and with longer locks from one to six times. The app reads the lock length from the runtime. Abstain counts toward support, not approval, and locks nothing beyond the referendum.

Votes can be cast with ENJ or with sENJ from a staking pool. The chain counts sENJ one to one like ENJ, and the vote panel says so, along with the fact that locked sENJ keeps earning rewards but can't be unbonded until the lock ends. Voting power is shown in ENJ with the math below it, so a 6x vote doesn't read as six times the balance.

Votes can be delegated per track or across all tracks at once, in one signature. On the account page the app frees expired locks and refunds deposits, but only the ones the runtime will refund.



Abstain, test wallet with no balance

Six wallets, one signing flow

Supported are Enjin Wallet over WalletConnect, generic WalletConnect and four browser extensions for Polkadot wallets. If a wallet holds several accounts, the app asks which one to use instead of quietly taking the first.

On a phone, getting into the wallet was the hardest part. iOS pauses a page in the background, even in the middle of sending. So "Open in Enjin Wallet" opens the wallet only once the relay has confirmed the request, and a second tap re-sends the same request, never a new one. Coming back to the page replaces the relay connection, which iOS lets die in the background.

Filing

Treasury proposals are made in a three-step wizard. It picks the smallest tier whose limit covers the amount, warns when the treasury holds less than requested and checks that a beneficiary address belongs to the right network. An address from another network is caught and converted only after asking. A second composer files any other call: cancel a referendum, whitelist a call, authorize a runtime upgrade by its code hash, or any encoded call. The code hash of an upgrade is computed in the browser from the file.

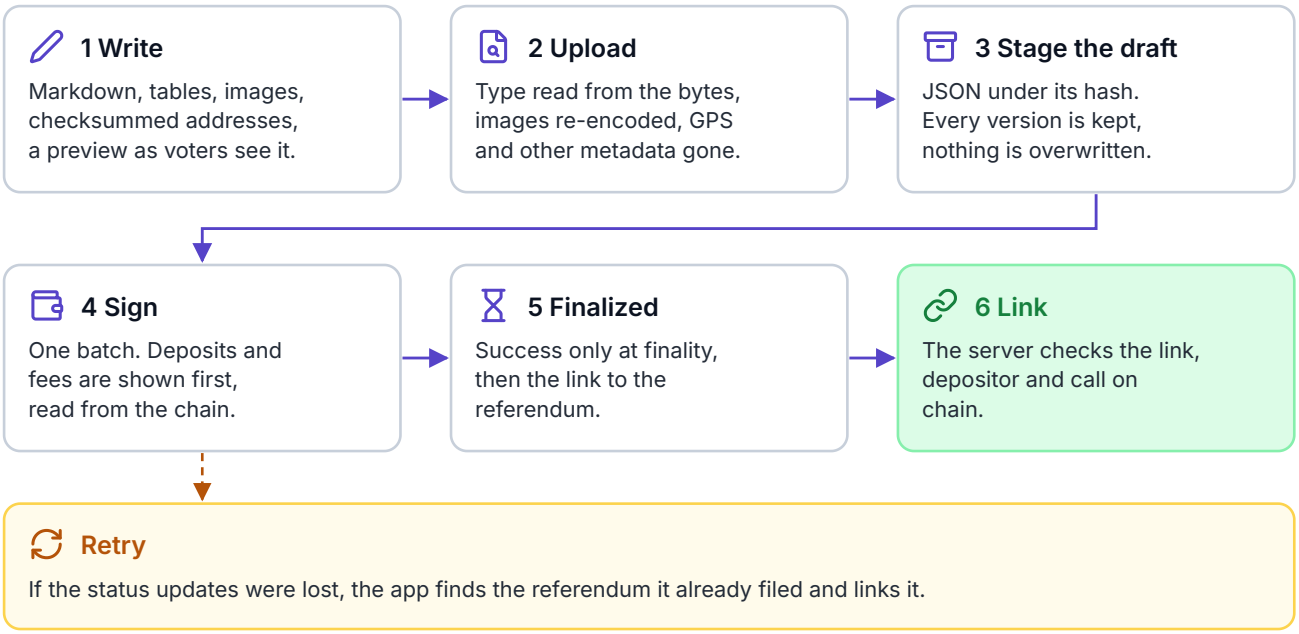
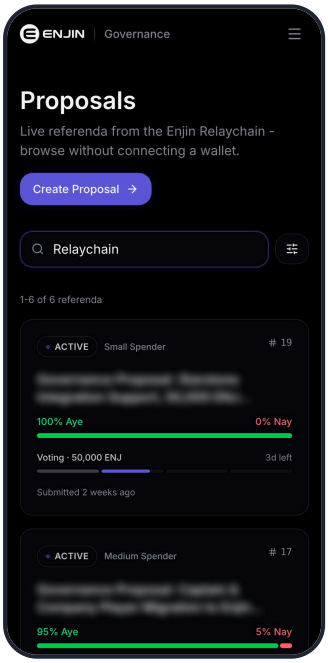
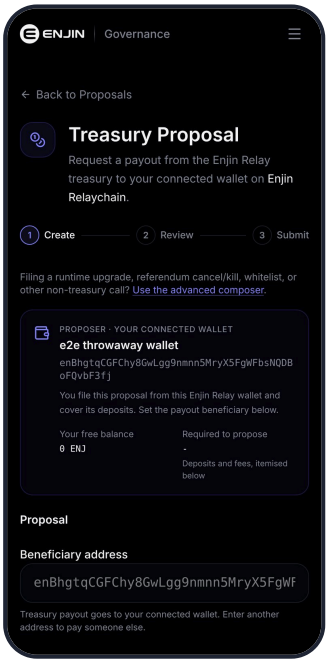


Fig. 8 The path of a proposal. Drafts are stored before anyone signs and can be resumed. Every staged version stays under its own hash, so a batch signed from an older version points at exactly the bytes it pins. Only the check on chain links draft and referendum.

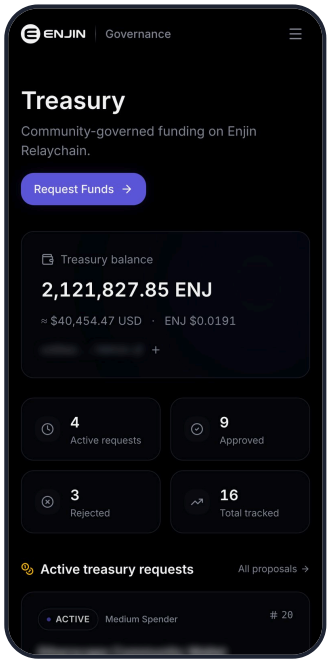
Before signing, the app lists what the proposal reserves: the submission deposit, the preimage deposits for the call and the EGOV1 record, the fees and the minimum balance, all read from the chain. It only goes on if the free balance covers them. The track's decision deposit is shown but not required: anyone can place it later.



Referenda, readable without a wallet



Treasury proposal, step 1



Treasury

On a phone: reading and searching referenda without a wallet, starting a treasury proposal, the treasury with its open requests. Screenshots of the real app on Enjin Relaychain. Left and right, the public instance gov.enjin.cloud; middle, a local production build with a test wallet that holds nothing and never signs. Other people's addresses and proposal titles blurred.

07

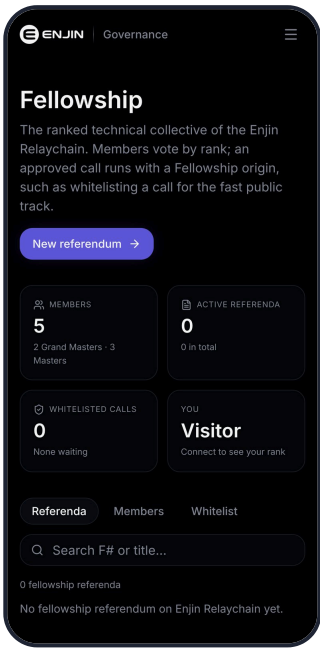
The Fellowship

The Fellowship is the Relaychain's ranked collective. Its members vote weighted by rank and can whitelist calls, which then run through a faster public track.

The `/fellowship` page shows members and ranks, every Fellowship referendum and the calls on the whitelist. It appears only where the runtime has the pallets. Fellowship referenda are numbered separately as F#N, so #7 and F#7 are never confused. In the database, too, an index is unique only per network and pallet.

A member of rank r on a track with minimum rank m counts with weight (e+1)(e+2)/2, where e = r - m. Approval is measured by weight, support by the number of members voting Aye. A member below the minimum rank is told which rank it would take.

Rank above the minimum (e)	Weight
0	1
1	3
2	6
3	10



The Fellowship



Fig. 9 The fast track. A Fellowship referendum whitelists a call, and a public referendum on the WhitelistedCaller track dispatches it. Both pages show the same four steps, link each other by the call hash and compare both tracks' terms with Root's.

Members of rank I and up file Fellowship referenda in the same composer as every other proposal, with a title, summary, text and EGOV1 record: whitelist a call or remove it, add, promote, demote or remove a member. The origin a change needs is named below the dropdown and turns amber when another one is picked.

08

How it is built

A Next.js application in TypeScript, with a few flat layers and clear ownership. A new layer gets added only when there is no other way.

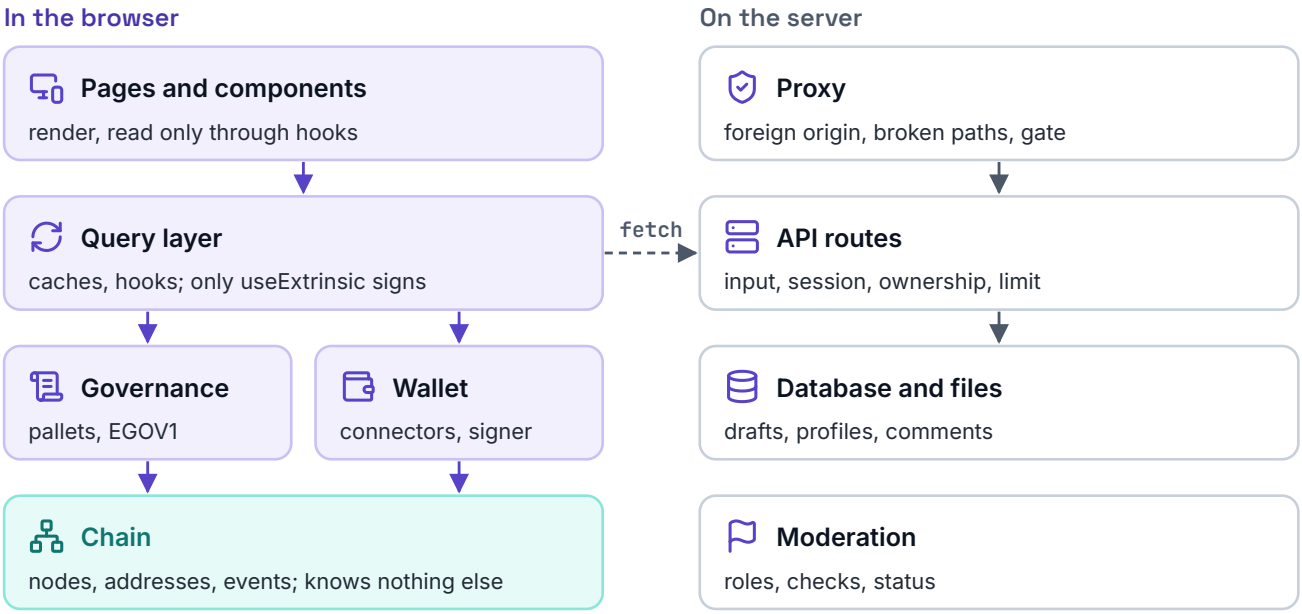


Fig. 10 The browser on the left, the server on the right. The chain layer at the bottom knows nothing about governance or storage. Pages read only through hooks. Signing happens in exactly one place. The server talks to the chain only to check before it acts.

Rules in the design

- **One place signs.** Components hand over a built call. Signing, sending and tracking to finality live in a single hook.
- **Server code stays on the server.** Modules that touch the database, storage or a secret are marked. Importing one into the browser breaks the build.
- **One source per truth.** The networks live in one file, the environment variables in one typed schema, the upload limit in one place for browser and server.

Parts

- Next.js 16 with the App Router, React 19, TypeScript in strict mode
- polkadot.js for the chain, behind a layer of its own
- WalletConnect and the browser-extension interface
- Postgres and S3-compatible file storage off chain
- Vitest, PGlite and Playwright for tests at three levels
- About 340 source files of its own, seven documents on architecture, flows and operations

The server checks every write in a fixed order: same origin, input shape, a signed-in user, ownership by public key, a moderation posting pause and a rate limit. When a setting is missing, the affected routes answer with an error instead of doing half the job. A status page for admins shows after every deploy what is set up and what is missing.

09

What it does today

Every capability is listed with its status. live means shipped on the main line and usable in the public instance at gov.enjin.cloud.

Area	What it does	Status
Reading referenda	List, search and filters; a detail page with call, tally, voters, lifecycle and outlook, all from the chain, decided referenda included.	live
Voting	Aye, Nay and Abstain with conviction, with ENJ or sENJ, voting power with its math.	live
Delegating and unlocking	Delegation per track or across all, freeing locks, refunding deposits where the runtime allows.	live
Treasury proposals	A three-step wizard, tier by amount, costs before signing, resumable drafts.	live
Any other call	Cancel, whitelist, upgrade by code hash, remark or encoded call, under every admin origin.	live
EGOV1	Text, attachments and call pinned to the chain by hash, checked in the browser, edits visible.	live
Proposal text	Markdown with tables, images and PDFs, checksummed addresses, a preview as voters see it.	live
Fellowship	Members, ranks, F# referenda, weighted voting, filing, the fast track.	live
Wallets	Six wallets, one flow for every signature, Enjin Wallet in one tap.	live
Community	Sign-in by signature with no fee, profiles, comments per proposal.	live
Moderation	Reports, one queue, roles by wallet, a public log, optional automatic checks.	live
Operations	A status page for admins, backups as one archive with a restore script, a form for security reports.	live
Search	A sitemap with every referendum, a share image per referendum, per-page metadata, private areas not indexed.	live
Legal	Imprint, privacy policy and terms, a link to the source code in every footer.	live

The two Matrixchains are in the network list but not enabled, so they are not in this table (chapter 14).

10

In production

Enjin Governance is not a prototype. The public instance runs at gov.enjin.cloud, and its EGOV1 records sit on Enjin Relaychain, where anyone can check them.

The public instance runs at gov.enjin.cloud, and its sitemap lists every referendum of the default network. The first public version came out on June 28, 2026, version 2.0 on November 11. The app follows the rules of whichever runtime version is active: it reads them from the chain and picks up an upgrade without being redeployed.



Fig. 11 The first 21 referenda on Enjin Relaychain, read straight from the chain. From #10 on, each carries a metadata hash, a link to its text, and all eleven texts are held by the app. For seven, the SHA-256 of the text matches the hash on chain. Four were edited after filing.

An edit rewrites the JSON at the same address, while the hash on chain keeps pointing at the first version (chapter 5). For three of the four edited referenda (#11, #14, #15), the original envelope was also removed from chain, so only the hash of the first version remains there.

1 The first record

August 17, 2026

On August 17, the link through `setMetadata` replaced the older route of a remark in the batch. The same day, referendum #10 became the first proposal on mainnet to pin its text this way. Its text is still at the same address today, and its hash matches.

2 The visible edit

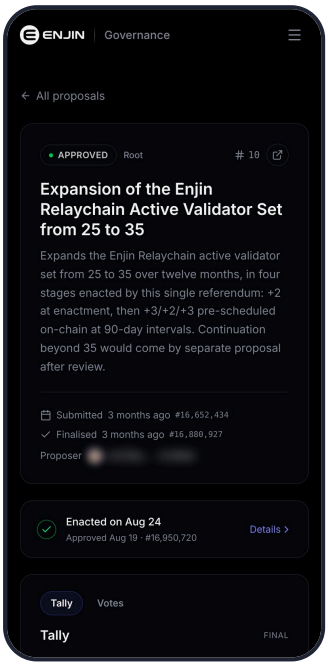
November 4, 2026

Referendum #19 was edited by its proposer four minutes after filing. The app recorded the edit with its time and shows “Edited”. The hash on chain still points at the first version. Anyone comparing the two sees that something changed, without having to trust the app.

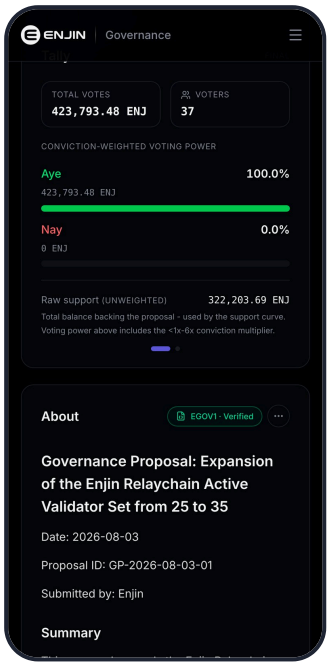
3 The upgrade to 1080

Version 2.0, November 11, 2026

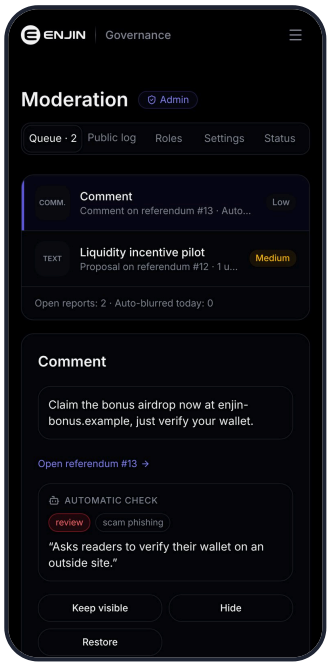
Version 1080 changed spend limits, deposits and the way support is measured. According to the changelog, treasury tiers, support, refunds, Abstain and the Fellowship were run again on a copy of mainnet on 1080.



A referendum on mainnet



Proposal text, EGOV1 verified



Moderation: one queue

Referendum #10 at gov.enjin.cloud, its checked text and the moderation queue. Only moderators see the queue, so it comes from a local production build on Enjin Relaychain and shows the repository's test data. Other people's addresses blurred.

11

Moderation

An open space for proposals and comments also draws scams, such as images showing a readable recovery phrase or links to fake wallets. Moderation keeps it clean without stepping into governance.

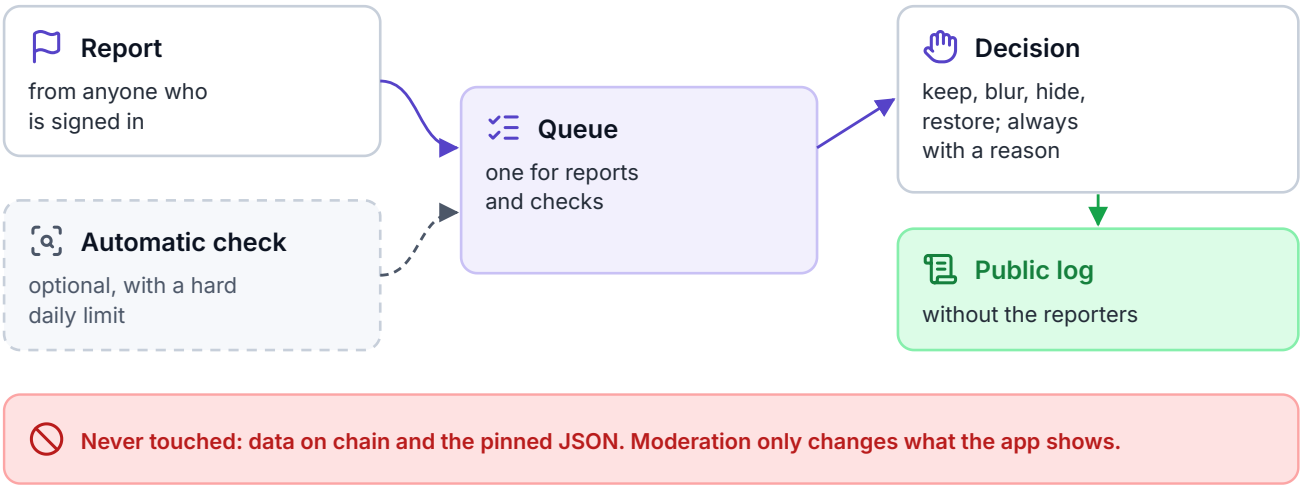


Fig. 12 Reports and automatic checks land in one queue. Moderators decide with a reason, and the decision is public. The chain and the pinned JSON stay untouched.

	Everyone	Moderator	Admin
Report content, read the public log	yes	yes	yes
Queue: keep, blur, hide, restore		yes	yes
Delete a file, pause a wallet's posting			yes
Roles, check settings, status page, backups			yes

Roles are tied to a wallet's public key, so they hold in every address format. The first admins are set in the configuration and can't be removed in the app. Every role is checked on the server, never only in the interface.

Automatic checks, with limits

Admins can switch on checks by a language model. Images and PDFs are checked before they are stored: a clear violation is rejected or, if the admins choose, held; a borderline file is stored blurred and served only after a decision. Proposal text and comments are checked after posting. Text can only be put in the queue, never hidden automatically. Every check is counted before it is sent, so a daily limit is a hard cap on cost. If the checking service is down, nothing is blocked. A setup problem such as a refused key shows on the status page and is reported to the moderators once a day.

12

Security and privacy

The app holds no money and no keys. Signing happens in the wallet. What it has to protect are sessions, drafts, uploaded files and moderation decisions.

Area	How it works today
Sign-in	By signature, with no password and no transaction. The server issues a single-use nonce and stores the exact message to be signed. Verifying uses up the nonce in the same step.
Sessions	An httpOnly cookie. The database knows only the SHA-256 of the token, never the token. Signing out deletes the session.
Ownership	Routes compare public keys, so the same wallet stays the same in every address format and no other one matches.
Other sites	Any write to the API whose origin names another site is refused. Write routes and anonymous routes are rate-limited.
Uploads	The type is read from the bytes, not from the browser's claim. Images are re-encoded and metadata such as GPS is removed. SVG and HTML are never accepted. Files leave only through an app route that respects moderation.
Rendering	Markdown becomes React elements, never injected HTML. Links must be http(s) or paths on the same site. A Content Security Policy and framing protection come on top.
Secrets	Server and browser variables are separate in the schema; no secret reaches the browser. The status page reports only whether something is set, never values or error text.
Deposits	The call of an ongoing referendum is never offered for reclaim. If the checks behind it fail, the list doesn't load at all rather than offering anything unchecked.
Backups	Only admins create and download them, through a short-lived signed link. Sessions and nonces are never included.
Disclosures	A public form for security reports, no account needed. The IP address is stored only as a hash.

According to the changelog, the project has been through several security reviews, one of them before version 2.0. Among other things, that one closed a crafted link that could have handed a proposer any call under a harmless title: links to the composer now choose only the network and the chamber. 18 test files run the real routes against attacks, replacing only the database, storage and outside services. There has been no external review (chapter 14).

13

How it is made

The repository's history starts on June 19, 2026, with version 1.0 as its first commit. In the first five months more than 180 commits and 15 merged pull requests had landed on the main line.

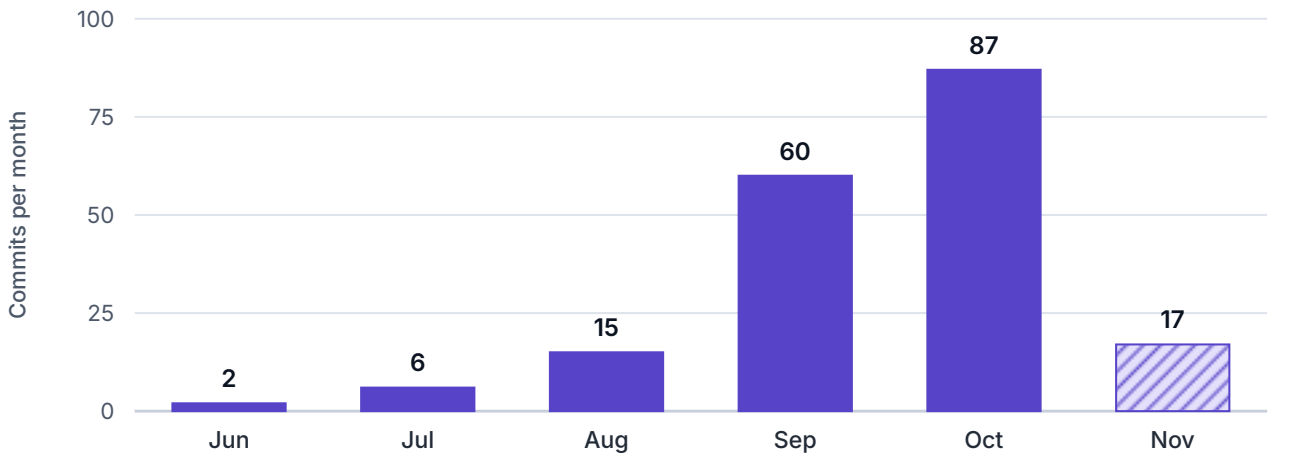


Fig. 13 Commits on the main line per month, to scale. The last month is hatched because it was still running when this was written. Whatever came before the first commit is not in the repository. The number counts steps, not value.

June 28, 2026
Version 1.0: reading, voting, delegation, treasury proposals with EGOV1

August 17, 2026
EGOV1 through `setMetadata`, first record on mainnet

October 8, 2026
Upgrades by code hash, treasury tiers from the runtime

November 11, 2026
Version 2.0: Fellowship, moderation, the rules of 1080

November 13, 2026
Version 2.0.3

The project is built and maintained by Chris Zemmel. Other developers contribute now and then: about one commit in twelve comes from one other person.

The pipeline

Every pull request and every push to the main line goes through the same steps: type check, lint, a search for unused code, a check of dependencies for known vulnerabilities, the tests and the build, then the browser tests against the production build. The dependency check also runs every Monday, because vulnerabilities get published without anyone changing anything.

- **Unit and route tests.** More than 1,200 tests in more than 100 files.
- **Database tests.** The SQL runs against a real in-process Postgres, with every migration production gets.
- **Browser tests.** 29 tests in Chromium against the production build. The chain data is real and comes from the Canary network. The test wallet never signs.

Bugs that became rules

Case	What happened	The rule since
The double lock	On most tracks, the app showed twice a vote’s real lock period.	Lock periods come from the runtime, never from an assumption.
The tier too large	The smallest treasury tier was offered up to 250 ENJ; 100 was allowed.	Limits per runtime version, a warning for an unknown version.
The wrong denominator	Support on mainnet looked about 28 percent too low.	Every value from runtime constants is tied to the runtime’s version.
The empty wallet	On iOS the wallet opened before the request reached the relay, and found nothing.	First the relay’s acknowledgment, then the wallet. A second tap re-sends the same request.
The second proposal	After lost status updates, Retry would have filed a second time.	Before any new attempt, look for the referendum already filed.
The hidden text	A hidden proposal could still be read through the API.	Hidden means hidden everywhere, except the JSON whose hash is on chain.

14

Limits today

A manifesto that hides its limits deserves no trust.

A one-person project

One person builds and maintains the app, with occasional help from others. If that person drops out, development stops. The code stays open, and anyone can run an instance.

Only the Relaychains

The two Matrixchains are registered but not enabled. Enjin Matrixchain still uses an older governance module that the app doesn't cover.

Hosted services

The public instance runs on providers for hosting, database and file storage. If they go down, texts, profiles and comments are missing. Reading and voting depend only on the chain.

Public nodes

Without a node of its own, the app reads through public endpoints. If they are slow or down, the app shows the error and tries again.

The badge checks against the app

"Verified" compares with the hash the server checked against the chain and stored when it linked the proposal. The browser doesn't reread the envelope on chain for it. The independent check from chapter 5 stays open to anyone.

No external audit

The security reviews are described in the changelog and backed by tests. A review by a third party is still to come. The automatic content check is an aid, not a guarantee.

WHAT IS NOT VERIFIED

"Not verified" is a valid status. These points are in the repository but were not checked against reality for this manifesto:

- The runs on copies of mainnet and the Canary network are described in the changelog. No script that repeats them is in the repository.
- The numbers in the denominator case (inactive issuance, referendum #15) come from the project documentation, not from a measurement of our own.
- The full browser test suite was not run for this manifesto. The screenshots come from gov.enjin.cloud; the wallet and moderation views from a local production build on Enjin Relaychain.
- The public instance's uptime is not measured.
- For #11, #14 and #15 it is proven that the original envelope is no longer on chain. How it was removed was not checked one by one.

CONTACT

Chris Zemmel Zyric · gov.enjin.cloud · github.com/chriszemmel/enjin-governance

Enjin Governance is free software under the AGPL-3.0.