IUNUA / YOUR PRACTICAL GUIDE

IUNUA System & Launch Guide

Version 1.3 - Working Guide - 10 September 2026

What we have, what it can do, what it costs and what we can change.

This practical guide is for you as the IUNUA project owner. It explains the token, wallet and wider trust system together: which functions exist today, which functions need more work, how a mainnet launch could proceed, what to budget and what can still be corrected afterward. It is a separate companion to the project whitepaper. The 10 September revision specifies the intended World-authenticated XMR treasury payment and Solana audit receipt, with explicit verification, privacy and recovery boundaries. Native Apple source now shows iPhone Swap, public-address QR scan and on-device IUNUA service records with a separate value rail; production fills stay locked.

The launch specification and cost scenarios are planning tools, not approved issuance or a production-readiness certificate. The proposed initial supply, custody and launch funding still need decisions. No mainnet transaction was submitted.

Read the matching website edition

A practical companion for your decisions

Use the capability tables to understand the system, the budget chapter to compare launch scopes, and the changeability map to plan for mistakes. The formal project whitepaper remains a separate document.

In this guide

01 / IUNUA SYSTEM & LAUNCH GUIDE

The launch in one view

A token can launch before the complete trust system, provided the first release makes only promises it can deliver.

IUNUA is the project and proposed token name. The first complete system target is a World-authenticated XMR treasury payment with IUNUA verification and a Solana audit receipt. Creating the IUNUA token is a separate capability and is insufficient to deliver that target.

Three releases with different requirements

Release	What users receive	What must be ready
1. Token issuance	An IUNUA mint and public token accounts on Solana mainnet. Compatible third-party wallets may hold and transfer it after compatibility checks.	Approved supply, mint features, public metadata, secured authorities, issuance review and distribution terms. Does not require the private trust ledger.
2. IUNUA Wallet launch	Our own app can safely receive, sign and confirm mainnet IUNUA transfers.	Separate production accounts, recovery, correct mint/program validation, transaction approval, resilient confirmation, signed releases and exact-package testing.
3. Full trust service	Verified contributions, private accounting, rewards, game entitlements and governed settlement.	Production ledger, verifiers, anti-abuse rules, game reconciliation and reviewed governance/settlement services. These are additional milestones.

Recommended first scope: a contained test-network rehearsal of the authenticated XMR payment and Solana receipt, followed by a separately authorized small production pilot after review. Token redemption, a market price, exchange listing and automatic Devnet migration are not decided.

The first payment system target

World App	Authenticate and separately approve the payment.
Monero treasury	Send the authorized XMR payment.
IUNUA verification	Check payment evidence and reconcile the private record.
Solana receipt	Publish a minimal audit commitment and verifier attestation.

Planned flow. Payment settlement and public receipt publication are separate operations with separate failure recovery.

Sources: IUNUA token and wallet capability status; IUNUA launch preparation: read-only chain evidence

02 / IUNUA SYSTEM & LAUNCH GUIDE

Current implementation: wallet apps and network support

Wallet apps are available. The current IUNUA token runs on Solana Devnet; the full trust system is not yet a production service.

Implementation evidence remains dated: wallet package review 8 September 2026 and World website preparation/configuration review 9 September 2026. Native Apple wallet source reviewed 10 September 2026 adds iPhone navigation, a public-address QR scanner, locked Swap estimates and on-device agreement composition. Those source changes do not silently upgrade the published 0.8.4 archives. The complete World-authenticated XMR treasury payment with IUNUA verification and a Solana receipt is Planned. This document revision does not add production signing, deploy a receipt program, fund a treasury or establish an end-to-end production payment test.

The wallet is installed software with implemented account and asset features. It is distinct from the website simulation, which uses modeled balances. The current network restrictions describe which actions the wallet can execute; they do not mean the whole application is imaginary or that every displayed asset is a test asset. Public holdings and market estimates are read-only where indicated.

How to read status labels

Label	Meaning
Available	Present in the reviewed release; network access, fees or ordinary account setup may still be needed.

How to read status labels	
Label	**Meaning**
Requires setup	Depends on configured services, credentials or funded test accounts; not certified live on every network.
Simulation	Uses modeled records or demonstration units, separate from chain balances.
Planned	Unfinished capability or target design.
Locked	Intentionally disabled; a visible screen or estimate does not enable execution.

Native macOS and Electron: reviewed 8 September 2026		
Capability	**Native macOS 0.8.4**	**Electron 0.8.4**
Solana Devnet balances and transfers	Available: Keychain-backed test wallet.	Available: reviewed transfers and account-bound receipts.
Assets and named accounts	Available: Solana accounts and public holdings.	Available: accounts and holdings across supported test networks.
Existing test key imports	Planned: no equivalent Electron import workflow.	Available: network-specific imports; no shared universal seed.
Monero Stagenet	Planned: limited native interface.	Requires setup: node and wallet RPC.
World Chain Sepolia	Planned: no equivalent creation/signing flow.	Requires setup: test account and TEST ETH; TEST WLD unverified.
RENDER public holdings	Planned: guidance only.	Available: read-only address lookup.
Market estimates	Available: informational estimates.	Available: informational estimates.
Executable swaps	Locked.	Locked: no approved executable route.
World ID eligibility	Requires setup: staging guidance; production verification unfinished.	Requires setup: staging/backend configuration; production verification unfinished.
Private trust loop and Pantheon	Simulation.	Simulation.
Complete game payments	Planned.	Planned.
Production signing	Locked.	Locked.

This matrix summarizes package/source review, not a new end-to-end network certification. A Solana account, a Monero wallet and an EVM account are different custody systems. Import and recovery support must be checked for the particular client and network. Do not assume that one phrase or backup restores every account.

Release and installation boundaries

Both macOS clients are version 0.8.4, but they are different applications. The reviewed Electron release is Developer ID signed, has a stapled notarization ticket and passed Gatekeeper. The native release is Apple Development signed, passes signature verification, and was rejected by Gatekeeper; it is a development distribution. These signing results identify distribution checks, not a financial or protocol security audit.

The iOS download is source version 0.8.4 requiring Xcode and the user's own signing. The browser extension remains version 0.7.0 and requires manual developer installation. The testnet setup kit is version 0.8.4. Exact archive identities are recorded in the whitepaper's release appendix and the linked release manifest; later source changes do not silently upgrade those files.

Native Apple iPhone source, 10 September 2026

The native Apple app is one target for Mac and iPhone. The 10 September source is a later client revision than the reviewed 0.8.4 packages. It does not enable production signing, a Jupiter fill, a hosted fiat partner, an XMR venue or an on-device Monero spend secret.

Native Apple wallet source: reviewed 10 September 2026

Capability	iPhone	Mac
Primary navigation	Available: Wallet, Assets, Scan, Swap and More. Scan is the middle tab. Wallet Lab and Agreements live under More.	Available: sidebar keeps Wallet Lab, Agreements and Swap first-class.
Receive address	Available: generated QR and share of the public Devnet address only.	Available: generated QR and share of the public Devnet address only.
Scan a recipient	Available: camera scan of a public Solana or Monero address. Wrong-network QRs are rejected. A Monero QR is never written to an IUNUA public receipt.	Locked: paste only. No camera in this source.
Solana Devnet IUNUA-T send	Available: Keychain test wallet; two-phone send with a scanned or pasted address.	Available: Keychain test wallet.
Swap estimates	Available: informational XMR, SOL, IUNUA, EUR and USD. IUNUA has no represented market price.	Available: the same informational estimates.
Executable swaps	Locked: no Jupiter fill, hosted fiat checkout or XMR venue.	Locked: no approved executable route.
Agreement composition	Available: on-device receipt with purpose, amount, optional fee note and settlement asset. Optional Devnet IUNUA-T public ref.	Available: the same composition from the sidebar.

Native Apple wallet source: reviewed 10 September 2026		
Capability	**iPhone**	**Mac**
Stagenet XMR value rail	Requires setup: iPhone is a wallet-rpc client. Spend keys stay on the RPC host. The app does not spawn monerod.	Requires setup: owner-started Stagenet daemon 38081 and wallet RPC 38083.
Production signing	Locked.	Locked.

Separate systems, separate limits

The website Trust Lab has seven simulated steps: create an alias, model eligibility, receive a reward, transfer, choose a settlement path, manage disclosure and model an upgrade. Its balances and receipts are local demonstration data. Progress survives internal navigation for the current session and resets on reload or explicit reset. The setup guide stores safe local preferences; it is not proof that the required services were installed or are healthy.

The local Game API, MCP bridge and Electron Wallet Lab now share one file-backed private ledger in the loopback runtime. That ledger is still a simulated local authority: it is not a production wallet, does not hold a private key and does not post a chain transaction. Apple, iOS and browser-extension demo stores remain separate. Newer game-link code in a checkout is not evidence that a downloadable wallet can approve and settle complete game payments.

Electron offers an optional contained local Solana validator workflow requiring Docker and explicit setup. A local node does not join mainnet automatically. Monero Stagenet is the intended treasury test rail: an authorized Stagenet send is the private-value movement, and IUNUA records the authorized debit on its private ledger and, when visibility is public, may send a separate Devnet IUNUA-T public reference. That record is not a public IUNUA mint, not a wrapped-XMR program and not a substitute for a later, separately previewed production receipt. World ID is an eligibility-proof reference, distinct from the World Chain network. RENDER holdings do not provide Render compute jobs, node operation, billing or token transfers. Estimates for established assets are informational; IUNUA has no represented market price.

Sources: IUNUA token and wallet capability status; IUNUA release manifest; IUNUA downloads and installation evidence; IUNUA developer integration guide; IUNUA Trust Lab; World ID documentation: overview

03 / IUNUA SYSTEM & LAUNCH GUIDE

What the token itself can do

The token is a public accounting primitive. Wallets and services supply the surrounding product behavior.

Token functions versus product features

Function	Current Devnet / proposed mainnet	Meaning and limit
Hold, receive and transfer	Devnet available in reviewed wallet paths. Proposed mainnet after issuance and compatible signing.	A token account holds units of one mint. Transfers move units; SOL pays network fees. Recipient account creation may also cost SOL.
Mint additional units	Authority exists on Devnet. Proposed pilot retains controlled mint authority.	Authorized issuance increases supply. 42 million is not a hard cap while this authority remains.
Burn owned units	Standard Token-2022 capability; not certified as an IUNUA wallet button.	A holder or permitted delegate can burn controlled units. Burn reduces supply; it creates no redemption payment.
Freeze one token account	Freeze authority exists on Devnet; proposed pilot capability.	Can block activity for the affected token account and later thaw it. It does not give the issuer ownership of the holder's tokens.
Pause the whole mint	Pausable extension exists and is unpaused on Devnet; proposed pilot capability.	Pause rejects minting, transfers and burning for the mint until resumed. This is an emergency stop, not a transaction reversal.
Update name, symbol and metadata	Update and pointer authorities exist; proposed pilot retains them.	Can correct editable metadata while authority remains. Wallet caches/indexers may take time to update; mint identity remains its address.
Delegate a limited amount	Standard token-account feature, not reviewed as an app workflow.	Holder authorization can permit spending/burning up to an allowance. Delegation is not Pantheon governance.
Automatic fees, hooks or seizure	Not in the observed Devnet extension list; not in this proposed launch draft.	No transfer-tax extension, custom transfer hook or permanent delegate is proposed. New powers must not be implied.

Token functions versus product features

Function	Current Devnet / proposed mainnet	Meaning and limit
Privacy, proof checks, rewards, voting	Separate planned services or simulation.	The current token does not conceal transfer amounts, prove useful work, enforce an allocation schedule or deliver governance votes.

IUNUA is not SOL, XMR, WLD or RENDER. Showing those assets or price estimates in the wallet does not make them backing reserves, allow conversion into IUNUA or provide a cross-chain bridge. A DEX market requires a separate compatible pool and liquidity; creating a token supplies neither demand nor a guaranteed price.

Sources: Solana token operations; Solana pausable mint; Solana token metadata and pointer; Solana documentation: token extensions

04 / IUNUA SYSTEM & LAUNCH GUIDE

Intended architecture: identity, treasury, verification and receipts

World authenticates; Monero settles XMR; IUNUA checks and records; Solana anchors the audit receipt.

Four responsibilities in the proposed system

World / identity	World App wallet-account authentication establishes the application session. World ID is a separate human-credential check where policy requires it.
Monero / funds	XMR is the intended principal treasury asset and payment asset for the initial reference flow. Monero establishes the actual payment state.
IUNUA / controls	A policy-controlled signer authorizes spending. Services verify payment evidence and maintain a private, durable accounting and audit record.
Solana / public receipt	A finalized receipt transaction anchors a commitment and verifier attestation. This records evidence about a payment; it does not move the XMR.

Planned architecture. The World network, Monero network and Solana network do not form one atomic transaction. No deployed bridge or cross-chain proof verifier is established here.

Private accounting and network truth

The private ledger records approved instructions, reservations, fees, receipts and adjustments. It must reconcile with the actual Monero wallet and chain. A database credit does not create spendable XMR, and a ledger state cannot override a failed or unconfirmed network payment. Private database updates can be atomic internally; World authentication, Monero settlement and Solana publication cannot be made atomic by a database transaction.

Authentication, eligibility and spending authority are separate. A World account session identifies an authenticated participant. A human credential, a skill credential, an invoice and evidence that work occurred answer different questions. Each must have a defined purpose, issuer, expiry and policy. An authenticated user receives only the treasury roles explicitly assigned to that account.

Mandatory receipt for the first audited-payment product

For the proposed audited treasury-payment flow, receipt publication is a required completion step disclosed before approval. The amount of information disclosed is minimized by policy. Other future private-ledger activities may have different publication rules; the earlier general idea of optional public settlement must not hide a missing receipt for a payment sold as audited. A Monero payment can be confirmed while its Solana receipt is still pending.

Governance and extensions

Pantheon remains the planned framework for changing spending policy, signer membership, verifier authority and disclosure rules. Proposals, review periods, emergency pauses and controlled activation need an adopted policy. An identity-provider outage may pause new ordinary payment requests; any emergency treasury path must use separate, disclosed authorization and leave an audit record.

The ordinary IUNUA Solana token, the XMR treasury and the receipt mechanism are separate components. Token issuance alone creates neither treasury custody nor payment verification. A receipt can be anchored without transferring an IUNUA token. Any later token utility, redemption, bridge or autonomous on-chain verification requires an additional specification and implementation.

Service record and value rail

IUNUA is a Token-2022 token on Solana. It is not a chain that hosts Monero as contract code. The wallet's test composition records a named service (purpose, amount, optional fee note and settlement asset) on the device and, when visibility is public, may send a Devnet IUNUA-T public reference. The selected value rail is separate: Stagenet XMR through owner-started wallet RPC, or an IUNUA-T-only record. Production SOL and IUNUA fills stay locked. Wrapped XMR, a homemade XMR-SOL bridge and putting a Monero address in the public IUNUA receipt are disallowed.

Wallet test composition

Scan or paste	A public Solana or Monero receive address. The scanner never accepts a key.
IUNUA record	Purpose, amount, optional fee and chosen rail. Optional Devnet IUNUA-T public ref.
Value rail	Stagenet XMR via wallet RPC, or IUNUA-T only. SOL stays locked.
Not a bridge	The two networks stay separate. An on-device agreement is not a legal deed.

Available as a test composition in native Apple source. World App sign-in, mainnet IUNUA and a production XMR payout remain Planned or Locked.

Sources: World App wallet authentication; Monero wallet RPC: payment and reserve proofs; Solana transactions and instruction execution; IUNUA intended architecture

05 / IUNUA SYSTEM & LAUNCH GUIDE

Connect with World: a human check and an account are different

World App account sign-in is required for the intended IUNUA treasury-payment flow; a World ID human check is a distinct policy condition.

World ID can supply a privacy-preserving human credential check for IUNUA. World App wallet authentication separately proves control of a World/EVM account. World Chain is a separate blockchain. None of these automatically creates, moves or signs an IUNUA token on Solana.

For this first payment product, World App wallet-account authentication is a required application step, followed by server-verified account linking and separate payment approval. The prepared website QR flow proves a human credential only and does not satisfy that wallet-account sign-in requirement. The exact desktop/mobile World App handoff and supported authentication transport still need implementation and end-to-end validation.

The intended connection journey

1. Ask in IUNUA	The user starts a human check. A server creates a short-lived request, and the official World IDKit displays its QR code or app link.
2. Approve in World	The user scans and approves the requested credential proof. Staging uses a simulator; production requires the real supported credential.
3. Verify and link	The server validates the proof and original request. Linking a Solana account additionally needs proof of wallet-key ownership and explicit consent.

Integration design. A scanned QR or a successful client callback alone is not authentication, token ownership or payment approval.

World connection: implementation and remaining work

Part	Current position
Website QR flow	Requires setup: official IDKit UI and a server verification/session module are prepared. As reviewed on 9 September 2026, no World app ID, RP ID, signing key or database is configured on the site. No real proof was exercised.
Released wallet apps	Requires setup / Planned: reviewed native and Electron 0.8.4 packages provide staging configuration guidance. They do not include the new website QR flow or a completed World login.
Browser human-check session	Prepared code: nonce, signal, browser, environment and action binding; server verification; replay/expiry rejection; 15-minute session. Activation and end-to-end review remain required.
World App wallet sign-in	Planned and required for the treasury-payment target: World App wallet authentication with server-side verification, account linking and a reviewed desktop/mobile handoff. The prepared QR human check does not implement this.
Link to a Solana wallet	Planned: wallet-created challenge, explicit message signature, secure single-use handoff, linking consent and recovery/unlink rules. A pasted address is insufficient.
Token eligibility and rewards	Planned: application or settlement services enforce approved rules. Human verification grants no automatic IUNUA tokens, payments, signing permission or voting rights.

Activation steps

• Register IUNUA and its relying party in the World Developer Portal. Choose the credential and a dedicated human-check action; decide how repeat login differs from a one-time reward claim.

• Configure app ID, RP ID, action and signing-key secret only on the server. Provision the verification-session database. Begin in staging; keep the site access restriction unchanged.

• Test the official QR flow with the simulator, including success, cancellation, expiry, replay, wrong browser and logout. Confirm the result on the server.

• Complete account-linking and recovery design, rate limits, retention/deletion, monitoring and independent review before production identity use.

• Configure World production credentials and test with an eligible real user. Separately integrate wallet handoff and the required World App wallet-account authentication. Solana mainnet remains a separate release decision.

The prepared reference uses an Orb legacy credential proof through IDKit and World's v4 verification endpoint. It stores only short-lived session records, a normalized nullifier and a hashed browser token; raw proof bodies are not retained. The backend accepts only the configured credential shape and must be updated and tested when the selected protocol or credential changes. Production retention cleanup and abuse controls remain launch requirements.

Privacy limit: linking a person identifier to a Solana address creates an association inside IUNUA. Public Solana accounts and transfers remain visible. World ID does not make ordinary token transfers private, provide skill certification or prove every contribution was valuable.

Sources: World IDKit integration; World React QR widget; World v4 server verification API; World App wallet authentication; World Chain overview

06 / IUNUA SYSTEM & LAUNCH GUIDE

The first complete transaction: sign in, pay, verify, record

Working reference: the recipient receives XMR; IUNUA coordinates the process and publishes a Solana receipt. Token redemption remains undecided.

The reference flow below assumes payment directly in XMR. This is a drafting assumption that makes the requested transaction concrete; it does not decide whether a future IUNUA token will represent a redeemable treasury claim. No actual treasury balance, address, recipient, amount or production transaction has been verified for this revision.

Wallet test composition versus production target

Native Apple source now lets two phones share a receive QR, scan a public address, record an agreement with purpose, amount, optional fee note and settlement asset, optionally send Devnet IUNUA-T as a public service record, and optionally spend Stagenet XMR through owner-started wallet RPC. That is a test composition. It is not a legal deed, not a deployed Monero contract on Solana, and not the production World App to XMR treasury flow specified in this paper.

What the wallet can compose today	
Part	**Required behavior / boundary**
Public service record	IUNUA-T on Solana Devnet may carry purpose, amount and fee as an on-device composition plus an optional explorer signature. The public ref must not contain a Monero address.
Selected value rail	Stagenet XMR via wallet RPC, or IUNUA-T only. Production SOL and IUNUA fills stay locked until mint, pool, signing review and a live venue exist.
iPhone custody	Spend keys stay on the wallet-rpc host. Scanning a QR does not import a key and does not authorize a later spend.
Production target	World App sign-in, a licensed or reviewed XMR payout and a mainnet IUNUA mint remain Planned or Locked. Swap execute stays locked.

An end-to-end payment request	
Part	**Required behavior / boundary**
1. Authenticate	Complete World App wallet authentication and server verification using a fresh challenge. If the adopted policy also requires World ID, verify that credential separately. Bind the resulting session to the IUNUA account.
2. Review	Show the source treasury, Monero network, recipient, XMR amount, estimated fee, purpose and public disclosure policy. Show XMR directly; any fiat estimate must carry its rate and time.
3. Authorize	Require an explicit payment approval bound to the exact instruction and a unique request ID. Enforce role, amount limits, required co-approvals, expiry and available unlocked funds at the signing service.
4. Submit	Reserve the accounting amount and fee budget. Construct, approve and submit the Monero transaction through protected signing infrastructure. Persist the transaction reference before automatic retries can create duplicate payments.
5. Verify settlement	Use Monero wallet/chain observations and suitable transaction proof to verify destination, received amount and confirmation policy. Pending, unknown and failed outcomes remain distinct. Never accept a screenshot or client success flag as settlement.
6. Record evidence	Commit the payment result, actual fees, evidence reference, policy version and verification outcome to a durable private record. Store proofs and identity links under explicit access controls.
7. Publish and display	Publish the approved receipt commitment and attestation on Solana. After its transaction is finalized, show both Monero payment status and Solana receipt status, with the permitted evidence and appropriate explorer references.

A receipt failure must not send the money again

Assign one durable payment ID to the original request and separate IDs to settlement and publication attempts. Reconcile an unknown Monero submission before creating another transfer. If XMR settles but Solana publication fails, retain Payment confirmed / Receipt pending and retry only publication. An idempotency key must prevent duplicate economic effects, not merely duplicate HTTP responses.

Failure and correction behavior

Part	Required behavior / boundary
Authentication or approval fails	Stop before signing. No authorization is inferred from an earlier QR scan or an expired session.
Fee or funds unavailable	Reject or obtain approval for a revised quote. Do not silently change the amount, recipient or fee ceiling.
Submission outcome unknown	Inspect the existing wallet transaction and chain state. Preserve the original request; do not assume failure and resubmit a new payment.
Payment confirmed, receipt unavailable	Retry receipt publication with the same payment identity. Display the partial completion honestly; keep durable evidence available.
Confirmation becomes insufficient	Mark the receipt as superseded or under review through a new append-only record, and apply the documented confirmation/reorganization policy.
Wrong confirmed recipient or amount	A confirmed payment is not undone by editing the ledger or receipt. A refund or compensating transfer is a separately approved transaction; recovery is not guaranteed.

First demonstration acceptance criteria

• Rehearse with isolated Monero Stagenet funds and Solana Devnet receipts before any real treasury payment; use an explicitly labeled World staging flow for that rehearsal.

• Demonstrate one successful authorized payment and a rejected unauthenticated request. Observe the exact wallet build, not just source code.

• Demonstrate timeout recovery, duplicate-request rejection and Solana publication failure after Monero confirmation without a second XMR payment.

• Have an authorized reviewer verify the selected Monero payment proof and recompute the receipt commitment from the disclosed evidence.

• Before any small mainnet pilot, approve custody, networks, real recipient and amount, fees, evidence disclosure, monitoring, recovery and independent review. No production activation is implied by this paper.

Sources: Monero wallet RPC: payment and reserve proofs; Solana transactions and instruction execution; World App wallet authentication

07 / IUNUA SYSTEM & LAUNCH GUIDE

What an on-chain receipt can prove

A public fingerprint proves consistency with disclosed evidence. Payment verification requires the evidence and the verification procedure.

A finalized Solana record establishes that its recorded bytes were included in that chain. A receipt commitment lets a reviewer check that a later-disclosed evidence package matches those bytes. Neither fact by itself establishes that the claimed XMR payment occurred. The initial design relies on identified IUNUA verifier attestations plus independently checkable Monero evidence for authorized reviewers.

Proposed visibility and evidence

Part	Required behavior / boundary
Public Solana receipt	An opaque receipt ID, schema and policy versions, commitment, verifier identity/signature, claimed verification outcome and a supersession reference when applicable. Exact encoding, storage and access rules remain to be implemented.
Restricted evidence	The payment instruction, consent and approvals, Monero transaction reference, destination, received amount, relevant payment proof, confirmation observations, timestamps and actual fee/accounting data.
Kept out of public receipts	World login payloads, person identifiers/nullifiers, treasury view/spend keys, recovery secrets and raw evidence. Monero transaction IDs and amounts should not be public by default; disclosure requires a deliberate policy.
Payment proof	Monero get_tx_proof/check_tx_proof can verify a payment for a supplied destination and report received amount, confirmations and pool state. Recheck against a trustworthy synchronized node and the adopted confirmation policy.
Treasury reserve evidence	A payment receipt does not prove the total reserve or solvency. Monero reserve proofs concern particular funds at verification; liabilities, other obligations, custody and later spending need separate accounting and review.

Commitments, signatures and trust

Specify canonical serialization, schema version, domain separation and a fresh high-entropy random salt before calculating the evidence commitment. Publish only the commitment; retain the salt with the protected evidence. A bare hash of predictable amounts or addresses can invite guessing. The verifier signature must bind the commitment, receipt ID, network, policy version and outcome. A signing key authenticates the verifier statement; it cannot make a false statement true.

The verification service must check that the proof corresponds to the expected request, recipient and amount; distinguish sender/spend evidence from recipient-payment evidence; check confirmation state;

and reject invalid or reused evidence where policy forbids reuse. Linking a valid payment to an unrelated invoice must fail. The service must not claim that a human-credential check proves an invoice was legitimate.

A chosen Solana memo or a dedicated receipt program could anchor the record. A memo alone cannot enforce authorization, uniqueness or verification semantics. A program with approved verifier keys can enforce receipt-submission rules, but still trusts those verifiers for Monero facts unless a separately designed and reviewed cross-chain verification mechanism is implemented. The receipt mechanism has not been selected or deployed.

Disclosure and corrections

Publicly releasing destination, transaction proof and transaction reference can allow others to verify the associated payment and amount, reducing the privacy of that payment. Keep access purpose-bound and disclose only the needed evidence. Revoking access cannot erase copies already obtained. Public receipts should be append-only: corrections point to the prior record and preserve who changed the claim and why.

Sources: Monero wallet RPC: payment and reserve proofs; Monero privacy and wallet keys; Solana transactions and instruction execution

08 / IUNUA SYSTEM & LAUNCH GUIDE

The XMR treasury: control, accounting and recovery

XMR is the intended principal reserve asset. Who owns and can spend it must be settled before funding a production treasury.

The design places the main treasury funds in Monero/XMR. It does not claim that a production treasury already exists or is funded. The recorded Solana Devnet game treasury in this paper is a different test asset/account. A small separate SOL fee budget is needed for public Solana receipts; owning XMR does not pay Solana fees.

Required treasury decisions

Part	Required behavior / boundary
Ownership and custody	Identify the treasury owner, beneficial interests, authorized operators and applicable obligations. Decide whether funds are project-owned, user-owned or held on behalf of users; a shared custody arrangement must not be implied silently.
Spending control	Use a selected, tested signing/custody design with spending limits, independent approvals where required and separation of online coordination from secret keys. Evaluate Monero-specific signing and recovery capabilities; Solana multisig code does not secure XMR.

Required treasury decisions	
Part	**Required behavior / boundary**
World sign-in boundary	The treasury signer must validate the authenticated payment authorization and policy. A login-only screen cannot enforce spending restrictions against another client that holds the spend key. Self-custody users retain external spending ability.
Accounting	Distinguish total and unlocked funds, reserved outgoing payments, fees, liabilities and pending reconciliation. Reconcile outgoing spends with appropriate evidence; a view-only wallet is not automatically a complete outgoing-spend ledger.
Recovery	Test protected backups, authorized recovery, signer replacement, key compromise response and provider outages. Revoke application sessions after account recovery; require separate approval to relink a World account to treasury roles.
Reserve risk	Holding treasury funds in XMR does not create a stable fiat value. Treasury coverage depends on asset price, liquidity, fees and obligations. The reference flow sets no exchange rate or guaranteed conversion.

Human login, identity verification and treasury key custody must remain separate. Never place XMR spend keys, view keys, World signing secrets or recovery material in the public paper, browser storage or on-chain receipts. The iPhone wallet is a client of owner-started Stagenet wallet RPC; it does not hold a Monero spend secret and does not spawn monerod. For a real pilot, expose only a small explicitly approved operating amount after the signing and recovery design has been validated.

Sources: Monero wallet RPC: payment and reserve proofs; Monero privacy and wallet keys; World App wallet authentication

09 / IUNUA SYSTEM & LAUNCH GUIDE

The token today: a public Devnet record

The current IUNUA mint exists on Solana Devnet. Production currency issuance is a separate milestone.

Finalized read-only verification on 2026-09-09T09:24:23.195685+00:00. Devnet mint slot 495559004 and treasury-balance slot 495559005. Mainnet genesis was verified; this Devnet mint address returned no account on mainnet at slot 445576054. That checks this address, not every possible IUNUA-named mint. Faucet health was not reverified.

Recorded Devnet state	
Field	**Snapshot**
Network and program	Solana Devnet; Token-2022
Mint	9UePLTf7PRfbZUDzRq8f4sVo3CE7DsZvEtY7ypYZprrk

Recorded Devnet state	
Field	**Snapshot**
Issued supply	42,000,000 IUNUA (test tokens)
Base units and precision	42000000000000000 base units; 9 decimals
Game treasury token account	8NcU2LK62iNV2ySjHZKgKf72r4yP7tP8dELREZvyr1nW
Game treasury balance	41,999,000 IUNUA (test tokens)
Mint and freeze authority	EixVckoyE92vZTVvA2YYkANenrvJdYDW1rTN5ZP8U43W
Pause state	Unpaused at verification; pause authority retained.
Metadata controls	Metadata pointer and update authorities retained.
Observed extensions	metadataPointer, pausableConfig, tokenMetadata

42 million is issued supply, not an enforced cap. The mint authority is still assigned, so the recorded number must not be advertised as immutable scarcity. The same authority address is recorded for minting, freezing, pausing and metadata control. This snapshot does not establish multisignature custody, decentralized control or governed authority execution.

The game treasury is separate from a faucet reserve. Neither is an individual player balance, and the difference between total issuance and this treasury is not an allocation schedule or proof of adoption. This edition makes no claim that the faucet is funded, reachable or ready to issue tokens.

A Solana mint identifies a token and records shared properties such as supply, decimals and controlling authorities. Creating that mint does not deploy a private ledger, proof verifier, game entitlement system or Pantheon settlement program. The observed extensions provide metadata and pause-related state; they do not establish private transfers or verified eligibility.

What registering or publishing means

The Devnet mint is already publicly inspectable on its network. Publishing a website or metadata does not turn it into a mainnet asset, establish a market or create enforceable economic rights. A future mainnet mint would be a separate issuance. No automatic conversion of Devnet balances, migration entitlement or redemption promise is established by this draft.

The current Devnet IUNUA is a test asset with no represented monetary value, redemption right or verified exchange rate. This describes the Devnet asset, not every capability of the wallet or the intended scope of IUNUA. Solana Devnet is distinct from mainnet and its state can be reset. The successful read-only checks used here did not issue, transfer, freeze or authorize any tokens.

Creating a mainnet token would establish a production-network mint, but it would not automatically enable this wallet to sign mainnet transactions or complete the trust-currency system. The configured production mint and RPC endpoint are currently unset and production signing is disabled. Custody and recovery, network configuration, reviewed signing and confirmation, security validation and explicit

activation are still required for a production wallet flow. Private accounting, eligibility verification and governed settlement have additional implementation requirements.

Sources: IUNUA launch preparation: read-only chain evidence; IUNUA mint on Solana Explorer (Devnet); Game treasury token account on Solana Explorer (Devnet); Solana documentation: create a token mint; Solana documentation: clusters and public RPC endpoints; Solana documentation: token extensions

10 / IUNUA SYSTEM & LAUNCH GUIDE

The prepared mainnet specification

A concrete proposal is ready for review. The decisions below are deliberately not marked approved.

Proposed pilot profile	
Setting	**Draft and remaining decision**
Name / symbol / network	IUNUA / IUNUA / Solana mainnet-beta. The mint address is not allocated. A ticker is not a unique identifier.
Program and precision	Token-2022, 9 decimals. Decide before mint creation; changing program or decimals later normally requires a new mint and migration.
Initial issuance	42,000,000 IUNUA as an unapproved planning assumption. 42000000000000000 base units. Allocation, vesting and future issuance remain undecided.
Supply policy	Retain mint authority under reviewed custody for the pilot; no enforced lifetime cap. Alternatively, issue the approved fixed supply and later revoke mint authority after review. Revocation cannot be undone.
Extensions and authorities	Metadata pointer, token metadata and pause; freeze authority enabled. No transfer fee, permanent delegate, confidential transfer or transfer hook. This selection needs wallet/venue acceptance.
Custody proposal	2-of-3 independently protected signers controlling a reviewed multisig vault. Public member addresses, vault, treasury and fee payer are not yet chosen. Do not reuse Devnet identities.
Metadata hosting	Prepare public HTTPS metadata and image with IUNUA name/symbol. Production URLs remain unset. The owner-only website cannot serve as public wallet metadata without a separate public asset host.
Fees and funding	Network fees in SOL. No IUNUA transfer tax in this profile. Operational SOL reserve and any liquidity must be separately funded.
Distribution	No sale, airdrop, vesting or Devnet conversion is authorized by this draft. Approve recipient policy, limits and contractual statements before distribution.

Retaining emergency controls is a tradeoff: it preserves some repair options but gives administrators meaningful power. Document who can mint, freeze, pause and edit metadata. A multisig is a custody control; it does not implement community voting. Squads or another reviewed vault must be rehearsed with these exact Token-2022 instructions, including metadata updates and authority rotation; built-in UI support is not assumed.

If broad wallet or DEX compatibility fails for the pause-enabled mint, compare a simpler Token-2022 profile or legacy SPL Token before issuance. Do not assume every extension can be added or removed later. Supply, mint address, program, decimals and extension choices are recorded in shared/mainnet/iunua-launch.json.

For an EU public offer or admission to trading, obtain a MiCA and other applicable-law assessment based on the actual rights, distribution and jurisdictions. No exemption, authorization or filing compliance is assumed here. This technical paper is not a substitute for a required regulatory whitepaper.

Sources: Solana documentation: token extensions; Solana set authority; Squads published pricing

Sources: ESMA MiCA rulebook

11 / IUNUA SYSTEM & LAUNCH GUIDE

Launch costs: chain fees versus building the product

The mint is inexpensive. Engineering, security, operations and any liquidity are separate budgets.

Budget boundary, 10 September 2026: the dated token-issuance quotes below remain estimates for the Solana token scope. They are not a complete quote for the newly specified World-authenticated XMR treasury, protected signer, payment verifier and public receipt service. Those components require a separate engineering, operations and review estimate after custody and receipt design decisions.

All amounts are planning estimates dated 9 September 2026, before tax. SOL is shown without an asserted spot price. EUR labor ranges are our estimates from explicit hours and rates, not vendor quotations or commitments. USD subscription prices below were checked on the providers' own pages.

Read-only mainnet cost example

Item	Assumption / amount
Mint rent reserve	1,024-byte planning budget: 0.007295616 SOL from mainnet getMinimumBalanceForRentExemption. Final account size must be calculated from serialized metadata and selected extensions.
Three initial token accounts	174-byte assumption each: 0.001912566 SOL per account, 0.005737698 SOL total. Confirm actual ATA extension size in rehearsal.

Read-only mainnet cost example	
Item	**Assumption / amount**
Base transaction fees	20 signature checks at the documented 5,000 lamports: 0.000100000 SOL. This is a count assumption, not a quote for constructed launch transactions.
Priority-fee allowance	0.001 SOL planning buffer. Re-estimate from actual transactions and congestion.
Subtotal before multisig	0.014133314 SOL, including account rent deposits and the fee allowances above. Rent is locked capital, not simply a consumed service fee.
Optional Squads service	Published creation fee: 0.1 SOL. Combined example: 0.114133314 SOL before extra vault/proposal accounts and transactions. Basic: USD 0/month; Pro token-management UI: USD 49/month.
Practical SOL reserve	Plan 0.25-1.0 SOL for initial operations and extra transactions, not a guarantee of final spend. At hypothetical EUR 50 / 100 / 200 per SOL, that reserve is EUR 12.50-50 / 25-100 / 50-200.
Growing holder count	If the project pays for 1,000 new 174-byte token accounts, rent alone would be 1.912566 SOL at this snapshot, plus transfers and fees. Do not double count accounts that already exist.

The quote was obtained at 2026-09-09T09:28:09.116Z. It verifies mainnet identity, then asks for rent at specified sizes. It does not sign, simulate or price a completed transaction. The actual launch must use final serialized transactions and fresh getFeeForMessage / simulation results. Rent, congestion, provider terms and SOL exchange rates can change.

Labor and review scenarios - alternative scopes, not additive totals		
Scope	**Estimated work and external review**	**Planning total**
A. Token issuance only	8-24 engineering hours at EUR 80-150/h = EUR 640-3,600; focused independent review EUR 1,000-4,000.	EUR 1,640-7,600 plus SOL, hardware, hosting and applicable legal work. This does not deliver our mainnet wallet.
B. Token + own wallet pilot	80-240 engineering/QA hours at EUR 80-150/h = EUR 6,400-36,000; independent review EUR 5,000-20,000.	EUR 11,400-56,000 plus SOL, hardware, hosting and legal work. Recommended planning scope; includes token preparation.
C. Complete trust service	400-1,200 hours at EUR 80-150/h = EUR 32,000-180,000; security/privacy/cryptographic review EUR 15,000-60,000.	EUR 47,000-240,000+ plus operating costs and legal work. High uncertainty; design changes can exceed this range.

For a working wallet-pilot budget, add a provisional EUR 2,000-10,000 legal/distribution assessment and EUR 300-900 for independent signing devices. Scenario B then totals EUR 13,700-66,900 before SOL, hosting, tax and contingency. A 25% contingency gives roughly EUR 17,125-83,625. If you build yourself, engineering hours remain real work but are not necessarily a cash invoice. Obtain scoped quotes before committing funds.

Recurring and optional costs	
Item	**Estimate or verified published price**
RPC	Helius Developer: USD 49/month, 10M credits and 50 requests/second. Business: USD 499/month. A public shared RPC is useful for checks; production reliability and quotas need assessment.
Application operations	Planning allowance EUR 25-150/month for metadata hosting, monitoring and backups at a small pilot; not a provider quote. Private ledger/game services add load and support work.
Maintenance	8-24 hours/month at EUR 80-150/h = EUR 640-3,600/month for patching, release support and incident readiness.
Liquidity / exchange access	No minimum or guaranteed listing is estimated. For example, a voluntarily selected EUR 5,000 quote-asset pool deposit is capital at risk, not a Solana launch fee; token inventory and pool costs are additional. No pool is prepared or approved.
Self-hosted node	Not required to issue or use a token. Dedicated infrastructure is a separate operational decision; the estimate uses hosted RPC.

Sources: IUNUA preparation preflight and cost assumptions; Solana transaction fees; Solana rent-exemption RPC; Helius published RPC prices; Squads published pricing

12 / IUNUA SYSTEM & LAUNCH GUIDE

What can change after launch?

Some controls are adjustable. Finalized transactions and foundational mint choices do not have an undo button.

Repair and evolution map		
Change or mistake	**Possible response**	**Limit**
Website, app UI or instructions	Ship a corrected release or website update.	Existing installed apps may stay old; changes do not rewrite on-chain state.
Wrong name, symbol or image URI	Update metadata while the relevant authority remains; correct hosted JSON/assets and notify indexers.	Caches may lag. Locked metadata cannot be edited. Mint address remains the same.
Wrong recipient or amount	Ask the recipient for a voluntary return; use a separately approved compensating transfer where justified.	Issuer has no general reversal or seizure power in this draft. Freezing or pausing does not return funds.

Repair and evolution map		
Change or mistake	**Possible response**	**Limit**
Unexpected transfer incident	Pause the whole mint or freeze affected accounts if the relevant controls are enabled and custody remains usable.	This blocks activity, including innocent users. It cannot undo a finalized transfer or recover a lost signing key.
Lost or compromised authority member	Use the remaining valid multisig quorum to rotate members/authorities and investigate.	If quorum is lost or attacker controls it, recovery may be impossible. A fee payer cannot substitute for an authority.
Need additional supply	Mint only while retained authority exists and approved policy allows it.	Dilutes holders; no immutable cap can be claimed. Revoked mint authority cannot be restored.
Want a fixed remaining maximum	Review final issuance, then revoke mint authority.	Irreversible; burning later reduces supply but does not permit reissuance. No lifetime mint counter is supplied by the basic mint.
Wrong decimals, program or missing mint extension	Create a replacement mint and an explicit, audited migration plan if the field cannot be changed.	New token identity, holder coordination, fraud risk and liquidity migration. Most mint extensions cannot simply be attached after initialization.
New rewards, game features or governance	Ship separately reviewed services/programs and explicit user permissions.	Their development can continue after token launch. Existing tokens do not retroactively grant rights or bind users to hidden terms.
Custom-program bug in a future service	Upgrade only if that program retains a valid upgrade mechanism and authority; otherwise migrate.	IUNUA controls its future programs, not the standard Solana Token-2022 program. Token authorities are not program-upgrade keys.

A staged approach preserves options

Start with a small, approved pilot, public authority disclosures and measured distribution. Rehearse pause/resume and rotation before distribution; retain tested quorum and an offline incident runbook. Do not revoke authorities just to display a reassuring badge before the policy is settled. Later reductions of authority should be separately approved and publicly documented.

If a replacement mint is necessary: stop new issuance where possible, publish both mint addresses and a snapshot policy, validate an opt-in migration or compensating distribution, prevent double claims, account for old/new supply, review pool migration and retain an immutable incident record. A migration does not erase the old token or chain history, and holders may decline to participate.

Sources: Solana set authority; Solana token metadata and pointer; Solana pausable mint; Solana documentation: token extensions

13 / IUNUA SYSTEM & LAUNCH GUIDE

Launch preparation and acceptance checklist

The preparation kit identifies missing decisions and produces read-only evidence. It has no transaction broadcast mode.

Ordered launch work	
Stage	**Required output before moving on**
1. Approve the specification	Choose initial supply, allocation/vesting, authority retention, extension profile, utility terms, launch countries and whether any sale is proposed. Record decisions against a specification hash.
2. Establish production custody	Create fresh independently held production signers outside this website. Rehearse 2-of-3 access, backup recovery and member rotation. Record only public addresses. Separate authority vault, treasury and low-balance fee payer.
3. Prepare public metadata	Publish IUNUA JSON and image on an appropriate public host; verify anonymous HTTPS delivery, hash, MIME type, name and symbol. Never expose private website data or keys.
4. Rehearse exact mint behavior	Use disposable Devnet identities with the proposed extensions. Verify mint/transfer/burn, freeze/thaw, pause/resume, metadata update and every authority rotation through the chosen multisig. Confirm actual wallet and desired venue acceptance.
5. Prepare the mainnet wallet	Implement the separate mainnet environment, account recovery/custody, allowlisted mint/program/decimals checks, fee/rent display, recipient review, confirmation/retry safety and packaged UI. Do not simply flip the existing lock flag.
6. Independent review and incident drill	Resolve critical findings. Test RPC outages, wrong cluster/mint, duplicate submissions, expired blockhash, rejected signatures, insufficient SOL, paused/frozen accounts, lost device and compromised member.
7. Review exact transactions	Generate an auditable unsigned instruction/transaction set using the approved identities and final metadata. Verify genesis, account ownership, rent, destination and supply; simulate where possible. Review all authority assignments before any signature. This final set cannot be produced yet because identities and policy are undecided.
8. Bounded issuance and acceptance	After explicit launch authorization, execute the reviewed creation and issuance flow with receipts, inspect finalized state and reconcile supply/treasury. Run small verified mainnet transfers with the exact released wallet before wider distribution.

Ordered launch work	
Stage	**Required output before moving on**
9. Expand deliberately	Increase limits only after monitoring and operational acceptance. Treat liquidity, sales, game rewards and the private trust service as separate reviewed releases.

An ambiguous submission must be checked by intended mint address and transaction status before retrying. Do not rerun a create/mint sequence blindly. Every irreversible authority revocation requires its own review; the preparation kit neither revokes powers nor transfers assets.

Prepared files

shared/mainnet/iunua-launch.json is the draft specification. scripts/prepare-iunua-mainnet.mjs validates identity, exact supply arithmetic, proposed powers and missing decisions; --quote performs only allowlisted read-only RPC calls. --check exits nonzero while decisions or public addresses are missing. docs/mainnet/preflight.json contains the current blocked status and live rent assumptions. docs/mainnet/README.md contains the operator runbook and metadata template instructions.

The tool never reads private keys, generates signers, requests airdrops, signs or submits transactions. Even a configuration marked complete is not a launch certificate. Network failures remain unknown. The wallet's production signing lock remains unchanged. The downloadable preparation kit contains public specifications and evidence only.

Sources: IUNUA mainnet preparation kit; IUNUA preparation preflight and cost assumptions; IUNUA token and wallet capability status

14 / IUNUA SYSTEM & LAUNCH GUIDE

Mainnet registration: the step-by-step guide

A mint is the token’s public identity. Follow creation, shared approval, initial issuance and verification as separate steps. This reference still needs an end-to-end Devnet rehearsal before production use.

This is complete project-specific reference source for creating the proposed IUNUA Token-2022 mint and issuing its initial supply through an existing autonomous Squads V4 2-of-3 vault. It uses the official token and multisig programs; there is no separate IUNUA Rust contract to deploy for this profile. It is not the full wallet, private ledger, eligibility verifier, exchange, payment service or governance application.

Validation scope: SDK instruction encoding, exact integer supply, deterministic addresses, atomic instruction grouping, proposal size, account-message round-trip, configuration/hash checks and mocked failure cases were tested offline. No Devnet/Mainnet transaction or Ledger interaction was performed for

this kit. An end-to-end Devnet rehearsal and independent code/custody review remain required before using it with real SOL. The existing wallet's production lock is unchanged.

Known dependency blockers recorded on 9 September 2026

The pinned dependency audit reports 11 affected dependency nodes (4 high, 7 moderate), originating from three advisory families: bigint-buffer native buffer overflow (GHSA-3gc7-fjrx-p6mg), stream-json denial of service (GHSA-528h-pc64-c93x), and uuid buffer bounds (GHSA-w5hq-g745-h8pq). These are not 11 independent vulnerabilities. Installing with --ignore-scripts currently selects the bigint pure-JavaScript fallback, but that is not a complete dependency remediation or production sign-off.

The shipped security-review.json records unresolved status, and operator.mjs refuses mainnet submission while it is unresolved. A developer must resolve or independently assess the affected paths, record the reviewed remediation/evidence and approval-document hash, and rerun offline plus end-to-end Devnet tests before producing a revised launch package. Do not just edit the status to bypass the block. npm audit fix --force suggests incompatible major-version changes; it was not used because it could remove required Token-2022/Squads functionality. The numbered mainnet commands below describe the future reviewed run, not an immediately launchable release.

Contents and commands

• registration.mjs: read-only address derivation, state checks, metadata checks, unsigned message preparation and initial-issuance verification.

• operator.mjs: explicitly invokes the official Squads CLI to propose, approve or execute. It can spend SOL and issue tokens when an operator supplies real configuration, a Ledger signer, a reviewed plan hash and the mainnet acknowledgement.

• config.example.json: public configuration with missing decisions/addresses. Copy to config.json; do not put keys, recovery phrases or private RPC credentials in it.

• metadata.template.json: public metadata template. Publish final JSON/image to an anonymous HTTPS host; the private IUNUA website is not suitable for token metadata.

• registration.test.mjs: offline tests with synthetic fixtures, not production identities.

• security-review.json: recorded dependency blockers; mainnet actions are disabled until remediation and review.

• package.json and package-lock.json: pinned JavaScript dependency versions and integrity records. The dependencies remain subject to a separate vulnerability/supply-chain review.

Step 1 - Approve the actual token policy

Confirm IUNUA name/symbol, 9 decimals, exact initial supply and allocation, and retained mint, freeze, pause, metadata-update and pointer authorities. 42,000,000 is a draft quantity, not an approved mainnet cap. This profile excludes transfer fees, permanent delegates, transfer hooks, confidential transfer and mint closing. Write a decision document that identifies the configuration, signers, release scope, risks and rollout limits. Record its real SHA256 in approvedLaunchDocumentSha256 after approval; a hash is an integrity reference, not proof of approval.

Step 2 - Install and review the tools

Install Node.js 22+, Rust/Cargo, the reviewed Solana tool suite and the official Squads CLI. This reference inspected Solana CLI 4.2.2, SPL Token CLI 5.6.1, Squads CLI source 0.1.7 and the pinned JS packages. Do not treat a version match as a security audit.

```
npm ci --ignore-scripts
npm test
cargo install squads-multisig-cli --version 0.1.7 --locked
solana --version
spl-token --version
squads-multisig-cli --version
```

If a tool's help/options differ, stop and reconcile versions before sending. The operator requires Squads CLI 0.1.7. Its documented Ledger interface uses a quoted usb://ledger URL; verify derivation paths and public keys on each device. Never enter a hardware-wallet recovery phrase into these scripts.

Step 3 - Rehearse on Devnet with separate identities

Copy the example to config.json, use cluster: "devnet", disposable signers, a new Devnet multisig and a seed such as iunua-rehearsal-1. Follow all remaining steps on Devnet first. Record create, proposal, approve, execute and verification results. Exercise transfer, burn, freeze/thaw, pause/resume and recovery separately through the vault. Check that a paused token rejects transfers, issuance and burns. Repeat with the actual intended wallet and any intended venue; SDK success is not wallet compatibility. Do not reuse those test identities on mainnet.

Step 4 - Establish mainnet custody

Use three independently protected production signers and verify backups and quorum recovery. Create an autonomous 2-of-3 Squads vault using the reviewed Squads interface or this official CLI command. Supply the three real public keys; each permission value 7 means initiate, vote and execute. Omit --config-authority so no single key can unilaterally rewrite membership. Creation spends real SOL.

```
# Run only after reviewing the custody decision and displayed transaction.
# Replace MEMBER_1, MEMBER_2 and MEMBER_3 with verified public addresses.
squads-multisig-cli multisig-create \
  --rpc-url https://api.mainnet-beta.solana.com \
  --program-id SQDS4ep65T869zMMBKyuUq6aD6EgTu8psMjkvj52pCf \
  --keypair 'usb://ledger' \
  --members 'MEMBER_1,7' 'MEMBER_2,7' 'MEMBER_3,7' \
  --threshold 2
```

Record the returned multisig address in config.json. The multisig configuration account and vault address are different. The builder checks program ownership, 2-of-3 membership, permissions, time lock and the absence of unilateral configuration authority. It does not enumerate spending limits or outstanding configuration proposals: inspect those separately and remove unintended authority paths. A live PDA-derived mint is new public state, not a renamed or migrated Devnet mint.

Step 5 - Publish metadata and complete public configuration

Publish a final logo and JSON metadata anonymously over HTTPS. Set name/symbol to IUNUA. Use the final JSON URL (at most 160 characters for this reference) in metadataUri, and SHA256 hashes of the exact hosted JSON and image bytes in the corresponding fields. Do not use an owner-only URL, tracking credential, private RPC URL or expiring signed link. The builder verifies HTTP responses, MIME types, hashes, name and symbol before preparation and before operator actions.

Copy config.example.json to config.json and complete the multisig, its exact three members, metadata hashes, reviewed supply, mint seed and approval-document hash. The zero-second time lock is an explicit default, not a recommendation; choose the reviewed delay and match the live vault. A nonzero delay is enforced by Squads during execution.

```
cp config.example.json config.json
# Edit config.json with approved PUBLIC values, then:
node registration.mjs derive config.json
```

Record the derived vault, mint and treasury addresses. The mint uses createAccountWithSeed with the vault as base; no separate mint private key exists. The vault must sign the atomic creation through Squads. Freeze, mint, pause, metadata update and metadata pointer powers are assigned to that vault from initialization. Never change the seed after announcing the mint address.

Step 6 - Fund the correct accounts and prepare creation

Fund the vault address with enough real SOL for token-account rent, and the participating member fee-payer addresses for proposals, votes and execution. Do not send vault funds to the multisig configuration address. Use current quotes and a small reserve; the earlier cost chapter is a dated estimate. Liquidity and token-market listing are separate expenses.

```
mkdir -p out
node registration.mjs build config.json create out/create.json
```

This command performs reads, verifies metadata/custody, computes exact mint rent and writes an unsigned Squads message and decoded instruction list. It does not sign, propose, transfer or mint. It deliberately refuses to overwrite an existing plan or use an occupied mint address. Independently inspect every program, authority, mint, account size, rent amount and instruction. Keep the printed plan SHA256 in the review record; do not automatically trust a hash supplied alongside an unreviewed file.

Step 7 - Propose mint creation

After the mainnet review gates are complete, select the first Ledger signer and explicitly acknowledge the network. A custom RPC may be supplied with IUNUA_RPC_URL; the underlying third-party CLI can display that endpoint, so avoid placing credential-bearing URLs in shared logs.

```
export IUNUA_SIGNER='usb://ledger'
export IUNUA_ALLOW_MAINNET='IUNUA MAINNET'
node operator.mjs propose config.json out/create.json REVIEWED_CREATE_SHA256
```

Replace the hash placeholder with the independently approved 64-character hash. Review the CLI/device prompt and confirm only the intended transaction. Proposing spends fees/rent and creates an on-chain proposal; it does not execute the mint instructions. Record the actual proposal index and

signature from the result. Wait for finalization. Indices are not assumed to be 1 or consecutive because other vault activity may occur.

Step 8 - Independently review, approve and execute creation

Each signer checks the stored on-chain proposal against the same reviewed plan. Use separate devices/operators for the two approvals, with the correct Ledger URL/derivation for each. Do not give both production keys to one machine just to finish the example.

```
node operator.mjs review config.json out/create.json REVIEWED_CREATE_SHA256 CREATE_INDEX
# First authorized member/device:
node operator.mjs approve config.json out/create.json REVIEWED_CREATE_SHA256 CREATE_INDEX
# Second authorized member/device, with its own IUNUA_SIGNER:
node operator.mjs approve config.json out/create.json REVIEWED_CREATE_SHA256 CREATE_INDEX
# After threshold and any configured time lock:
node operator.mjs execute config.json out/create.json REVIEWED_CREATE_SHA256 CREATE_INDEX
```

The operator verifies the proposal bytes, selected member, network, vault policy and fresh state before calling the CLI. The CLI supplies the current outer transaction blockhash and signing flow. Squads and Token-2022 enforce authorization on-chain. Creation, extension initialization, mint initialization and metadata initialization occur atomically. The mint then exists publicly with zero supply. If the result is unclear, inspect the signature, proposal and mint address before taking another action; never blindly repeat a creation or issuance command.

Step 9 - Prepare and execute initial issuance

After mint creation is finalized and checked, build the separate issuance plan. The builder confirms the exact mint profile, retained authorities, zero supply and absence of the treasury ATA.

```
node registration.mjs build config.json issue out/issue.json
node operator.mjs propose config.json out/issue.json REVIEWED_ISSUE_SHA256
node operator.mjs review config.json out/issue.json REVIEWED_ISSUE_SHA256 ISSUE_INDEX
node operator.mjs approve config.json out/issue.json REVIEWED_ISSUE_SHA256 ISSUE_INDEX
# Second authorized member/device:
node operator.mjs approve config.json out/issue.json REVIEWED_ISSUE_SHA256 ISSUE_INDEX
node operator.mjs execute config.json out/issue.json REVIEWED_ISSUE_SHA256 ISSUE_INDEX
```

Use the newly reviewed issuance hash and actual issuance proposal index. This proposal creates the vault-owned treasury token account and mints the exact initial amount into it in one transaction. The non-idempotent account creation makes repeated initial-issuance proposals fail atomically once that treasury exists. Do not pre-create the treasury ATA. Retained mint authority can still issue more tokens through a different authorized transaction, so this replay guard is not a lifetime supply cap.

Step 10 - Verify and preserve the launch record

```
node registration.mjs verify config.json > out/initial-issuance-receipt.json
unset IUNUA_ALLOW_MAINNET
```

This initial-issuance check requires exact supply and treasury balance, the expected metadata and extensions, vault authorities, and an unfrozen undelegated treasury. Run it before any distribution; after transfers or burns, an exact initial treasury match is no longer expected and reconciliation must follow the actual transaction history. Record finalized signatures/slots, configuration and source hashes, proposal

indices, authority policy and the mint address. Inspect the exact address in Solana Explorer on mainnet-beta, not a same-named search result.

Step 11 - Recognition and wallet rollout

A mainnet mint already exists on-chain after creation; explorer search inclusion, third-party verification, wallet token lists, market liquidity and exchange listings are separate processes. Follow current Solana Explorer/Jupiter verification guidance if desired; verification is not a promise of investment quality or IUNUA utility. Release a separately reviewed IUNUA mainnet wallet configuration and transaction flow, then perform approved low-value canary transfers and reconciliation before wider distribution. The installed 0.8.4 wallets do not gain production support because this kit exists.

Step 12 - Corrections after launch

Use new reviewed vault proposals for retained mint authority, pause/resume, freeze/thaw, metadata/pointer updates and authority rotation. These are deliberately not automatic incident-response commands in this creation kit. Do not revoke any authority simply to clear a checklist. Metadata edits can correct a name/image/description if the update authority is retained; finalized transfers cannot be undone. Decimals, token-program choice and incompatible extension changes may require a replacement mint plus an explicit migration plan. A wallet/website update cannot rewrite chain history. See the guide's changeability chapter for the full decision map.

Production acceptance still needed

• Run the complete reference with disposable identities on Devnet and record actual finalized results, including bad-signature, lost-device, pause/freeze and retry cases.

• Verify the pinned CLI parses repeated transaction-message arguments correctly in the operator environment; official 0.1.7 source uses Clap Vec<u8> with repeated long options. Validate Ledger support and exact device display, not just source code.

• Independently review application and dependency code. Offline unit tests do not test network execution, hardware compatibility, compute limits, live program changes or custody procedures.

• Check CPI/Token-2022 behavior and transaction simulation through the chosen vault, current rent and fees, metadata hosting continuity, outstanding vault proposals/spending limits and intended wallet/venue support.

• Complete policy, public-distribution/legal, security, monitoring and wallet-release work from the mainnet prerequisites before expanding the relevant launch scope.

The full handwritten creation/verification/operator source is printed in the practical guide. The dependency lockfile, tests, metadata template and this runbook are included in the downloadable archive. Nothing in this archive claims that a mainnet mint was created or a production launch approved.

Sources: IUNUA registration reference source v1.0; Solana: metadata pointer and token metadata; Solana: pausable mint; Solana: token verification and Explorer visibility; Squads V4 CLI: source and hardware-wallet interface; Squads: creating vault transactions

15 / IUNUA SYSTEM & LAUNCH GUIDE

Mainnet code: full reference source and file tree

Read the complete project-specific source below, then use the matching downloadable files for review and rehearsal. The operator entry point can submit real transactions when explicitly configured.

The following is the complete handwritten registration implementation: dependency declaration, public templates, builder and operator. It creates the proposed basic token through an existing reviewed vault. The full dependency lockfile, test source and runbook are in the downloadable reference kit; third-party program/SDK source is linked, not reproduced as IUNUA code.

The complete file tree

```
mainnet-registration/
|-- README.md                 # the step-by-step procedure
|-- package.json              # pinned direct dependencies
|-- package-lock.json         # exact dependency integrity
|-- config.example.json       # public launch settings
|-- metadata.template.json    # public token metadata
|-- registration.mjs          # read-only builder + verification
|-- operator.mjs              # explicit propose / approve / execute
|-- security-review.json      # unresolved audit; mainnet blocked
|-- registration.test.mjs     # offline tests, synthetic identities
`-- .gitignore                # excludes local config and outputs

Created locally by the operator (not packaged):
|-- config.json               # approved public values, no secrets
`-- out/
    |-- create.json           # reviewed mint-creation proposal
    |-- issue.json            # reviewed initial-issuance proposal
    `-- initial-issuance-receipt.json

Keys stay on the selected hardware devices, outside this tree.
Runtime: Node + official Squads CLI -> Squads vault -> Token-2022.
No new IUNUA on-chain Rust program is deployed by this profile.
```

Code that exists versus the rest of the IUNUA system

Component	Included here	Still separate
Token identity and extensions	Deterministic seeded mint, metadata/pause initialization, vault authorities.	Actual mainnet identities and approved policy.
Initial issuance	Treasury creation and exact checked mint amount in one proposal.	Allocation/distribution rules, future issuance and any market.

Code that exists versus the rest of the IUNUA system		
Component	**Included here**	**Still separate**
Shared authorization	Exact-message review plus official CLI propose, vote and execute with Ledger.	Custody ceremony, live rehearsal, independent audit and operational acceptance.
Verification	Genesis, mint profile, authority, initial treasury and supply checks.	Production monitoring, accounting and incident-response implementation.
Wallet and trust service	No wallet signing or private-trust functionality is added.	Mainnet wallet release, ledger, proofs, eligibility, payments and settlement.

package.json

```
{
  "name": "iunua-mainnet-registration-reference",
  "version": "1.0.0",
  "private": true,
  "type": "module",
  "engines": {"node": ">=22"},
  "scripts": {"test": "node --test registration.test.mjs"},
  "dependencies": {
    "@solana/web3.js": "1.99.0",
    "@solana/spl-token": "0.4.15",
    "@solana/spl-token-metadata": "0.1.6",
    "@sqds/multisig": "2.1.4"
  }
}
```

config.example.json

```
{
  "cluster": "mainnet-beta",
  "name": "IUNUA",
  "symbol": "IUNUA",
  "decimals": 9,
  "initialSupply": "42000000",
  "mintSeed": "iunua-v1",
  "multisig": null,
  "vaultIndex": 0,
  "members": [null, null, null],
  "timeLockSeconds": 0,
  "metadataUri": null,
  "metadataSha256": null,
  "imageSha256": null,
  "approvedLaunchDocumentSha256": null
}
```

metadata.template.json

```
{
  "name": "IUNUA",
  "symbol": "IUNUA",
  "description": "IUNUA token. See published project terms for rights and usage.",
  "image": "https://REPLACE-WITH-PUBLIC-HOST/iunua.png"
}
```

registration.mjs

```
// Read-only builder. All public token actions execute through the Squads vault.
import assert from 'node:assert/strict';
import { createHash } from 'node:crypto';
import { readFile, writeFile } from 'node:fs/promises';
import { pathToFileURL } from 'node:url';
import { Connection, PublicKey, SystemProgram,
  TransactionMessage } from '@solana/web3.js';
import * as token from '@solana/spl-token';
import { pack, createInitializeInstruction } from '@solana/spl-token-metadata';
import * as squads from '@sqds/multisig';

export const PROGRAM = token.TOKEN_2022_PROGRAM_ID;
export const GENESIS = {
  'mainnet-beta': '5eykt4UsFv8P8NJdTREpY1vzqKqZKvdpKuc147dw2N9d',
  devnet: 'EtWTRABZaYq6iMfeYKouRu166VU2xqa1'
};
export const digest = bytes => createHash('sha256').update(bytes).digest('hex');
const key = s => { assert.equal(typeof s, 'string'); return new PublicKey(s); };
const isHash = s => typeof s === 'string' && /^[a-f0-9]{64}$/.test(s);
const TEST_KEYS = new Set([
  '9UePLTf7PRfbZUDzRq8f4sVo3CE7DsZvEtY7ypYZprrk',
  'EixVckoyE92vZTVvA2YYkANenrvJdYDW1rTN5ZP8U43W',
  '98ACca4GvCEyooGv8huVLQ3BRbfw8jr3CyDWm4gwz5JF',
  '8NcU2LK62iNV2ySjHZKgKf72r4yP7tP8dELREZvyr1nW'
]);
export function validate(c) {
  assert.ok(Object.hasOwn(GENESIS, c.cluster), 'Choose mainnet-beta or devnet.');
  assert.equal(c.name, 'IUNUA'); assert.equal(c.symbol, 'IUNUA');
  assert.equal(c.decimals, 9, 'Changing precision needs a new profile.');
  assert.match(c.initialSupply, /^[1-9][0-9]{0,10}$/);
  assert.ok(BigInt(c.initialSupply) * 10n ** 9n < 2n ** 64n);
  assert.match(c.mintSeed, /^[a-z0-9-]{1,32}$/);
  key(c.multisig); assert.equal(c.vaultIndex, 0);
  assert.equal(c.members.length, 3);
  c.members.forEach(key); assert.equal(new Set(c.members).size, 3);
  assert.ok(Number.isInteger(c.timeLockSeconds) && c.timeLockSeconds >= 0);
  if (c.cluster === 'mainnet-beta') {
    assert.ok(![c.multisig, ...c.members].some(k => TEST_KEYS.has(k)),
      'Known test identities must not control production.');
  }
  const u = new URL(c.metadataUri);
  assert.equal(u.protocol, 'https:'); assert.ok(!u.username && !u.password);
  assert.ok(c.metadataUri.length <= 160, 'Use a short public metadata URL.');
  assert.ok(isHash(c.metadataSha256) && isHash(c.imageSha256));
  return c;
}
export async function identities(c) {
  validate(c);
  const multisig = key(c.multisig);
  const [vault] = squads.getVaultPda({ multisigPda: multisig, index: 0 });
  // This is a seeded account, not a new keypair or a migrated Devnet mint.
  const mint = await PublicKey.createWithSeed(vault, c.mintSeed, PROGRAM);
  const treasury = token.getAssociatedTokenAddressSync(mint, vault, true, PROGRAM);
  return { multisig, vault, mint, treasury };
}
export function connect(c) {
  const endpoint = process.env.IUNUA_RPC_URL ||
    (c.cluster === 'devnet' ? 'https://api.devnet.solana.com' :
      'https://api.mainnet-beta.solana.com');
  return new Connection(endpoint, 'finalized');
}
export async function context(c, connection = connect(c)) {
  const ids = await identities(c);
  assert.equal(await connection.getGenesisHash(), GENESIS[c.cluster]);
  const raw = await connection.getAccountInfo(ids.multisig, 'finalized');
```

```
  assert.ok(raw && raw.owner.equals(squads.PROGRAM_ID), 'Wrong multisig owner.');
  const [ms] = squads.accounts.Multisig.fromAccountInfo(raw);
  assert.equal(ms.threshold, 2, 'Expected 2-of-3.');
  assert.equal(ms.members.length, 3);
  assert.ok(ms.configAuthority.equals(PublicKey.default),
    'A unilateral configuration authority bypasses shared control.');
  assert.equal(ms.timeLock, c.timeLockSeconds);
  assert.deepEqual(ms.members.map(m => m.key.toBase58()).sort(), [...c.members].sort());
  assert.ok(ms.members.every(m => m.permissions.mask === 7));
  return { ...ids, connection, ms };
}
export async function verifyMetadata(c) {
  const response = await fetch(c.metadataUri, { signal: AbortSignal.timeout(20000) });
  assert.ok(response.ok, 'Metadata unavailable anonymously.');
  assert.ok(response.headers.get('content-type')?.includes('json'));
  const bytes = Buffer.from(await response.arrayBuffer());
  assert.equal(digest(bytes), c.metadataSha256, 'Metadata bytes changed.');
  const data = JSON.parse(bytes);
  assert.equal(data.name, c.name); assert.equal(data.symbol, c.symbol);
  const u = new URL(data.image); assert.equal(u.protocol, 'https:');
  assert.ok(!u.username && !u.password);
  const image = await fetch(u, { signal: AbortSignal.timeout(20000) });
  assert.ok(image.ok && image.headers.get('content-type')?.startsWith('image/'));
  assert.equal(digest(Buffer.from(await image.arrayBuffer())), c.imageSha256);
}
export function metadata(c, ids) {
  return { updateAuthority: ids.vault, mint: ids.mint, name: c.name,
    symbol: c.symbol, uri: c.metadataUri, additionalMetadata: [] };
}
export function mintSizes(c, ids) {
  const fixed = token.getMintLen([
    token.ExtensionType.MetadataPointer, token.ExtensionType.PausableConfig
  ]);
  return { fixed, full: fixed + token.TYPE_SIZE + token.LENGTH_SIZE +
    pack(metadata(c, ids)).length };
}
export function createInstructions(c, ids, rent) {
  assert.ok(Number.isSafeInteger(rent) && rent > 0);
  const { vault, mint } = ids;
  return [
    SystemProgram.createAccountWithSeed({ fromPubkey: vault, basePubkey: vault,
      newAccountPubkey: mint, seed: c.mintSeed, lamports: rent,
      space: mintSizes(c, ids).fixed, programId: PROGRAM }),
    token.createInitializeMetadataPointerInstruction(mint, vault, mint, PROGRAM),
    token.createInitializePausableConfigInstruction(mint, vault, PROGRAM),
    token.createInitializeMint2Instruction(mint, c.decimals, vault, vault, PROGRAM),
    createInitializeInstruction({ programId: PROGRAM, metadata: mint,
      updateAuthority: vault, mint, mintAuthority: vault,
      name: c.name, symbol: c.symbol, uri: c.metadataUri })
  ];
}
export function issueInstructions(c, ids) {
  const { vault, mint, treasury } = ids;
  return [
    // Deliberately non-idempotent: a second initial-issuance proposal fails
    // because this treasury ATA already exists. Both instructions are atomic.
    token.createAssociatedTokenAccountInstruction(vault, treasury, vault, mint, PROGRAM),
    token.createMintToCheckedInstruction(mint, treasury, vault,
      BigInt(c.initialSupply) * 10n ** 9n, c.decimals, [], PROGRAM)
  ];
}
export async function verifyMint(c, ctx) {
  const { connection, mint, vault } = ctx;
  const m = await token.getMint(connection, mint, 'finalized', PROGRAM);
  assert.ok(m.isInitialized); assert.equal(m.decimals, c.decimals);
  assert.ok(m.mintAuthority?.equals(vault) && m.freezeAuthority?.equals(vault));
  const pointer = token.getMetadataPointerState(m);
  const pause = token.getPausableConfig(m);
```

```
  assert.ok(pointer?.authority?.equals(vault) && pointer.metadataAddress?.equals(mint));
  assert.ok(pause?.authority.equals(vault)); assert.equal(pause.paused, false);
  const extensions = token.getExtensionTypes(m.tlvData).sort((a, b) => a - b);
  assert.deepEqual(extensions, [token.ExtensionType.MetadataPointer,
    token.ExtensionType.PausableConfig, token.ExtensionType.TokenMetadata].sort((a,b) => a-b));
  const md = await token.getTokenMetadata(connection, mint, 'finalized', PROGRAM);
  assert.deepEqual(md, metadata(c, ctx));
  return m;
}
export function messageBytes(ids, instructions) {
  // Squads stores its own message format; this is NOT a signed Solana transaction.
  const message = new TransactionMessage({ payerKey: ids.vault,
    recentBlockhash: PublicKey.default.toBase58(), instructions });
  return Buffer.from(squads.utils.transactionMessageToMultisigTransactionMessageBytes({
    message, vaultPda: ids.vault
  }));
}
export function instructionView(ix) {
  return { program: ix.programId.toBase58(), dataHex: ix.data.toString('hex'),
    accounts: ix.keys.map(k => ({ address: k.pubkey.toBase58(),
      signer: k.isSigner, writable: k.isWritable })) };
}
export async function build(c, stage, ctx) {
  ctx ??= await context(c);
  assert.ok(['create', 'issue'].includes(stage));
  await verifyMetadata(c);
  let instructions, rent;
  if (stage === 'create') {
    assert.equal(await ctx.connection.getAccountInfo(ctx.mint, 'finalized'), null,
      'Mint already exists. Verify it; never blindly repeat creation.');
    rent = await ctx.connection.getMinimumBalanceForRentExemption(mintSizes(c, ctx).full);
    instructions = createInstructions(c, ctx, rent);
  } else {
    assert.equal((await verifyMint(c, ctx)).supply, 0n, 'Initial issuance already occurred.');
    assert.equal(await ctx.connection.getAccountInfo(ctx.treasury, 'finalized'), null,
      'Treasury exists; investigate before initial issuance.');
    instructions = issueInstructions(c, ctx);
    rent = await ctx.connection.getMinimumBalanceForRentExemption(
      token.getAccountLen([token.ExtensionType.ImmutableOwner,
        token.ExtensionType.PausableAccount]));
  }
  assert.ok(await ctx.connection.getBalance(ctx.vault, 'finalized') >= rent,
    'Vault needs SOL for account rent; members separately pay transaction fees.');
  const bytes = messageBytes(ctx, instructions);
  return { schemaVersion: 1, stage, cluster: c.cluster,
    configSha256: digest(JSON.stringify(c)), createdAt: new Date().toISOString(),
    mint: ctx.mint.toBase58(), vault: ctx.vault.toBase58(),
    treasury: ctx.treasury.toBase58(), rentLamports: rent,
    messageSha256: digest(bytes), messageBase64: bytes.toString('base64'),
    instructions: instructions.map(instructionView) };
}
export async function checkPlan(c, plan, ctx) {
  assert.equal(plan.schemaVersion, 1); assert.equal(plan.cluster, c.cluster);
  assert.equal(plan.configSha256, digest(JSON.stringify(c)));
  assert.equal(plan.mint, ctx.mint.toBase58());
  assert.equal(plan.vault, ctx.vault.toBase58());
  assert.equal(plan.treasury, ctx.treasury.toBase58());
  assert.ok(['create', 'issue'].includes(plan.stage));
  const ixs = plan.stage === 'create' ? createInstructions(c, ctx, plan.rentLamports) :
    issueInstructions(c, ctx);
  const bytes = messageBytes(ctx, ixs);
  assert.equal(bytes.toString('base64'), plan.messageBase64);
  assert.equal(digest(bytes), plan.messageSha256);
  assert.deepEqual(ixs.map(instructionView), plan.instructions);
  return bytes;
}
export async function checkProposal(c, plan, index, ctx) {
  const bytes = await checkPlan(c, plan, ctx);
```

```
  const transactionIndex = BigInt(index);
  assert.ok(transactionIndex > 0n && transactionIndex < 2n ** 64n);
  const [address] = squads.getTransactionPda({ multisigPda: ctx.multisig, index: transactionIndex });
  const raw = await ctx.connection.getAccountInfo(address, 'finalized');
  assert.ok(raw && raw.owner.equals(squads.PROGRAM_ID));
  const [tx] = squads.accounts.VaultTransaction.fromAccountInfo(raw);
  assert.ok(tx.multisig.equals(ctx.multisig)); assert.equal(tx.vaultIndex, 0);
  assert.equal(tx.index.toString(), index); assert.equal(tx.ephemeralSignerBumps.length, 0);
  const [stored] = squads.types.transactionMessageBeet.serialize({
    ...tx.message, instructions: tx.message.instructions.map(ix => ({ ...ix,
      accountIndexes: Array.from(ix.accountIndexes), data: Array.from(ix.data) }))
  });
  assert.ok(Buffer.from(stored).equals(bytes), 'On-chain proposal differs from reviewed plan.');
  const [proposalAddress] = squads.getProposalPda({ multisigPda: ctx.multisig, transactionIndex });
  const proposalRaw = await ctx.connection.getAccountInfo(proposalAddress, 'finalized');
  assert.ok(proposalRaw && proposalRaw.owner.equals(squads.PROGRAM_ID));
  const [proposal] = squads.accounts.Proposal.fromAccountInfo(proposalRaw);
  assert.ok(proposal.multisig.equals(ctx.multisig));
  assert.equal(proposal.transactionIndex.toString(), index);
  return { transaction: address.toBase58(), status: proposal.status.__kind,
    approved: proposal.approved.map(k => k.toBase58()) };
}
async function main([action, configPath, argument, output]) {
  assert.ok(configPath, 'Usage: node registration.mjs derive|build|verify CONFIG [create|issue OUT]');
  const c = JSON.parse(await readFile(configPath, 'utf8'));
  if (action === 'derive') {
    const ids = await identities(c);
    console.log(JSON.stringify(Object.fromEntries(Object.entries(ids).map(([k,v]) =>
      [k, v.toBase58()])), null, 2)); return;
  }
  const ctx = await context(c);
  if (action === 'build') {
    assert.ok(output, 'Provide a new output file.');
    const plan = await build(c, argument, ctx);
    await writeFile(output, JSON.stringify(plan, null, 2) + '\n', { flag: 'wx' });
    console.log('Review plan SHA256:', digest(JSON.stringify(plan)));
  } else if (action === 'verify') {
    const m = await verifyMint(c, ctx);
    const treasury = await token.getAccount(ctx.connection, ctx.treasury, 'finalized', PROGRAM);
    assert.ok(treasury.owner.equals(ctx.vault) && treasury.mint.equals(ctx.mint));
    assert.ok(!treasury.isFrozen && treasury.delegate === null);
    const expected = BigInt(c.initialSupply) * 10n ** 9n;
    assert.equal(m.supply, expected); assert.equal(treasury.amount, expected);
    console.log(JSON.stringify({ verifiedAt: new Date().toISOString(),
      mint: ctx.mint.toBase58(), supplyBaseUnits: m.supply.toString(),
      treasuryBaseUnits: treasury.amount.toString(), initialIssuanceVerified: true }, null, 2));
  } else throw new Error('Unknown read-only action.');
}
if (process.argv[1] && import.meta.url === pathToFileURL(process.argv[1]).href)
  main(process.argv.slice(2)).catch(e => { console.error(e.message); process.exitCode = 1; });
```

operator.mjs

```
// Explicit operator entry point. This file CAN submit transactions via the CLI.
// The read-only registration.mjs never calls it. No auto-approval or auto-retry.
import assert from 'node:assert/strict';
import { readFile } from 'node:fs/promises';
import { spawnSync } from 'node:child_process';
import { pathToFileURL } from 'node:url';
import { PublicKey, TransactionMessage, VersionedTransaction,
  ComputeBudgetProgram } from '@solana/web3.js';
import * as squads from '@sqds/multisig';
import { context, build, digest, checkPlan, checkProposal } from './registration.mjs';

const security = JSON.parse(await readFile(new URL('./security-review.json', import.meta.url)));
export function gates(action, c, plan, expectedHash, env, review = security) {
  assert.ok(['review', 'propose', 'approve', 'execute'].includes(action));
  assert.equal(digest(JSON.stringify(plan)), expectedHash,
    'Plan differs from the independently reviewed SHA256.');
  if (action === 'review') return;
  assert.match(c.approvedLaunchDocumentSha256 ?? '', /^[a-f0-9]{64}$/,
    'Record the actual signed-off launch decision document, not a dummy hash.');
  if (c.cluster === 'mainnet-beta') {
    assert.equal(review.status, 'resolved-and-reviewed',
      'Mainnet blocked: resolve recorded dependency advisories and complete review.');
    assert.match(review.approvalDocumentSha256 ?? '', /^[a-f0-9]{64}$/);
    assert.equal(env.IUNUA_ALLOW_MAINNET, 'IUNUA MAINNET');
    assert.ok(env.IUNUA_SIGNER?.startsWith('usb://ledger'),
      'This mainnet operator flow requires a Ledger signer.');
  } else assert.ok(env.IUNUA_SIGNER, 'Select a disposable Devnet signer.');
}
export function proposalSize(ids, member, index, ixs) {
  const inner = new TransactionMessage({ payerKey: ids.vault,
    recentBlockhash: PublicKey.default.toBase58(), instructions: ixs });
  const instructions = [ComputeBudgetProgram.setComputeUnitPrice({ microLamports: 5000 }),
    squads.instructions.vaultTransactionCreate({ multisigPda: ids.multisig,
      transactionIndex: index, creator: member, vaultIndex: 0,
      ephemeralSigners: 0, transactionMessage: inner }),
    squads.instructions.proposalCreate({ multisigPda: ids.multisig,
      transactionIndex: index, creator: member })];
  const message = new TransactionMessage({ payerKey: member,
    recentBlockhash: PublicKey.default.toBase58(), instructions }).compileToV0Message();
  const bytes = new VersionedTransaction(message).serialize().length;
  assert.ok(bytes <= 1232, 'Proposal exceeds packet size; stop and revise atomic grouping.');
  return bytes;
}
async function main([action, configPath, planPath, expectedHash, index]) {
  assert.ok(configPath && planPath && expectedHash,
    'Usage: node operator.mjs ACTION CONFIG PLAN REVIEWED_SHA256 [PROPOSAL_INDEX]');
  const c = JSON.parse(await readFile(configPath, 'utf8'));
  const plan = JSON.parse(await readFile(planPath, 'utf8'));
  gates(action, c, plan, expectedHash, process.env);
  const ctx = await context(c);
  const bytes = await checkPlan(c, plan, ctx);
  if (action === 'review' || action === 'approve' || action === 'execute') {
    assert.match(index ?? '', /^[1-9][0-9]*$/);
    const state = await checkProposal(c, plan, index, ctx);
    console.log(JSON.stringify({ ...state, reviewedPlanSha256: expectedHash }, null, 2));
    if (action === 'review') return;
    if (action === 'approve') assert.equal(state.status, 'Active');
    else { assert.equal(state.status, 'Approved'); assert.ok(state.approved.length >= 2); }
  }
  // Recheck initial conditions and public metadata immediately before signing.
  const fresh = await build(c, plan.stage, ctx);
  assert.equal(fresh.messageBase64, plan.messageBase64,
    'State/rent changed. Create a fresh reviewed plan; do not force execution.');
  const version = spawnSync('squads-multisig-cli', ['--version'], { encoding: 'utf8' });
  assert.equal(version.status, 0, 'Install the reviewed Squads CLI first.');
```

```
  assert.match(version.stdout, /\b0\.1\.7\b/, 'Review any CLI version change.');
  const who = spawnSync('solana-keygen', ['pubkey', process.env.IUNUA_SIGNER],
    { encoding: 'utf8' });
  assert.equal(who.status, 0, 'Cannot identify the selected signer.');
  const member = who.stdout.trim(); assert.ok(c.members.includes(member));
  const args = ['--rpc-url', ctx.connection.rpcEndpoint,
    '--program-id', squads.PROGRAM_ID.toBase58(),
    '--keypair', process.env.IUNUA_SIGNER, '--multisig-pubkey', c.multisig];
  if (action === 'propose') {
    const ixs = fresh.instructions.map(ix => ({
      programId: new PublicKey(ix.program), data: Buffer.from(ix.dataHex, 'hex'),
      keys: ix.accounts.map(k => ({ pubkey: new PublicKey(k.address),
        isSigner: k.signer, isWritable: k.writable }))
    }));
    proposalSize(ctx, new PublicKey(member),
      BigInt(ctx.ms.transactionIndex.toString()) + 1n, ixs);
    args.unshift('vault-transaction-create'); args.push('--vault-index', '0');
    // Clap Vec<u8> consumes repeated options; no shell interpolation.
    for (const byte of bytes) args.push('--transaction-message', String(byte));
  } else {
    args.unshift(action === 'approve' ? 'proposal-vote' : 'vault-transaction-execute');
    args.push('--transaction-index', index);
    if (action === 'approve') args.push('--action', 'Approve');
  }
  const result = spawnSync('squads-multisig-cli', args, { stdio: 'inherit' });
  if (result.status !== 0) throw new Error(
    'Submission failed or is uncertain. Check proposal, signature and chain state before retrying.');
  console.log('Record the CLI signature and actual proposal index; wait for finalized evidence.');
}
if (process.argv[1] && import.meta.url === pathToFileURL(process.argv[1]).href)
  main(process.argv.slice(2)).catch(e => { console.error(e.message); process.exitCode = 1; });
```

Recorded dependency audit blockers

Package / advisory	Mainnet acceptance requirement
bigint-buffer / GHSA-3gc7-fjrx-p6mg	bigint-buffer Vulnerable to Buffer Overflow via toBigIntLE() Function; remediation or independently justified assessment and repeat testing.
stream-json / GHSA-528h-pc64-c93x	stream-json: pick/ignore/filter/replace filters are O(depth²) on nested input - small crafted JSON blocks the event loop for seconds→minutes (DoS); remediation or independently justified assessment and repeat testing.
uuid / GHSA-w5hq-g745-h8pq	uuid: Missing buffer bounds check in v3/v5/v6 when buf is provided; remediation or independently justified assessment and repeat testing.

security-review.json


```
{
  "checkedAt": "2026-09-09",
  "status": "blocked-unresolved-advisories",
  "approvalDocumentSha256": null,
  "counts": {
    "info": 0,
    "low": 0,
    "moderate": 7,
    "high": 4,
    "critical": 0,
    "total": 11
  },
  "advisories": [
    {
      "package": "bigint-buffer",
      "severity": "high",
      "url": "https://github.com/advisories/GHSA-3gc7-fjrx-p6mg",
      "id": "GHSA-3gc7-fjrx-p6mg"
    },
    {
      "package": "stream-json",
      "severity": "moderate",
      "url": "https://github.com/advisories/GHSA-528h-pc64-c93x",
      "id": "GHSA-528h-pc64-c93x"
    },
    {
      "package": "uuid",
      "severity": "moderate",
      "url": "https://github.com/advisories/GHSA-w5hq-g745-h8pq",
      "id": "GHSA-w5hq-g745-h8pq"
    }
  ]
}
```


Code validation on 9 September 2026: nine offline tests cover instruction ordering, authority assignment, exact 42,000,000 x 10^9 arithmetic, non-idempotent initial treasury creation, proposal serialization and byte matching, packet size, wrong-network/quorum controls, metadata corruption and operator gates. These tests neither submit transactions nor exercise Ledger hardware. The reference is not audited or production-certified. The dependency audit separately reported 11 affected dependency nodes across three advisory families. Mainnet operator actions are blocked pending remediation and independent review; see security-review.json.

Sources: IUNUA registration reference source v1.0; Solana: metadata pointer and token metadata; Solana: pausable mint; Solana: token verification and Explorer visibility; Squads V4 CLI: source and hardware-wallet interface; Squads: creating vault transactions; IUNUA registration dependency review record

16 / IUNUA SYSTEM & LAUNCH GUIDE

Readiness gates for the full trust service

These additional gates apply to the complete trust-currency service. A basic token and wallet pilot has a narrower scope.

Additional gates for the first audited treasury payment

Part	Required behavior / boundary
World authentication	A real server-verified World App account session, tested wallet handoff and explicit treasury role mapping. Human verification is checked separately when policy requires it.
XMR custody and signer	Approved owner, funding source, signing design, limits, co-approval rules, backup/recovery and monitored Monero infrastructure.
Payment verification	Expected-recipient/amount proof checks, trustworthy confirmation observations, durable request binding, replay prevention and reconciliation.
Solana receipt mechanism	Adopted schema and commitment construction, authorized verifier keys, durable publication, finalized-status checks and append-only correction rules.
End-to-end acceptance	A test-network rehearsal including partial failures and no double payment, independent review, then a separately authorized small real payment in the exact released app.

Required milestones and acceptance evidence

Milestone	Evidence required before real-value use
Specify policy and economic rights	Versioned decisions for issuance, rewards, fees, treasury, migration and any user obligations or rights.
Implement the private service	Authenticated access, durable transactional accounting, scoped authorization, recovery, reconciliation and privacy-aware logging.
Verify eligibility	Real provider and event adapters; adversarial tests for replay, wrong purpose, expired proofs, duplicate claims and invalid evidence.
Complete wallet and game flows	Explicit account linking and approval; correct network and amount; actual signing, confirmation, failure handling and recovery in exact release packages.
Implement settlement and authority controls	Reviewed programs/adapters, constrained custody, supply accounting, pause behavior, governed upgrades and preserved historical policy references.
Validate safety independently	Security and cryptographic review, resolved critical findings, reproducible builds and realistic end-to-end testing.

Required milestones and acceptance evidence	
Milestone	**Evidence required before real-value use**
Prepare operations and governance	Monitoring, backups with restore drills, incident ownership, key rotation, dispute handling and approved change procedures.
Approve a bounded production launch	Applicable legal/privacy assessment, published user terms, explicit governance authorization and defined rollout limits.

These are prerequisites for the complete production trust service, not a requirement to finish every future feature before issuing a basic token. The initial token-and-wallet pilot must satisfy its own custody, signing, distribution and operational checks and clearly exclude unfinished capabilities.

Executable swaps need a separately verified route and failure handling. Complete game payments need explicit wallet approval and reconciliation. These can be delivered after a bounded token launch, but must not be presented as available beforehand.

Writing or publishing this paper changes documentation only. It does not activate a signing route, remove authorities, issue a mainnet token, establish token-holder rights or change the website's private access policy.

Sources: IUNUA token and wallet capability status; IUNUA downloads and installation evidence; IUNUA developer integration guide

17 / IUNUA SYSTEM & LAUNCH GUIDE

Glossary: the words behind the idea

Everyday definitions for the terms used in this draft.

Key terms	
Term	**Meaning**
Trust currency	The project's proposed system for recording qualifying contributions and permitted uses under shared rules.
IUNUA	The ecosystem and proposed mainnet token name/symbol.
Mint	The network account defining a token and its shared properties.
Mint authority	The authority permitted to issue additional tokens. Retaining it is different from enforcing a fixed cap.
Devnet / mainnet	Separate testing and production networks. A Devnet balance is not a mainnet balance.

Key terms	
Term	**Meaning**
Token-2022	Solana's token program supporting extensions. Only enabled extensions apply to a particular mint.
Private ledger	The proposed authoritative accounting record, with access limited by implemented controls.
Commitment	A cryptographic representation intended to bind a value while concealing it under stated assumptions. No production construction is established here.
Nullifier	A unique marker intended to prevent reusing the same permitted claim or spend.
Eligibility proof	Evidence checked for a specific condition; it is not automatically evidence that useful work occurred.
Settlement	Completion and reconciliation of a value transfer under its applicable rules.
Audit grant	Proposed permission to disclose specified information to a named reviewer for a defined purpose.
Pantheon	The proposed framework for reviewing and authorizing policy or program changes.
Timelock	A required delay between approval and activation; the production duration is undecided.
BFF / MCP	Backend for frontend / Model Context Protocol. Here, proposed backend architecture and a separate local simulation integration respectively.
Notarization / checksum	Distribution review evidence / a file identity check. Neither establishes protocol or financial readiness.
XMR treasury	The intended Monero funds account or custody system; separate from the recorded Solana Devnet game treasury.
Service record	The IUNUA/Solana record of a named service: purpose, amount, optional fee and chosen rail. It does not by itself move XMR.
Value rail	The network that actually moves funds. In the wallet test composition this is Stagenet XMR or IUNUA-T only; production SOL stays locked.
Swap estimates	Informational mid-market quotes. A visible estimate does not enable execution.
Wallet-rpc client	The iPhone talks to owner-started Monero wallet RPC. Spend keys stay on that host.
Receipt commitment	A salted cryptographic fingerprint of defined evidence; matching bytes alone do not establish the truth of the claim.
Verifier attestation	A signed statement by an identified verifier; its trust assumptions must be disclosed.

Key terms	
Term	**Meaning**
Idempotency	A repeated request has no additional economic effect; an unknown payment must be reconciled before another is sent.

Sources: IUNUA intended architecture; IUNUA developer integration guide; Solana documentation: create a token mint; Solana documentation: clusters and public RPC endpoints; Solana documentation: token extensions

References and evidence dates

Project evidence establishes project state. External documentation explains reference technology.

The project whitepaper and the separate System & Launch Guide share the same dated technical evidence. The wallet package assessment remains dated 8 September 2026. Mint, treasury and mainnet rent observations are separately dated 9 September 2026. Native Apple wallet source for iPhone navigation, QR scan, locked Swap and agreement composition is dated 10 September 2026. This document separation does not change software, economic policy or chain state.

This edition was revised on 10 September 2026. Current-state statements are limited to the evidence and dates identified here. Source checkout changes, a later explorer view or a different package may show a different state. This is a project draft; unresolved decisions remain unresolved.

Earlier project documents describing only a simulator or stating that no token exists predate the wallet and Devnet milestones. They remain useful historical design notes, but this edition uses the dated package assessment and chain evidence for current-state claims. Planned components in those notes have not been promoted to available functionality.

IUNUA token and wallet capability status

Release review: 8 September 2026. Website figures are separately dated snapshots.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/token-status

IUNUA release manifest

Generated 8 September 2026, 17:14:46 UTC. Exact versions, archive sizes and SHA-256 hashes.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/downloads/client-manifest.json

IUNUA downloads and installation evidence

Package and signing review: 8 September 2026. Version numbers alone do not establish package identity.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/downloads

IUNUA launch preparation: read-only chain evidence

Devnet mint and treasury, mainnet genesis/address absence and rent queries. Checked 2026-09-09T09:24:23.195685+00:00. No transaction submitted.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/documents/iunua-mainnet-preparation-evidence-2026-09-09.json

IUNUA mint on Solana Explorer (Devnet)

Explorer displays current network information, which may differ from the paper snapshot.

https://explorer.solana.com/address/9UePLTf7PRfbZUDzRq8f4sVo3CE7DsZvEtY7ypYZprrk?cluster=devnet

Game treasury token account on Solana Explorer (Devnet)

Public test token account; not an individual player balance.

https://explorer.solana.com/address/8NcU2LK62iNV2ySjHZKgKf72r4yP7tP8dELREZvyr1nW?cluster=devnet

IUNUA intended architecture

Project design, not production implementation evidence.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/architecture

IUNUA Trust Lab

Seven-step browser simulation; no real proof verification or settlement.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/trust-lab

IUNUA developer integration guide

Local Game API and MCP simulation boundaries; source presence is not a released payment feature.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/developers

Solana documentation: create a token mint

Accessed 9 September 2026. Mint state, supply, decimals and authorities.

https://solana.com/docs/tokens/basics/create-mint

Solana documentation: clusters and public RPC endpoints

Accessed 9 September 2026. Distinct networks and Devnet testing limitations.

https://solana.com/docs/references/clusters

Solana documentation: token extensions

Accessed 9 September 2026. General Token-2022 capabilities do not prove an extension is enabled for IUNUA.

https://solana.com/docs/tokens/extensions

World ID documentation: overview

Accessed 9 September 2026. Reference concept; no partnership or completed IUNUA production integration is asserted.

https://docs.world.org/world-id/overview

Solana token operations

Accessed/prepared 9 September 2026. Holdings, transfers, minting, burning, delegation and account control. General protocol capabilities are not evidence of an app button.

https://solana.com/docs/tokens/basics

Solana pausable mint

Accessed/prepared 9 September 2026. Pause rejects transfers, mints and burns; resume restores permitted activity.

https://solana.com/docs/tokens/extensions/pausable

Solana token metadata and pointer

Accessed/prepared 9 September 2026. Metadata fields, pointer and update authority. Authority removal can make metadata immutable.

https://solana.com/docs/tokens/extensions/metadata

Solana set authority

Accessed/prepared 9 September 2026. Role-specific authority changes and revocations.

https://solana.com/docs/tokens/basics/set-authority

Solana transaction fees

Accessed/prepared 9 September 2026. 5,000 lamports per signature reference and prioritization-fee calculation.

https://solana.com/docs/core/fees

Solana rent-exemption RPC

Accessed/prepared 9 September 2026. Rent must be queried for the actual account size. The included quote uses stated planning sizes.

https://solana.com/docs/rpc/http/getminimumbalanceforrentexemption

Helius published RPC prices

Accessed/prepared 9 September 2026. Developer USD 49/month; Business USD 499/month. Usage limits apply.

https://www.helius.dev/pricing

Squads published pricing

Accessed/prepared 9 September 2026. Basic USD 0/month plus 0.1 SOL deployment; Pro USD 49/month. Confirm exact extension support in rehearsal.

https://docs.squads.so/main/getting-started/pricing

ESMA MiCA rulebook

Accessed/prepared 9 September 2026. EU public offers and admission to trading need a jurisdiction-specific assessment. This technical draft is not a regulatory filing or legal clearance.

https://www.esma.europa.eu/publications-and-data/interactive-single-rulebook/mica

IUNUA preparation preflight and cost assumptions

Accessed/prepared 9 September 2026. Machine-readable blocked readiness and read-only mainnet rent quote. No execution capability.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/documents/iunua-mainnet-preflight.json

IUNUA mainnet preparation kit

Accessed/prepared 9 September 2026. Public draft configuration, metadata template, validation tool, tests, runbook and evidence. No private keys or executable launch authorization.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/documents/iunua-mainnet-preparation-kit.zip

IUNUA registration reference source v1.0

Prepared 9 September 2026. Complete handwritten builder/operator code, configuration templates, dependency lockfile, offline tests and step-by-step runbook. No transaction or Ledger rehearsal performed.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/documents/iunua-mainnet-registration-v1.0.zip

Solana: metadata pointer and token metadata

Checked 9 September 2026. Metadata authority, variable account sizing and prefunding for rent.

https://solana.com/docs/tokens/extensions/metadata

Solana: pausable mint

Checked 9 September 2026. Extension initialization order and pause behavior for mint, transfer and burn.

https://solana.com/docs/tokens/extensions/pausable

Solana: token verification and Explorer visibility

Checked 9 September 2026. Creating a mint and obtaining third-party search/verification status are distinct.

https://solana.com/docs/tokens/how-to-verify-a-token

Squads V4 CLI: source and hardware-wallet interface

Checked 9 September 2026; package source squads-multisig-cli 0.1.7 inspected through crates.io. The operator pins that version; CLI/hardware execution still needs rehearsal.

https://github.com/Squads-Protocol/v4/blob/main/cli/README.md

Squads: creating vault transactions

Checked 9 September 2026. Vault proposals hold arbitrary instructions; approval and execution are distinct. The implementation uses pinned @sqds/multisig 2.1.4 APIs.

https://docs.squads.so/main/development/typescript/instructions/create-vault-transaction

IUNUA registration dependency review record

Recorded 9 September 2026 in security-review.json. Mainnet operator actions remain blocked while the dependency review is unresolved. Original advisory links are included in that record.

https://iunua-trust-currency.amun-ra-14th.chatgpt.site/documents/iunua-mainnet-registration-v1.0.zip

World IDKit integration

Reviewed 9 September 2026. Provider documentation establishes the integration mechanism; it does not certify an IUNUA deployment.

https://docs.world.org/world-id/idkit/integrate

World React QR widget

Reviewed 9 September 2026. Provider documentation establishes the integration mechanism; it does not certify an IUNUA deployment.

https://docs.world.org/world-id/idkit/react

World v4 server verification API

Reviewed 9 September 2026. Provider documentation establishes the integration mechanism; it does not certify an IUNUA deployment.

https://docs.world.org/api-reference/developer-portal/verify

World App wallet authentication

Reviewed 9 September 2026. Provider documentation establishes the integration mechanism; it does not certify an IUNUA deployment.

https://docs.world.org/mini-apps/commands/wallet-auth

World Chain overview

Reviewed 9 September 2026. Provider documentation establishes the integration mechanism; it does not certify an IUNUA deployment.

https://docs.world.org/world-chain/index

Monero wallet RPC: payment and reserve proofs

Reviewed for the 10 September 2026 architecture revision. Provider capabilities do not establish an implemented IUNUA integration.

https://www.getmonero.org/resources/developer-guides/wallet-rpc.html

Monero privacy and wallet keys

Reviewed for the 10 September 2026 architecture revision. Provider capabilities do not establish an implemented IUNUA integration.

https://www.getmonero.org/resources/moneropedia/stealthaddress.html

Solana transactions and instruction execution

Reviewed for the 10 September 2026 architecture revision. Provider capabilities do not establish an implemented IUNUA integration.

https://solana.com/docs/core/transactions