

# Eduardo Melo

## Blockchain engineering in products and protocols.

I lead blockchain engineering at Philcoin and work on an EVM Layer 1 in Rust. These pages show my part in each project, including the parts hidden inside private repositories.

[eduardo.melo00@gmail.com](mailto:eduardo.melo00@gmail.com)

[linkedin.com/in/eduardofmelo](https://linkedin.com/in/eduardofmelo)

Florianópolis, Brazil

|    |                    |                                                                                              |
|----|--------------------|----------------------------------------------------------------------------------------------|
| 01 | Founders Club      | I built the purchase flow, API, escrow contracts and NFT certificate.                        |
| 02 | Phil app · Rewards | I connected mobile claims to Polygon payments, then worked through retries and confirmation. |
| 03 | Ambassador Program | Attribution, a PostgreSQL ledger and KMS-signed treasury payments.                           |
| 04 | PHL → GIVE Swap    | Position contracts on Polygon and proof-based claims on BSC.                                 |
| 05 | YouthChain         | Rust consensus and EVM execution on a four-validator testnet.                                |
| 06 | Rise               | Token deposits converted to USDC, then moved through CCTP across more than seven networks.   |
| 07 | FearsOff           | Manual and automated reviews of exchange and DeFi contracts, with fixes for the teams.       |
| 08 | Orium Network      | Rental contracts across three chains and a subgraph indexing more than 15 event types.       |
| 09 | Seedify            | SFUND staking reviews, the GMFAM token and two NFT collections.                              |

Most repositories are private. Public links point to products, contracts and transactions where available. The flows are simplified walkthroughs.

# Founders Club

A buyer pays first; the certificate follows identity approval.

I built the frontend, backend and smart contracts that turn a membership purchase into an identity-verified NFT certificate and benefits inside the Phil app.

Head of Blockchain · Philcoin

2026

## My part

### Product interface

Built the purchase experience, wallet connection, account authentication and membership certificate views.

### Backend & integration

Connected Cognito authentication, Ddidit identity verification, webhook handling and the Phil API.

### Smart contracts

Implemented upgradeable membership, escrow, tier upgrades and NFT certificate contracts.

## System flow

### 01. Purchase

The member chooses a tier and pays through the membership contract. For a new buyer, the funds are held while identity verification is pending.

### 02. Verify identity

A verified identity-provider webhook updates eligibility. The backend connects the identity decision to the membership and the Phil account.

### 03. Issue certificate

After approval, the membership flow releases escrow to the treasury and issues the NFT membership certificate.

### 04. Connect benefits

Membership state is reflected in the Phil profile. PHL benefits are handled through the rewards service rather than assumed to come from the NFT contract.

### Delivery

The purchase, identity verification, certificate minting and treasury-release flow reached Polygon production. Membership benefits connect back to the Phil ecosystem.

## Links and records

**Live product:** [Open the live Founders product](#). Public membership portal. No account or wallet is needed to see the entry experience.

**On-chain artifact:** [Membership contract on Polygon](#). Deployed membership address recorded in the production launch.

Solidity · React · FastAPI · Polygon · Ddidit KYC · AWS KMS · Cognito

The walkthrough explains the implementation. It does not connect a wallet or submit transactions.

# Phil app · Rewards

A reward claim has to survive a timeout without paying twice.

I own the claim pipeline connecting the Phil mobile app, API, rewards service and Polygon contracts. My work covers contract upgrades, transaction reliability and reconciliation back into the application.

Head of Blockchain · Philcoin

Jun 2025 to present

## My part

### Claim pipeline

Owned the integration from the app request through the API and rewards server to the contract.

### Contract delivery

Implemented and deployed the V2.1 rewards upgrade, including a usable rollback target.

### Production operations

Fixed gas and retry behavior, hardened duplicate prevention and added event-based claim confirmation.

## System flow

### 01. Request reward

A member requests a claim. The application checks the reward and records the request before it reaches the transaction service.

### 02. Submit once

The rewards service coordinates the transaction, with deduplication and gas handling designed around the fact that an earlier request may still be mining.

### 03. Execute on-chain

The rewards contract processes the claim. The V2.1 implementation changed the claim execution path while preserving the existing proxy integration.

### 04. Reconcile

The confirmer reads ClaimExecuted events at finalized blocks and brings the application record into agreement with the chain.

### Delivery

Shipped the RewardsLibrary V2.1 upgrade to Polygon and strengthened the claim pipeline with deduplication and a confirmer that reconciles finalized contract events with application records.

## Links and records

**Deployment transaction:** [Rewards V2.1 deployment](#). Recorded mainnet deployment transaction from the May 2026 upgrade.

**Upgrade transaction:** [Proxy implementation update](#). Recorded transaction activating the new rewards implementation.

[Solidity](#) · [Polygon](#) · [Python](#) · [TypeScript](#) · [React Native integration](#) · [QuickNode](#) · [AWS](#)

App users, private service endpoints and confidential incident details are excluded.

# Ambassador Program

Each payout has to trace back to the commission that earned it.

I built the commission and payout system behind the Phil Ambassador Program: attribution, a commission ledger, member APIs, treasury transfers and the member and operations interfaces.

Head of Blockchain · Philcoin

Aug 2026 to present

## My part

### Commission accounting

Built network attribution, commission accrual and the ledger behind member earnings.

### Treasury payments

Implemented payout reservation, KMS-signed token transfers, transaction tracking and receipt reconciliation.

### Product surfaces

Built member views and operations tooling, connecting eligibility, balances, payout history and administrative appointment.

## System flow

### 01. Attribute activity

Qualifying activity is associated with the correct ambassador network before a commission is accrued.

### 02. Record earnings

Commission entries are kept in the database ledger. A payout reserves the relevant balance transactionally.

### 03. Sign & submit

The treasury transfer is signed through AWS KMS. The transaction is identified before broadcast so submission and later reconciliation refer to the same payment.

### 04. Confirm payout

The monitor checks the transaction receipt and reconciles the payout status shown to the member.

### Delivery

The program backend and backoffice API reached production in September 2026. Member and admin interfaces were delivered to development; production promotion is a separate release step.

## Links and records

**Implementation walkthrough:** The ledger-to-treasury flow above. Reconstructs the delivered accounting and payout architecture from the implementation records.

**Release scope:** Backend shipped; interfaces in development. September release records distinguish deployed APIs from the development-stage member and admin screens.

[FastAPI](#) · [PostgreSQL](#) · [AWS KMS](#) · [ERC-20](#) · [Vue](#) / [Ionic](#) · [Next.js](#) · [Go backoffice](#)

Backend and backoffice API in production; member and administrative screens delivered to development.

# PHL → GIVE Swap

Each PHL swap keeps its own destination, unlock time and claim.

I built the interface and contracts for converting PHL on Polygon into time-locked claims on a destination chain, including wallet pairing, agreement recording and Merkle proofs.

Head of Blockchain · Philcoin

2026

## My part

### Swap experience

Built the interface, Phil wallet pairing, position display and agreement-signing flow.

### Position contracts

Implemented swap positions and locked receipts with a separate unlock time and destination for each position.

### Destination claims

Implemented snapshot-based Merkle claims, duplicate-claim protection and deployment across Polygon and BSC.

## System flow

### 01. Record agreement

The wallet signs the acknowledgement and records the document hash before the swap. Personal information is not written as plain text on-chain.

### 02. Create position

The swap creates a position carrying its amount, destination chain and unlock time. The interface reads positions directly from the contract.

### 03. Publish proof

A snapshot becomes a Merkle root published to the destination claim contract by its administrator.

### 04. Claim once

After the position unlocks, the contract verifies its proof, marks the leaf as claimed and sends the destination tokens to the recipient.

### Delivery

Deployed the public interface, Polygon swap and receipt contracts, and a funded BSC demo claim contract. Demo parameters: a ten-minute lock and TGIVE destination token.

## Links and records

**Live demonstration:** [Open the GIVE Swap demo](#). Public demo interface. Inspect without connecting a wallet or making a transaction.

**On-chain artifact:** [Polygon swap contract](#). The deployed v3 swap contract referenced by the demo.

**On-chain artifact:** [BSC claim contract](#). Destination claim contract for the v3 demonstration.

[Solidity](#) · [Foundry](#) · [React](#) · [Vite](#) · [Wagmi](#) · [WalletConnect](#) · [Polygon](#) · [BSC](#) · [Merkle proofs](#)

Demonstration configuration: ten-minute lock and TGIVE demo token on BSC. This is not represented as the final production rollout.

# YouthChain

Four validators must reach the same state after every transaction.

I work on an EVM-compatible Layer 1 written in Rust, spanning DAG-based BFT consensus, BLS signatures, deterministic execution and the infrastructure that lets applications use the chain.

Core Protocol Engineer · part-time

2026 to present

## My part

### Consensus engine

Co-designed and implemented DAG-based BFT consensus with BLS signature aggregation.

### Execution correctness

Traced state-root mismatches to transaction execution and made replay deterministic by construction.

### Protocol-to-app connection

Built protocol contracts, JSON-RPC integration and an off-chain sequencer with an engagement rollup.

## System flow

### 01. Receive

An application submits a transaction through the chain interface. Nodes propagate the input for the consensus process.

### 02. Agree

Validators participate in DAG-based consensus, using BLS signatures to certify agreement.

### 03. Execute

Agreed transactions run through the execution layer. The deterministic path gives the same ordered history the same state transition.

### 04. Expose result

The resulting state becomes available to applications. Engagement batching connects mobile activity to the on-chain accounting path.

### Delivery

Implemented and exercised the consensus, EVM execution and engagement pipeline on a four-validator testnet, including validator disruption and recovery. Production-readiness work remains separate.

## Links and records

**Architecture walkthrough:** Consensus to application state. The interactive flow above describes the implemented system boundary and the determinism decision.

**Development evidence:** Four-validator testnet exercises. Internal testnet records cover load, validator interruption, recovery and execution consistency. Private infrastructure is not linked publicly.

[Rust](#) · [EVM](#) · [DAG / BFT](#) · [BLS](#) · [JSON-RPC](#) · [P2P](#) · [Solidity](#) · [Docker](#) · [AWS](#)

Development-stage testnet. Mainnet readiness and production performance are outside this case's claims.

# Rise

Making payroll work when deposits arrive in different tokens and chains.

I worked across contracts and backend services for crypto payroll, implementing multi-token deposits into USDC and Circle CCTP v1 transfers across more than seven networks.

Full-Stack Solidity Developer

Feb 2025 to Aug 2025

## My part

### Deposit contracts

Architected a Uniswap v4 integration for multi-token deposits that settle into USDC.

### Cross-chain transfers

Implemented Circle CCTP v1 integration for the payroll flow across supported networks.

### Backend orchestration

Worked from upgradeable proxy contracts through TypeScript APIs, database records and asynchronous jobs.

## System flow

### 01. Deposit token

The customer starts with a supported ERC-20 token rather than first managing a separate conversion outside the payroll product.

### 02. Settle into USDC

The integration converts the input into the settlement asset used by the payroll flow.

### 03. Transfer across chains

The backend coordinates the source-chain burn, attestation and destination-chain mint steps through asynchronous processing.

### 04. Track completion

The service tracks transfer progress through to the destination so the payroll application can represent the result.

### Delivery

Delivered deposit and cross-chain payroll integrations across the contract and backend stack, covering 50+ input tokens and more than seven networks.

## Links and records

**Product context:** [Rise payroll platform](#). Public product context. The private implementation and contribution scope are described in this case.

**Career record:** 50+ tokens · 7+ networks. Scope recorded in my CV for the February to August 2025 engagement; historical figures, not current platform-wide claims.

[Solidity](#) · [Uniswap v4](#) · [Circle CCTP v1](#) · [TypeScript](#) · [Node.js](#) · [Prisma](#) · [Inngest](#) · [Foundry](#)

The flow explains my integration work without exposing customer transactions or private payroll data.

# FearsOff

The review followed permissions, balances and unusual state changes.

I performed smart contract security reviews for exchange and DeFi engagements through FearsOff, combining manual analysis, automated tooling and remediation guidance.

**Web3 & Smart Contract Auditor**

Dec 2024 to Feb 2025

## My part

### Manual security review

Reviewed contract behavior, authorization boundaries and paths that affect assets.

### Adversarial tooling

Used Slither, Mythril, Echidna and Manticore, with formal verification techniques on critical functions.

### Actionable reports

Delivered findings with remediation and gas-optimization guidance for engineering teams.

## System flow

### 01. Map assumptions

Identify the assets, privileged roles and state transitions that define the contract's trust boundaries.

### 02. Challenge behavior

Combine manual reasoning with automated tools to search for paths that violate those assumptions.

### 03. Assess impact

Connect a suspected issue to a specific affected asset, permission or accounting path and prioritize its consequence.

### 04. Guide remediation

Translate the technical finding into concrete implementation guidance for the team responsible for the contract.

### Delivery

Delivered audits and remediation guidance for engagements covering Crypto.com exchange contracts, HTX integration modules and Smardex AMM and liquidity contracts.

## Links and records

**Career record:** Exchange and DeFi audit scope. Client scope and methods are recorded in my CV. Engagements were performed through FearsOff.

**Review walkthrough:** Method and decision process. The workflow above explains my approach without revealing confidential reports, exploit details or unremediated findings.

[Solidity](#) · [Slither](#) · [Mythril](#) · [Echidna](#) · [Manticore](#) · [Formal verification](#) · [Gas analysis](#)

Audit reports and vulnerability details remain confidential.

# Orium Network

NFT rentals needed contracts that games could use and data they could query.

I contributed to a multi-chain NFT rental and lending protocol, then connected contract events to the application through The Graph and partner integrations.

**Solidity Developer**

Jul 2024 to Oct 2024

## My part

### Protocol development

Worked on Solidity components for NFT rental and lending across three EVM networks.

### The Graph integration

Led event indexing and designed the subgraph schema for application and analytics queries.

### Partner mechanics

Supported gaming partners with rental behaviors suited to their integrations.

## System flow

### 01. Interact

A user initiates a supported rental or lending action through the protocol.

### 02. Update contract

The relevant contract changes its state and emits the events that describe the action.

### 03. Index events

The subgraph maps more than 15 event types into entities that applications can query.

### 04. Serve the product

Partners use the indexed view to support their rental interfaces and analytics across supported chains.

### Delivery

Contributed to the protocol on Ethereum, Polygon and Arbitrum and led a subgraph schema covering more than 15 event types. Supported custom rental mechanics for gaming partners.

## Links and records

**Career record:** Three chains · 15+ event types. Implementation scope recorded in my CV for the July to October 2024 engagement.

**Architecture walkthrough:** Contract-to-subgraph flow. The flow explains the boundary between authoritative contract state, indexed entities and partner applications.

[Solidity](#) · [The Graph](#) · [Ethereum](#) · [Polygon](#) · [Arbitrum](#) · [NFT rental](#) · [Event indexing](#)

Simplified architecture walkthrough; partner data and private repositories are not included.

# Seedify

Staking, tokens and NFT launches needed the same review discipline.

I led blockchain developers and the contract security review process across staking, token launches and NFT releases, while remaining hands-on with implementation and deployment.

**Solidity Developer & Team Lead**

Oct 2022 to Oct 2023

## My part

### Blockchain team leadership

Led the development team and owned the security review process across contract deliveries.

### Staking & token work

Worked on SFUND staking security and the GMFAM token launch.

### NFTs & release tooling

Deployed two NFT collections and implemented CI/CD for smart contract releases.

## System flow

### 01. Define mechanics

Translate staking, token or collection requirements into explicit contract behavior and permissions.

### 02. Implement

Build the required contract components while coordinating implementation across the blockchain team.

### 03. Review

Apply contract review as part of delivery, including the paths that can affect staked assets or privileged operations.

### 04. Release

Use deployment tooling and repeatable release processes to ship the contracts and collections.

### Delivery

Delivered staking security work, the GMFAM token and two NFT collections. Historical project scale: \$114M+ in staking TVL covered by the work and 2,472 ETH in collection volume.

## Links and records

**Public collection:** [The Mounts of Seedworld](#). Public collection artifact associated with the ecosystem work.

**Token context:** [SFUND on BscScan](#). The SFUND token address. This is not the staking contract.

[Solidity](#) · [Staking](#) · [ERC-20](#) · [ERC-721](#) · [Smart contract auditing](#) · [CI/CD](#) · [Team leadership](#)

TVL and collection volume refer to the historical engagement period, not current balances or personal revenue.