Learn blockchain by building the rules that make lying useless

If you work with crypto teams, vendors, or blockchain products, you do not need more opinions, you need X-ray vision. This course gives you that by walking you through the construction of a blockchain node from scratch, with each layer explained in plain language and reinforced with practical labs. Developers gain a buildable blueprint, managers gain the ability to ask the right questions and spot bad designs, and integrators gain a clean mental model for how blocks, transactions, wallets, and explorers actually interact.

Most blockchain courses explain concepts and leave you guessing how the pieces fit. This one forces the pieces to fit because you build them in the only order that really works: data structures first, then validation and state transitions, then mining and consensus, then networking, forks, reorgs, syncing, and finally the user-facing tools (API, wallet, explorer).

You can follow the concept track with no coding, or take the builder track and implement a real node that interoperates with other students' nodes. By the end, you will not just "know what a blockchain is", you will know exactly why each rule exists and what breaks when you remove it.

Who This Course is For

Teachers, trainers, and technical content creators

- You want a structured sequence with clean mental models, labs, and test questions you can reuse.

Product managers, tech leads, engineering managers

- You need to make decisions, review designs, and challenge vague claims.
- You want to understand tradeoffs (PoW vs PoS vs participation models) without getting buried in math.

Founders and investors who want to evaluate blockchain projects

- You want to distinguish real engineering from marketing.
- You want a checklist for what a serious chain must implement to be credible.

Integrators and solution architects

- You need to connect wallets, services, and infrastructure to a chain.
- You want to know what is safe to assume, what is never safe to assume, and where failures happen.

Security engineers and auditors

- You want a concrete map of attack surfaces: replay, double spend races, eclipse, DoS, fork games, and what defenses actually work.

Software developers (backend, full-stack, systems)

- You want real architecture, not just theory.
- You want to understand validation, mempool, mining, networking, and reorgs well enough to implement them.

Smart contract developers who feel “the base layer is a black box”

- You want to understand what happens beneath your contract logic.

What the course will give you

For developers

- A clear, reusable blueprint for a blockchain node:
  - state, transactions, blocks, deterministic hashing
  - stateless and stateful validation
  - mempool and fee policy
  - mining and verification
  - P2P networking, gossip, syncing
  - fork handling and safe reorgs
  - persistence, indexes, snapshots
  - HTTP API, wallet flow, explorer indexing
- The ability to port the design to your preferred language stack
- Debugging instincts for distributed truth (why two honest nodes disagree, and how they converge)

For managers, leads, and architects

- A correct mental model of what a blockchain does and what it cannot do
- A practical vocabulary that maps to real components (mempool, tip, reorg, cumulative work, sync)
- The ability to review proposals and spot missing pieces:
  - “Where is replay protection?”
  - “How do you handle reorgs and persistence?”
  - “How does a new node sync safely?”
- Better decision-making around consensus choices and tradeoffs

For integrators and platform teams

- The end-to-end flow from wallet to node to mempool to block to explorer
- Clear expectations for node APIs and data contracts
- A realistic view of latency, finality, and error cases (and how to handle them in products)

For security-focused roles

- A structured attack map with concrete defenses:
  - replay and nonce rules
  - double-spend race mechanics
  - eclipse isolation and peer selection
  - DoS patterns and rate limits
  - fork games and 51 percent realities
- The habit of designing for adversarial inputs, not good-faith inputs

For teachers and communicators

- A lesson roadmap that builds understanding without hand-waving
- Repeatable exercises, test questions, and demos that show “why this rule exists”
- A narrative that makes blockchain teachable without hype

Blockchains look mysterious from the outside because you usually only see the final product: a network that agrees on a shared history without trusting any single operator. The fastest way to understand what is really happening is to build the pieces yourself, in the right order. In this course you will go from “what data exists” to “how it changes safely” to “how it spreads and converges across many computers”, until you can explain and implement the full lifecycle of a real blockchain node.

Building a blockchain from scratch is not hard because of one “magic algorithm”. It is hard because the system is made of layers that only make sense when they are built on top of each other. If you start with consensus too early, you argue about how to choose a winner before you define the game. If you start with networking too early, you end up debugging sockets before you even have a correct transaction format to send.

This course fixes that by following a deliberate build order:

- **Data structures (what exists):** define blocks, transactions, hashes, and identifiers so every node can describe the same reality in the same bytes.
- **State transitions (how it changes safely):** define the rules that decide what is valid, how balances change, and how replay and double-spend attempts get rejected.
- **Distribution (how it spreads and converges):** only after one node is correct, you make many nodes talk, sync, disagree briefly, then converge again using deterministic rules.

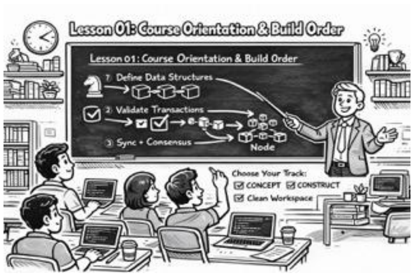
You can take the course in two modes, and switching is always allowed:

- **Concept track:** no coding required; you focus on the architecture, rules, and “why” behind each component.
- **Builder track:** you implement a working node using the Python reference code (and you can port it to any language you prefer). Your node is designed to interoperate with other students’ nodes, so the class can form a shared network together.

Each lesson is a 2-hour class with:

- **Intro:** what the lesson is about and why it matters
- **Outcomes:** what you should be able to do by the end
- **Python Code:** code to download to implement what learned
- **Explanation:** the full logic and mental models (with misconceptions and decentralization rules)
- **Practical:** what you build or simulate, with conceptual pseudocode (and separate downloadable Python code when needed)
- **Questions:** quick checks to confirm understanding
- **Next lesson hook:** what you will unlock next

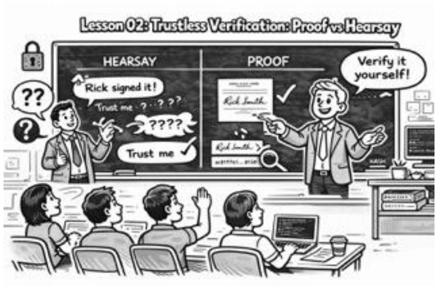
Lesson 01: Course orientation and the build order



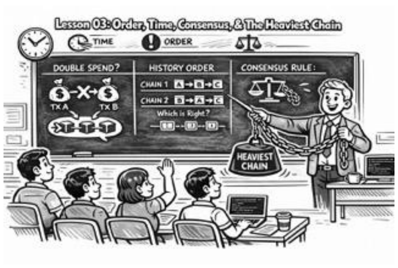
You set the rules of the game before you touch details: what you are building, the order you will build it in, and why that order prevents dead ends and confusion. You also choose your track (concept or builder) and set up a clean workspace so every future lesson builds smoothly on the last.

Lesson 02: Trustless verification, proof vs hearsay

You learn the core mindset of decentralized systems: assume anyone can lie, and verify everything yourself. This lesson makes signatures and hashes feel real by separating “someone told me” from “I can prove it with math”, which becomes the foundation for every validation rule later.



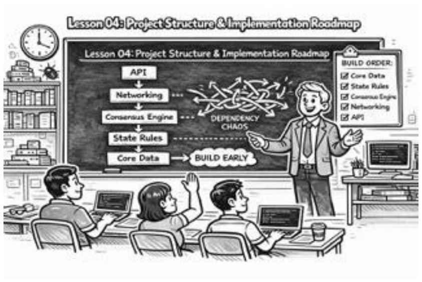
Lesson 03: Order, time, consensus, and the heaviest chain



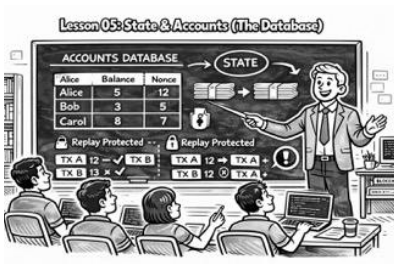
Signatures prove who authorized a message, but they do not solve “which happened first”. You learn why ordering is the real enemy (double spends), what consensus actually means (agreeing on history order), and why “heaviest chain wins” is the healing rule that lets networks converge after temporary splits.

Lesson 04: Project structure and implementation roadmap

You design your codebase like a system, not like a script. This lesson prevents dependency chaos by separating core data, state rules, consensus engine, networking, and API, and it gives you a build order that avoids circular imports and lets you test progress early.



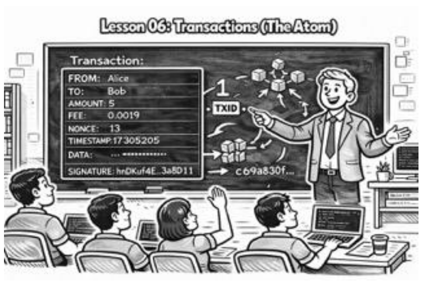
Lesson 05: State and Accounts (the database)



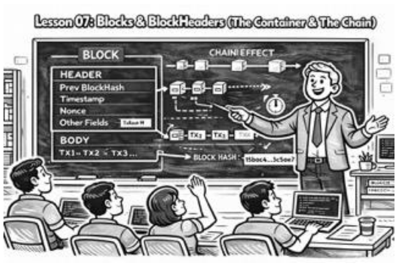
You build the ledger that your node protects: balances and nonces per account. This lesson teaches the “state” as a deterministic database that must produce the same results on every honest node, and why nonce tracking is not optional if you want replay protection.

Lesson 06: Transactions (the atom)

You define the smallest unit of change: a transaction as an intent to update state. You learn what every field is for (from, to, amount, fee, nonce, timestamp, data, signature) and how a deterministic transaction id (txid) is computed so nodes can deduplicate and reference transactions reliably.



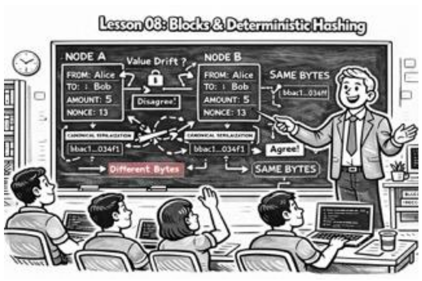
Lesson 07: Blocks and BlockHeaders (the container and the chain)



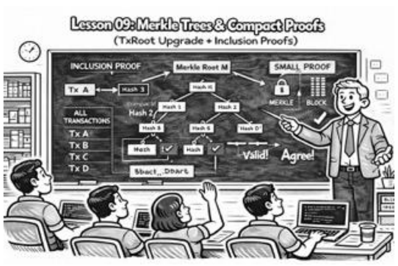
You package transactions into blocks and learn why headers exist: fast verification and chain linking without downloading everything first. You also lock in the key rule that the block hash is the header hash, and the body is represented by a summary field (TxRoot).

Lesson 08: Blocks and deterministic hashing

You learn the hidden rule that keeps decentralized systems from drifting: deterministic serialization. Two honest nodes can still disagree if they hash different bytes, so this lesson forces you to define canonical encoding and proves why “same values” must mean “same bytes”.



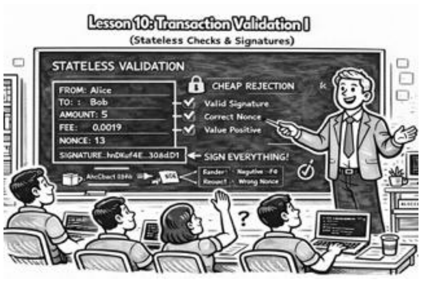
Lesson 09: Merkle trees and compact proofs (TxRoot upgrade + inclusion proofs)



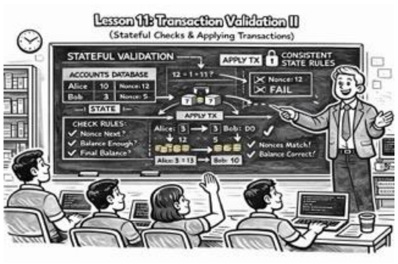
You upgrade TxRoot from a simple hash-of-concatenation into a Merkle root so you can prove membership without downloading all transactions. This is where you learn compact proofs: how to verify "this tx is in this block" using only a small path of hashes, which is crucial for scalability and light clients.

Lesson 10: Transaction validation I (stateless checks and signatures)

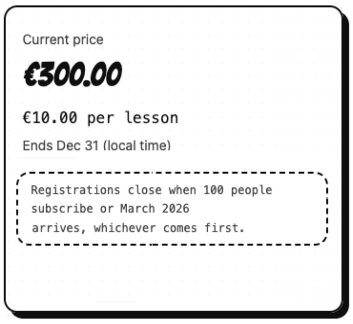
You implement the first half of the node's brain: checks that require no database reads. You learn how to reject garbage cheaply, verify signatures correctly, and why every meaningful field must be covered by the signed message to prevent tampering.



Lesson 11: Transaction validation II (stateful checks and applying transactions)



You add context: nonce rules, balance rules, and the actual state transition that updates accounts. This lesson turns transactions into real money movement and shows how replay attempts and double spends fail when the node enforces stateful validation consistently.



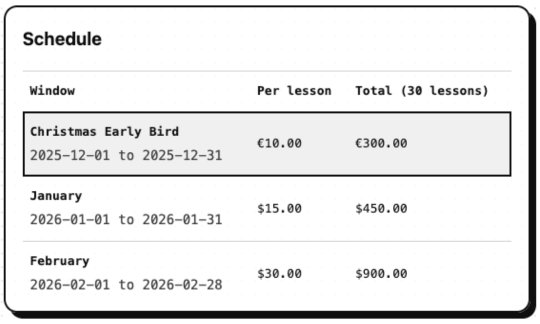
Current price

€300.00

€10.00 per lesson

Ends Dec 31 (local time)

Registrations close when 100 people subscribe or March 2026 arrives, whichever comes first.

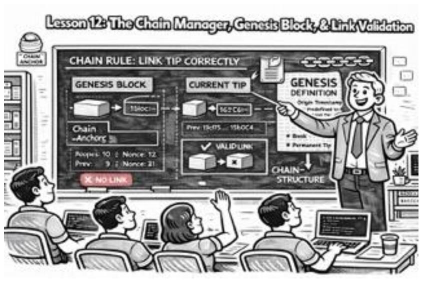


Schedule

Window	Per lesson	Total (30 lessons)
Christmas Early Bird 2025-12-01 to 2025-12-31	€10.00	€300.00
January 2026-01-01 to 2026-01-31	$15.00	$450.00
February 2026-02-01 to 2026-02-28	$30.00	$900.00

Lesson 12: The chain manager, genesis block, and link validation

You build the component that enforces "blockchain" as more than a list: a structure with rules. You define the genesis anchor, then implement the first chain rule: blocks must link to the current tip correctly, which prevents random insertion and broken histories.

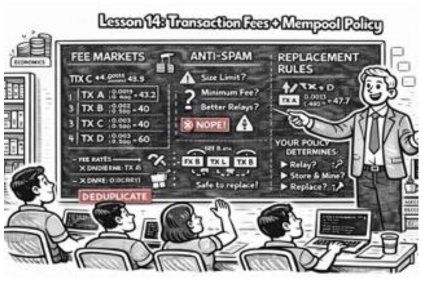

Lesson 12: The Chain Manager, Genesis Block, & Link Validation
CHAIN RULE: LINK TIP CORRECTLY
GENESIS BLOCK
CURRENT TIP
GENESIS DEFINITION
VALID LINK
NO LINK
CHAIN STRUCTURE

Lesson 13: The Mempool (the waiting room)

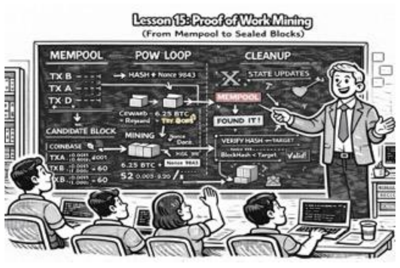
You build the staging area between validation and mining. This lesson explains why mempools are local and imperfect, why deduplication matters, how miners select transactions under constraints, and why you must clean the mempool when new blocks arrive.

Lesson 14: Transaction fees + mempool policy

You define what your node will and will not relay, store, and mine. This lesson introduces fee markets (fee rate), anti-spam rules (size limits, minimum fees), and replacement rules (how higher-fee transactions can replace earlier ones safely), which is where economics meets network survival.



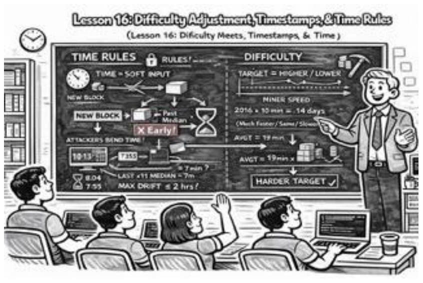
Lesson 15: Proof of Work mining (from mempool to sealed blocks)



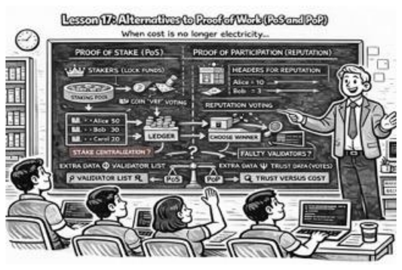
Your project becomes a real blockchain: a system that makes writing history expensive and verifying it cheap. You build candidate blocks, coinbase rewards, the PoW loop, and block verification, then connect mining to state updates and mempool cleanup.

Lesson 16: Difficulty adjustment, timestamps, and time rules

You learn why “time” is a soft input that attackers try to bend. This lesson defines what timestamps are allowed to do, how nodes reject unreasonable time claims, and how difficulty adjusts so block production stays stable even as hardware and conditions change.



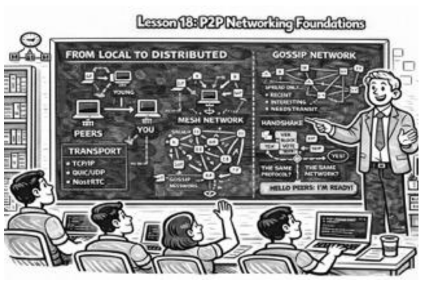
Lesson 17: Alternatives to Proof of Work (PoS and PoP)



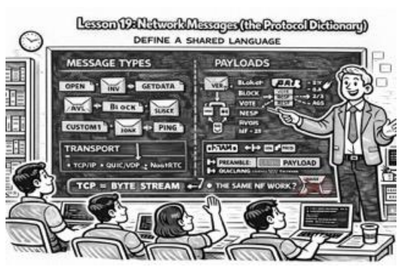
You explore the design space: what changes when “cost” is no longer electricity. You learn the high-level mechanics of PoS and participation-based models, what extra data structures they require, and why syncing and legitimacy proofs become more complex when consensus depends on stake or reputation.

Lesson 18: P2P networking foundations

You turn your node from a local program into a distributed system. This lesson introduces peers, transport choices, why gossip beats full mesh, and why every connection must start with a handshake that proves you are speaking the same protocol on the same network.



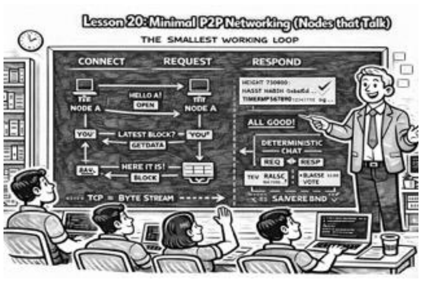
Lesson 19: Network Messages (the Protocol Dictionary)



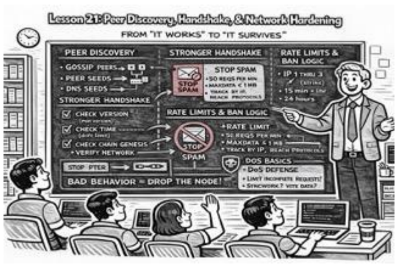
You define a shared language: message types and payloads. This lesson teaches message envelopes and the critical networking reality that TCP is a byte stream, so you need framing rules to avoid random parse failures and subtle bugs.

Lesson 20: Minimal P2P networking (nodes that talk)

You make nodes actually communicate: connect, request the latest block, and respond deterministically. This lesson is the smallest working network loop that proves your protocol and concurrency model are correct before you add heavy features like syncing and fork handling.



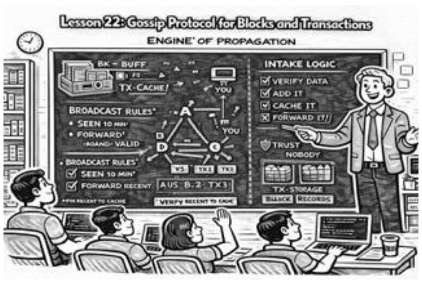
Lesson 21: Peer discovery, handshake, and network hardening (rate limits, bans, DoS basics)



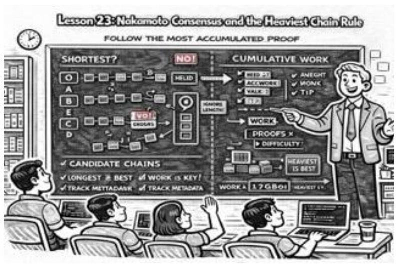
You move from "it works" to "it survives". This lesson introduces peer discovery strategies, stronger handshake checks, rate limits, ban logic, and basic defenses against spam and denial-of-service behaviors that would otherwise crash your node.

Lesson 22: Gossip protocol for blocks and transactions

You implement the engine of propagation: validated data spreads neighbor-to-neighbor. This lesson adds broadcast rules, seen-caches to prevent infinite rebroadcast storms, and the core intake logic: verify, store, forward, and never trust the messenger.

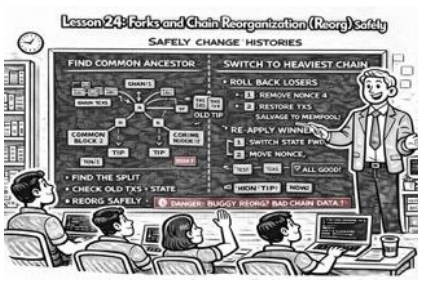


Lesson 23: Nakamoto Consensus and the heaviest chain rule



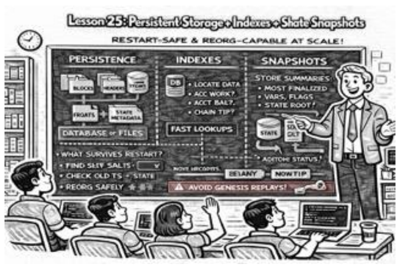
You implement the rule that makes decentralized networks converge: follow the valid chain with the most accumulated proof. This lesson teaches why "longest" is a simplification, why cumulative work matters, and what metadata you must track to compare competing histories efficiently.

Lesson 24: Forks and chain reorganization (reorg) safely



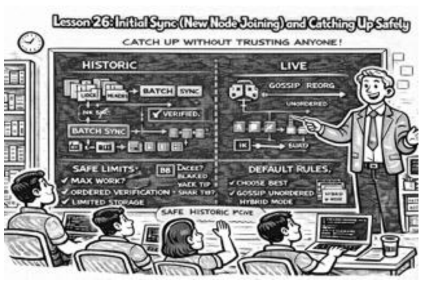
You implement the dangerous part: switching histories without corrupting state. This lesson teaches how to find the common ancestor, roll back losing blocks, re-apply winning blocks, salvage transactions back to the mempool, and why a buggy reorg can silently destroy balances and nonces.

Lesson 25: Persistent storage + indexes + state snapshots



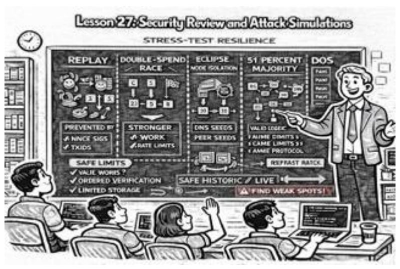
You make your node restart-safe and reorg-capable at scale. This lesson introduces what to persist (blocks, headers, metadata, state), how indexes make lookups fast, and why snapshots let you avoid replaying from genesis every time while still preserving verifiability.

Lesson 26: Initial sync (new node joining) and catching up safely



You solve the new node problem: how to join a network that is far ahead without trusting anyone. This lesson defines batch syncing, ordered verification, safe limits, and the separation between historical sync (ordered) and live gossip (unordered).

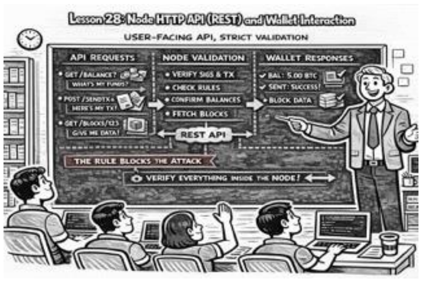
Lesson 27: Security review and attack simulations



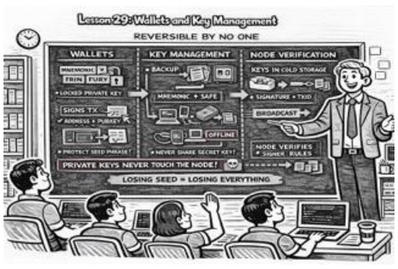
You stress-test your mental model by simulating real attacks: replay, double-spend races, eclipse isolation, 51 percent power, and denial-of-service patterns. The goal is not paranoia, it is clarity: you learn exactly which rule blocks which attack, and where your design is still weak.

Lesson 28: Node HTTP API (REST) and wallet interaction

You add the user-facing door to your node. This lesson builds a small REST API so wallets can query balances, submit signed transactions, and fetch blocks, while keeping all validation rules inside the node so the API never becomes a security bypass.



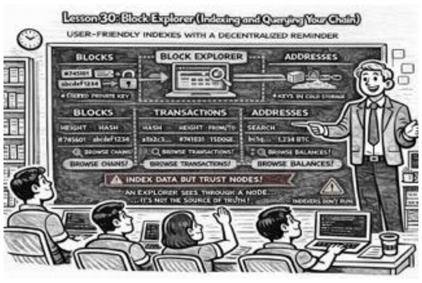
Lesson 29: Wallets and key management



You build the user's key system: keys, addresses, mnemonics, and offline signing. This lesson teaches the real security boundary: private keys never touch the node, wallets create proof, nodes verify proof, and losing your seed phrase means losing control permanently.

Lesson 30: Block explorer (indexing and querying your chain)

You build a read-only viewer that makes your blockchain human-browsable. This lesson adds indexing and query logic so you can navigate blocks, transactions, and addresses, and it teaches the key decentralization reminder: an explorer is a window into a node, not a source of truth by itself.


Lesson 30: Block Explorer (Indexing and Querying Your Chain)
USER-FRIENDLY INDEXES WITH A DECENTRALIZED REMINDER
BLOCKS
BLOCK EXPLORER
ADDRESSES
KEYS IN COLD STORAGE
BLOCKS
HEIGHT HASH
#745601 abcdef1234
BROWSE CHAINS!
TRANSACTIONS
BROWSE TRANSACTIONS!
ADDRESSES
SEARCH
1.234 BTC
BROWSE BALANCES!
INDEX DATA BUT TRUST NODES!
AN EXPLORER SEES THROUGH A NODE...
...IT'S NOT THE SOURCE OF TRUTH!

Why this moment matters

When the spotlight was on blockchain, it attracted hype, noise, and scams. Now the spotlight is on AI, and that is exactly why this is the moment for serious blockchain and decentralization work to happen quietly and correctly. Forget cryptocurrencies. Blockchain is about coordination, verification, resilience, and trustless infrastructure, things every industry needs, including AI itself. Models, data, provenance, incentives, and governance all break without decentralization. This is your window to stop chasing trends and start architecting the systems that will actually matter for the next decade.

Roberto Capodieci