
Our Project Overview

Our Goal:

Build a decentralized platform where universities and certified institutions can issue verifiable credentials (NFT-style) to students, and employers can verify them through blockchain — removing fake degrees or certificates.

Our Tech Base:

- **Language:** Rust
 - **Blockchain:** Stellar Soroban
 - **Data:** IPFS for off-chain metadata
 - **Frontend:** React (optional by Day 20+)
 - **Credential Format:** Soulbound NFT (non-transferable credential)
-

Our 25-Day Deep Roadmap

Phase 1 – Foundations (Days 1–4)

Build understanding and set up the environment

- **Day 1 – Blockchain & Stellar Fundamentals**

- Learn the basics of blockchain, smart contracts, and consensus.
 - Study how Stellar works: accounts, ledgers, and transactions.
 - Understand Soroban architecture and why Rust is used.

- **Day 2 – Development Environment Setup**

- Install all necessary tools: Rust, Cargo, Stellar CLI, Soroban SDK, IPFS desktop node.
 - Set up test accounts and configure CLI access to the testnet.
 - Create a Git repository and folder structure for the project.
 - Verify we can compile and deploy a “hello world” Soroban contract (conceptually check only).

- **Day 3 – Smart Contract Design Thinking**

- Study common credential verification systems.
 - Design entities: Issuer, Credential, User (student), Employer.
 - Define data attributes for each.
 - Decide what goes on-chain vs off-chain.

- **On-chain:** credential hash, issuer ID, owner ID.

- **Off-chain:** actual degree file / certificate on IPFS.

- Sketch a data-flow diagram: university → blockchain → employer.

- **Day 4 – Identity & NFT Model Research**

- Understand Decentralized Identity (DID) concepts.

- Learn what soulbound NFTs mean (non-transferable tokens).
 - Define how these NFTs represent a certificate.
 - Plan DID → wallet → credential linkage.
-

💡 Phase 2 – Smart Contract Core (Days 5–11)

Design, implement, and deploy the on-chain logic

- **Day 5 – Contract Data Model & API Definition**

- Define on-chain structures: issuers, credentials, ownership.
- List every function and parameter (register issuer, mint credential, revoke credential, verify).
- Draft the API and storage model documentation.

- **Day 6 – Access Control Rules**

- Design authorization: who can issue, who can view, who can revoke.
- Plan admin privileges and issuer registration workflow.
- Document all access restrictions.

- **Day 7 – Token Lifecycle & Storage Layout**

- Define credential lifecycle: issuance → ownership → verification → revocation.
- Finalize how to assign unique IDs.
- Plan IPFS integration for metadata CIDs.
- Describe how to mark credentials as soulbound (non-transferable).

- **Day 8 – Read & Verify Functionality**

- Design functions to read credentials and verify authenticity.
- Plan event logging (issuer registered, token minted, revoked).
- Specify output data for verification requests.

- **Day 9 – Unit Testing Strategy**

- Write a structured test plan:
 - Positive and negative test cases for registration, issuance, and revocation.
 - Verify immutability and non-transferability.
- Create a checklist of all edge cases to test.

- **Day 10 – Security Review**

- Review contract logic for vulnerabilities:
 - Unauthorized access, reentrancy, overflow, replay attacks.
- Draft mitigations for each.
- Finalize input validation rules.

- **Day 11 – Deployment & Smoke Testing**

- Deploy to Stellar testnet.
 - Register test issuer and issue a mock credential.
 - Verify and revoke it to confirm expected behavior.
 - Document transaction hashes and outcomes.
-

📍 Phase 3 – Off-Chain Integration (Days 12–17)

Build backend & IPFS layers that talk to the blockchain

- **Day 12 – Backend Architecture Planning**

- Choose backend environment (Rust Axum or NodeJS).
- Define backend tasks:
 - Connect to blockchain.
 - Manage off-chain metadata.
 - Handle authentication and signature requests.
- Design backend database schema for caching and analytics.

- **Day 13 – IPFS Integration**

- Configure IPFS node.
- Design metadata structure:
 - Credential details (degree name, issuer, date, hash).
- Store JSON metadata and the certificate file on IPFS.
- Create logic to pin and retrieve files.
- Decide how to link IPFS CIDs with blockchain token IDs.

- **Day 14 – DID & Wallet Management**

- Implement user identity handling.
- Define how wallets link to DIDs.
- Map DIDs → issuer/student/employer profiles in backend.
- Add authentication flow for signing blockchain transactions.

- **Day 15 – Employer Verification Logic**

- Build backend endpoint that allows an employer to input a credential ID or wallet address and check validity.
- Ensure it fetches data both from blockchain and IPFS.
- Define API responses for “valid,” “revoked,” and “not found.”

- **Day 16 – Credential Revocation & Update Process**

- Design how issuers can revoke credentials (mark invalid).
- Add timestamping and reasons for revocation (stored off-chain).

- Plan notification or event system for affected users.
 - **Day 17 – Testing Backend + Blockchain Sync**
 - Test full cycle:
 - Issuer registers.
 - Credential minted → metadata uploaded → IPFS CID stored.
 - Employer queries credential and gets correct status.
 - Record all successful flows.
-

📍 Phase 4 – Frontend & UX Layer (Days 18–22)

Create user interfaces for issuers, students, and employers

- **Day 18 – UI Wireframes & Flow Design**
 - Create rough sketches for three roles:
 - **Issuer Dashboard** – Issue/revoke credentials.
 - **Student Profile** – View owned credentials.
 - **Employer Search** – Verify credentials.
 - Define user journeys and input forms.
- **Day 19 – Frontend Setup**
 - Initialize frontend framework (React or Svelte).
 - Plan UI state management.
 - Connect test backend endpoints conceptually (mock data).
 - Add basic navigation and layouts.
- **Day 20 – Blockchain Integration on UI**
 - Connect wallet interaction layer.
 - Trigger blockchain reads/writes from the UI via backend.
 - Display verification results (valid / revoked).
 - Implement QR or search-by-wallet-address option.
- **Day 21 – Credential Visualization**
 - Design certificate cards showing NFT image + metadata.
 - Include IPFS link and verification hash.
 - Make each credential clickable for detailed info.
- **Day 22 – Polishing & Usability Testing**
 - Run a few end-to-end tests:
 - University issues certificate.
 - Student sees NFT credential.

- Employer verifies instantly.
 - Note UI/UX issues and refine flows.
-

📍 Phase 5 – Finalization & Presentation (Days 23–25)

Testing, optimization, documentation, and pitch preparation

- **Day 23 – Performance & Security Testing**
 - Perform stress testing on minting and verification.
 - Check contract gas usage and optimize storage fields.
 - Test revocation multiple times for consistency.
 - Verify that no credential is transferable.
- **Day 24 – Documentation & Presentation Material**
 - Write project documentation:
 - System architecture diagram.
 - Smart contract API doc.
 - Data flow: issuer → blockchain → employer.
 - Key features and security measures.
 - Create presentation slides:
 - Problem, solution, features, roadmap, demo screenshots.
 - Add “Decentralized Identity” and “Soulbound NFT” visuals.
- **Day 25 – Demo, Review & Pitch**
 - Record or perform a live demonstration:
 - Register university.
 - Issue a credential.
 - Verify from employer side.
 - Show on-chain proof (transaction hash).
 - Summarize learnings and next steps:
 - Scalability.
 - Integration with government or global verification networks.

Our Deliverables Summary

Phase	Output
1	Setup, architecture docs, and system design diagrams
2	Functional smart contract deployed to testnet
3	Backend service + IPFS integration working
4	Functional UI and verification workflow
5	Complete MVP, presentation deck, and demo video

Our End Result

By Day 25, **we** will have a working MVP where:

- Universities issue blockchain-backed NFT credentials.
- Students control them in their wallets (non-transferable).
- Employers can instantly verify authenticity through **our** app.
- All data is tamper-proof, transparent, and decentralized.