Genuinegrads - Blockchain-Based Certificate Verificate System with Compressed NFTs and Zero- Knowledge Proofs

M.H.M.Shahadh

E2240185

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka in partial fulfillment of the requirements of the Degree of Bachelor of Information Technology(External) in Information Technology

01/2026

DECLARATION

We declare that this report is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of Student: M.H.M. Shahadh

Signature of Student: 

Date: 2026/01/05

Supervised by

Name of Supervisor: Mr. Chathuranga Chandrasekara

Signature of Supervisor: 

Date: 2026/01/05

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project supervisor for the valuable insights, continuous guidance, and constructive feedback provided throughout the development of this project. Their support and direction played a significant role in shaping the progress and quality of the final outcome.

I also extend my appreciation to the university lecturers and instructors who contributed to my academic and technical growth during my degree programme. The knowledge and foundation they provided were essential in completing this project successfully.

In addition, I would like to convey special thanks to Ackee Blockchain's School of Solana for the valuable lessons and practical learning resources, and to Turbine for providing professional training in Solana development. The guidance and hands-on exposure gained through these programmes significantly strengthened my understanding and skills in Solana development and supported the implementation of this project.

Finally, I am grateful to everyone who supported and encouraged me throughout this journey.

ABSTRACT

Academic credential fraud, including forged certificates and unverifiable qualifications, poses a persistent challenge to higher education institutions, employers, and regulatory bodies, particularly in large-scale and cross-institutional verification contexts. GenuineGrads addresses this problem by designing and implementing a secure, scalable, and privacy-aware academic credential issuance and verification platform using blockchain technology. The system enables universities to issue tamper-resistant academic certificates as compressed non-fungible tokens (cNFTs) on the Solana blockchain, significantly reducing issuance costs while preserving independent verifiability. A multi-tenant architecture with strict data isolation supports multiple universities on a shared platform, while public verification is enabled through certificate identifiers and QR codes without requiring institutional intervention. To address privacy concerns inherent in traditional verification workflows, the platform integrates Zero-Knowledge Proofs (ZKPs) to allow students to selectively prove academic claims, such as achievement or eligibility thresholds, without revealing full academic records. The solution incorporates robust certificate lifecycle management, including bulk issuance, revocation with audit trails, and verifier-friendly status checks. Evaluation results demonstrate that GenuineGrads achieves cost-effective issuance, reliable verification, and practical usability within real-world operational constraints, highlighting its suitability as a trust infrastructure for modern digital academic credentials.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ALT	– Address Lookup Table
API	– Application Programming Interface
BBS	– Boneh–Boyen–Shacham (Signature Scheme)
cNFT	– Compressed Non-Fungible Token
CPI	– Cross-Program Invocation
CSV	– Comma-Separated Values
DAS	– Digital Asset Standard
DB	– Database
DID	– Decentralized Identifier
E2E	– End-to-End
ERD	– Entity Relationship Diagram
GPA	– Grade Point Average
IMM	– International Marketing Management
IPFS	– InterPlanetary File System
JWT	– JSON Web Token
MPL	– Metaplex
NFT	– Non-Fungible Token
NIC	– National Identity Card
PDA	– Program Derived Address
PKI	– Public Key Infrastructure
RPC	– Remote Procedure Call
SRS	– Software Requirements Specification
SSE	– Server-Sent Events
TOTP	– Time-based One-Time Password
UI	– User Interface
UNESCO	– United Nations Educational, Scientific and Cultural Organization
VC	– Verifiable Credential

W3C – World Wide Web Consortium

ZKP – Zero-Knowledge Proof

CHAPTER 1: INTRODUCTION

1.1 CHAPTER INTRODUCTION

Academic credential fraud including degree mills/diploma mills, forged transcripts, and fraudulent qualifications has become a persistent global challenge that affects workforce quality, public trust, and safety-critical sectors. UNESCO's IIEP highlights degree/diploma mills as "dubious providers" offering bogus qualifications, and describes credential abuse as an ongoing threat to higher education systems and employers [5].

Secure digital credentialing strategies are gaining popularity as governments and organizations accelerate digital transformation to enhance service delivery and lower administrative burden. Strengthening digital public infrastructure and enhancing digital service delivery are key components of Sri Lanka's national digital plan [14].

This project, GenuineGrads, addresses these challenges by delivering a blockchain-based academic credential issuance and verification platform that supports multi-university operation, scalable issuance, and privacy-preserving verification of selected claims.

1.2 BACKGROUND AND CONTEXT

Traditional credential verification workflows typically depend on PDF certificates, manual registrar checks, email-based confirmation, or third-party verification intermediaries. These approaches introduce several systemic weaknesses: documents can be forged or altered, verification is slow and does not scale efficiently for high volumes, and verifiers often require excessive personal information to confirm a simple claim (e.g., "has a degree" or "GPA $\geq$ X"). This aligns with broader observations in credential-fraud and integrity literature that emphasize the operational burden and risk of weak verification practices[2].

GenuineGrads is positioned as a trust infrastructure for academic credentials: universities issue verifiable digital certificates; students hold and share those credentials; and employers/verifiers validate authenticity quickly using a public verification portal. The interim work establishes the

project context and target region (South Asia), and frames the need for low-friction, scalable verification with stronger integrity guarantees. (Interim Report, 2025)

1.3 PROBLEM STATEMENT

Current academic certificate issuance and verification processes are often centralized, time-consuming, and difficult to scale, while providing limited support for privacy-preserving disclosure. Verifiers may rely on documents that are easy to falsify or must conduct slow manual checks. Students may be required to disclose more information than necessary to validate a limited claim (e.g., confirming degree completion or meeting a GPA threshold). Additionally, institutions lack an affordable, universally verifiable mechanism for bulk issuance and revocation that can be validated independently by third parties. (Interim Report, 2025)

1.4 AIM

The aim of GenuineGrads is to design and implement a secure, scalable, and privacy-aware academic credential system that enables universities to issue tamper-resistant credentials and enables employers/verifiers to validate authenticity efficiently, while allowing selective disclosure of sensitive academic claims when required. (Interim Report, 2025)

1.5 OBJECTIVES

Based on the finalized project direction, the key objectives are:

1. Implement role-based portals for issuer (university), holder (student), and verifier (employer/public). (Interim Report, 2025; SRS, 2025)

2. Support scalable certificate issuance using compressed NFTs (cNFTs) and an auditable credential lifecycle [20].

3. Provide an integrity-preserving revocation approach aligned to the chosen credential lifecycle design. (Interim Report, 2025)

4. Enable verification through a unified digital asset retrieval mechanism suitable for compressed assets (e.g., DAS-style APIs) [7].

5. Incorporate Zero-Knowledge Proofs (ZKPs) to support selective disclosure of claim predicates (e.g., GPA ≥ X) without revealing full academic records [21].

6. Ensure the system is usable in practice through clear workflows, traceability, and performance-aware design for issuance and verification. (Interim Report, 2025; SRS, 2025)

1.6 SCOPE OF THE PROJECT

The scope of GenuineGrads is defined by its three primary portals and its supporting credential infrastructure:

- University Portal: institutional onboarding, student enrollment (single/bulk), certificate template management, certificate issuance, and revocation. (SRS, 2025)
- Student Dashboard: viewing issued certificates and generating ZKP proofs for eligible claims. (SRS, 2025)
- Employer/Verifier Portal: validating certificate authenticity and verifying ZKP-based claims when provided. (SRS, 2025)

From a technology boundary perspective, the system includes:

- Compressed NFT issuance model using Bubblegum v2 as the program interface for creating and interacting with cNFTs on Solana [20].
- State compression / Merkle-tree anchored credential state, consistent with Solana's state compression model for compressed NFTs [6].
- Digital Asset Standard (DAS)-style retrieval for querying asset data and supporting efficient verification experiences [7].
- ZKP proof generation and verification pipeline using Circom/snarkjs-style flows for proving/verification steps [21].

Out of scope for this implementation (to keep delivery feasible within the final-year timeline) includes full national-level interoperability infrastructure, legally binding e-signature integration, muti-sig, and cross-country credential federation beyond the multi-university platform model. (SRS, 2025)

1.7 METHOD AND APPROACH OVERVIEW

GenuineGrads adopts a system architecture where:

- Universities act as trusted issuers, minting verifiable digital credentials;
- Students are credential holders, controlling sharing and optionally generating privacy-preserving proofs;
- Employers/verifiers validate credentials through certificate identifiers/QR and asset verification interfaces. (Interim Report, 2025; SRS, 2025)

The approach uses blockchain anchoring to provide immutability and third-party verifiability, while ZKPs are introduced to address privacy requirements that cannot be solved by public metadata alone.

1.8 CHAPTER SUMMARY

This chapter presented the motivation for GenuineGrads by highlighting the growing problem of credential forgery and emphasizing the necessity for scalable, privacy-preserving verification methods. It then stated the problem, defined the project aim, listed core objectives, and clarified the project scope and approach. The next chapter reviews existing credential verification systems and related literature to position the contribution of GenuineGrads.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

2.1 CHAPTER INTRODUCTION

This chapter reviews the academic credential fraud problem and why verification is difficult at scale, major digital credentialing paradigms used in real deployments, and blockchain-based credential systems and standards relevant to GenuineGrads. It concludes with a gap analysis that motivates the selected design direction: scalable on-chain anchoring for authenticity and verifiable status, combined with privacy-preserving verification for sensitive claims.

2.2 ACADEMIC CREDENTIAL FRAUD AND VERIFICATION CHALLENGES

Academic credential abuse includes diploma/degree mills, forged certificates, counterfeit transcripts, and misuse of accreditation terminology, all of which undermine trust in higher education and recruitment processes. UNESCO's IIEP–ETICO highlights ongoing "credential abuse" trends and the operational risks posed by diploma and accreditation mills [10].

International bodies such as CHEA/UNESCO have also documented institutional and government-level actions needed to discourage degree mills, indicating that the problem is not isolated to any single region [4].

Beyond authenticity, modern verification introduces two practical challenges:

1. Scalability: Manual registrar checks and email-based verification do not scale efficiently for large volumes (mass recruitment, migration, scholarship screening, etc.).

2. Privacy: Verifiers frequently request more information than necessary to confirm limited predicates (e.g., "has a degree" or "meets minimum threshold"), creating unnecessary exposure of personal academic data.

These two pressures high-volume verification and privacy-preserving disclosure are key drivers behind modern digital credential initiatives.

2.3 DIGITAL CREDENTIAL PARADIGMS

Real-world credential systems typically fall into three broad paradigms.

2.3.1 CENTRALIZED DIGITAL REPOSITORIES (GOVERNMENT / PLATFORM-HOSTED)

Centralized repositories store academic awards in a government- or platform-managed database and provide "pull" and "share" mechanisms for users. India's National Academic Depository (NAD) is an example of a 24×7 online depository where academic institutions store and publish digital academic awards, with retrieval and authenticity guarantees supported by the platform [15].

NAD is integrated through DigiLocker's NAD portal, which describes the ecosystem for institutions and student access workflows [15].

Strengths:

- Simple user experience.
- Fast verification.
- Centralized governance.

Limitations:

- Creates a strong dependency on a central operator.
- Cross-border verification and independent third-party validation depend on the platform's availability and policies.

2.3.2 PKI / DIGITALLY SIGNED CERTIFICATES (NON-BLOCKCHAIN AUTHENTICITY)

Some systems issue digitally signed certificates where authenticity and tamper detection rely on established digital signature infrastructure rather than blockchain anchoring. University Malaya describes e-Scroll as a digitally signed certificate designed to authenticate signer identity and ensure certificate content is not altered [25].

Strengths:

- Mature cryptographic foundations.
- Clear integrity checks.

Limitations:

- Verification often depends on trusted distribution channels, certificate chains, or institutional portals.
- Revocation and global trust portability can be operationally complex.

2.3.3 HOLDER-CENTRIC VERIFIABLE CREDENTIALS (STANDARDS-BASED, WALLET-ORIENTED)

A widely adopted standardization approach is the W3C Verifiable Credentials (VC) model, which defines an issuer–holder–verifier ecosystem and an extensible data model for expressing credentials that can be protected from tampering. The VC ecosystem is designed to support portable credentials that can be stored in wallets and presented by holders to verifiers, enabling selective sharing depending on cryptographic mechanism and implementation choices [26].

2.4 BLOCKCHAIN-BASED CREDENTIAL SYSTEMS AND DEPLOYMENTS

Blockchain-based systems commonly use one of two approaches: anchoring document hashes or credential commitments, or issuing structured credentials (e.g., VCs) with blockchain trust layers for status and issuer authorization.

2.4.1 BLOCKCERTS (OPEN STANDARD BLOCKCHAIN CREDENTIALS)

Blockcerts positions itself as an open standard for creating, issuing, viewing, and verifying blockchain-registered certificates, emphasizing cryptographic signing and tamper resistance [3].

MIT's digital diploma initiative describes the use of Blockcerts tooling to issue and verify credentials using blockchain technology, illustrating early university-level adoption [9].

Key benefits:

- Verifiable authenticity
- Tamper-evident documents
- Independent verification.

Common limitation for many document-style approaches: Privacy and selective disclosure are not inherently supported unless additional cryptographic layers are added.

2.4.2 OpenCerts / OpenAttestation (Singapore Government Ecosystem)

Singapore's credential verification ecosystem includes OpenCerts for verification of issued certificates [18]. GovTech documentation presents OpenAttestation as an open-source framework for endorsement and verification of documents using blockchain, with authenticity and integrity assurances [17], [16]. SkillsFuture Singapore describes OpenCerts as a GovTech platform used to issue tamper-proof digital certificates and verify authenticity [23]. A university example (NTU) states that OpenCerts enables issuance and verification of tamper-proof digital degree certificates

and transcripts. A university example (NTU) states that OpenCerts enables issuance and verification of tamper-proof digital degree certificates and transcripts [19].

Strengths:

- Strong government support
- Clear verification UX
- Practical national deployment.

Limitations (general): document-format systems still face selective disclosure constraints unless extended with privacy-preserving credential schemes.

2.4.3 EBSI AND CROSS-BORDER VERIFIABLE CREDENTIALS (EU)

The European Blockchain Services Infrastructure (EBSI) promotes a verifiable credentials framework based on W3C VCs, wallets, and blockchain-backed trust infrastructure, emphasizing user control and cross-border verification [11]. EBSI reports a multi-university cross-border pilot involving multiple university alliances and institutions, demonstrating practical experimentation at EU scale [27]. EBSI issuance guidance describes wallet-based VC issuance initiation patterns (QR codes/redirects), highlighting operational realities of wallet interactions [12].

Strengths:

- Standards-first approach (W3C VC/DID).
- Strong interoperability intent.
- Cross-border pilots.

Limitations: Operational complexity (wallet ecosystem maturity, conformance, governance alignment) can be non-trivial for smaller institutions.

2.4.4 PERMISSIONED VS PERMISSIONLESS CREDENTIAL LEDGERS

A core design debate in blockchain credentialing is whether to use permissioned networks (consortium-controlled) or permissionless public networks. Castro-Iragorri and Giraldo analyze academic certification adoption across both models and discuss trade-offs relevant to scaling and governance [1].

This debate informs GenuineGrads because public chains improve independent verifiability, but governance and cost characteristics must support real-world issuance volumes and revocation/status needs.

2.5 PRIVACY-PRESERVING VERIFICATION AND SELECTIVE DISCLOSURE

Many verification scenarios do not require full transcript disclosure. Modern digital credential research emphasizes selective disclosure and predicate proofs, allowing holders to prove statements such as "GPA ≥ X" or "belongs to set Y" without revealing the full underlying record.

The W3C VC 2.0 family describes selective disclosure and derived predicates as a capability when supported by the credential/proof mechanism [26]. A 2024 review on selective disclosure in digital credentials highlights ZKP-based approaches as a method to prove statements while minimizing disclosed attributes [22]. W3C's Data Integrity BBS cryptosuite specification explicitly targets selective disclosure and unlinkable proofs within the issuer–holder–verifier model [26].

Implication for GenuineGrads: a certificate verification portal alone is insufficient for privacy-sensitive claims. A practical system must support a mechanism (ZKP/selective disclosure proofs) that reduces oversharing while remaining verifiable by third parties.

2.6 SYNTHESIS AND GAP ANALYSIS

From the reviewed systems and standards, several gaps emerge that directly motivate the GenuineGrads design:

1. Selective disclosure is not universal in deployed document-style systems. Systems that verify documents (even when blockchain-anchored) often validate authenticity but do not natively support predicate proofs over sensitive academic attributes[3], [17], [22].
2. Scalability and cost must support bulk issuance. Real deployments require institution-friendly issuance at scale. Comparative studies on blockchain for certificates emphasize benefits but also underline operational and organizational considerations [1], [24].
3. Multi-institution support with strong data isolation is operationally important. National platforms (e.g., NAD) centralize storage and verification workflows, but multi-university platforms must ensure governance boundaries and reduce cross-tenant exposure [15].
4. Revocation/status verification must be easily checkable by verifiers. Credential systems must provide a verifier-friendly method to determine whether a credential remains valid, without requiring manual institutional intervention. (This is handled in GenuineGrads via the certificate lifecycle and verification flow described later in Chapter 4, consistent with the project design.)

GenuineGrads combines scalable, independently verifiable certificate authenticity and status through blockchain anchoring and standardized retrieval patterns, with ZKP-enabled selective disclosure for sensitive academic claims. This aligns with the issuer–holder–verifier ecosystem in W3C VC models while targeting the specific operational realities of multi-university issuance and public verification.

2.7 CHAPTER SUMMARY

This chapter reviewed the credential fraud context and surveyed major credential verification paradigms, including centralized repositories (e.g., NAD), digitally signed certificates (e.g., e-Scroll), blockchain-anchored document frameworks (Blockcerts, OpenAttestation/OpenCerts), and standards-driven VC ecosystems (W3C VC, EBSI). The synthesis highlighted key gaps especially selective disclosure and scalable operational verification which motivate GenuineGrads' combined approach: scalable verifiable credentials with optional ZKP-based privacy-preserving claim verification. The next chapter presents the system requirements and specification baseline used to implement the platform.

CHAPTER 3: SPECIFICATION AND DESIGN APPROACH

3.1 CHAPTER INTRODUCTION

The objective of this chapter is to define what the GenuineGrads system is required to do (specification) and to present the top-level design approach describing how the final system satisfies those requirements. To describe the system clearly, the specification and design are presented using multiple viewpoints (business/user viewpoint, user interaction viewpoint, dynamic behavior viewpoint, and data viewpoint). Fine implementation details and code-level descriptions are intentionally excluded and are presented later in the Implementation chapter.

Note on evolution: The system design evolved from the initial SRS as implementation progressed. This report documents the final implemented design, while explicitly justifying the most important deviations from the SRS to avoid confusion and to demonstrate design decision-making.

3.2 SYSTEM OVERVIEW

GenuineGrads is a multi-stakeholder academic credential platform that supports:

- University onboarding and student management
- Certificate issuance (single and bulk) as verifiable digital credentials
- Public verification using certificate identifiers/QR
- Credential lifecycle management including revocation and status checks
- Optional privacy-preserving verification of selected student achievement using Zero-Knowledge Proofs (ZKPs)

The platform serves four primary actor groups:

1. Super Admin (Platform Administrator)
2. University Admin (Issuer)
3. Student (Credential Holder)
4. Verifier/Employer (Credential Verifier)

3.3 BUSINESS AND USER REQUIREMENTS

3.3.1 BUSINESS REQUIREMENTS

From an operational perspective, the system must enable universities to issue credentials in a way that is:

- Trusted (tamper-resistant and independently verifiable)
- Scalable (supports bulk issuance flows)
- Auditable (issuance/revocation events traceable)
- User-friendly (verification possible without contacting the university)
- Privacy-aware (only required information disclosed to verifiers)

3.3.2 USER REQUIREMENTS (ACTOR-BASED)

This section summarizes the capability required per stakeholder. The detailed functional requirements are grouped here by user role to keep the report readable. Detailed use case diagrams are presented in Chapter 4, Figures 4 - 7 .

A. Super Admin Requirements

- ✓ Approve/Reject universities.
- ✓ Manage platform governance functions.

B. University Admin Requirements

- ✓ Create and manage university profile and settings
- ✓ Register students (single + bulk upload)
- ✓ Manage academic metadata needed for issuance
- ✓ Issue certificates (single + bulk issuance)
- ✓ Revoke issued certificates and maintain lifecycle status
- ✓ View analytics

C. Student Requirements

- ✓ Authenticate and access dashboard
- ✓ View issued certificates and achievements
- ✓ Share certificate for verification (ID/QR/share link concept)
- ✓ Generate ZKP proofs for eligible claims (when applicable)
- ✓ View verification logs / certificate status

D. Verifier/Employer Requirements

- ✓ Verify certificate authenticity using certificate ID/QR
- ✓ View issuer and certificate status (valid/revoked/invalid)
- ✓ Verify claim proofs (ZKP verification) when supplied by student
- ✓ Access verification without requiring an account (public portal)

3.4 DESIGN VIEWPOINTS USED TO DESCRIBE THE SYSTEM

To communicate the final system clearly, the design is described using multiple viewpoints. Each viewpoint highlights different aspects of the system and reduces ambiguity.

3.4.1 STATIC ARCHITECTURE VIEWPOINT

This viewpoint shows how the system is partitioned into layers/modules and how major components interact. The final layered architecture is shown in Chapter 4, Figure 1.

3.4.2 DATA VIEWPOINT

This viewpoint explains how data is structured and where it resides.

GenuineGrads uses a two-scope database model:

- Shared database for platform-level, cross-university data
- Private per-university database for institution-owned student and academic records

Detailed ER models are presented in Chapter 4, Figures 2 and 3.

3.4.3 DYNAMIC BEHAVIOR VIEWPOINT

This viewpoint explains how the system behaves over time for key workflows.

Key workflows covered:

- Student registration
- Certificate drafting and asset preparation
- Certificate minting (initiation → blockchain confirmation → finalization)
- Certificate verification
- Certificate revocation
- ZKP proof generation & verification

Dynamic behaviors are detailed in Chapter 4, Figures 8-17.

3.4.4 USER INTERFACE VIEWPOINT

The GenuineGrads system provides dedicated user interfaces for each stakeholder group, including the Super Admin, University Admin, Student, and Verifier portals. These interfaces were implemented to support role-specific workflows such as certificate issuance, certificate access, proof generation, and public verification.

User interface screenshots for each portal are provided in Appendix B.

3.5 CONSTRAINTS AND KEY DESIGN DRIVERS

The final design was guided by the following constraints and decision drivers:

1. Multi-university support with data isolation: Universities must operate independently while sharing a common platform. This drove the separation into shared vs private databases.

2. Scalable issuance including bulk operations: The system must support issuing many certificates efficiently (bulk issuance), which influenced minting workflow design and operational logging.

3. Verifier accessibility (public verification): Employers should verify without needing institutional access, driving a public verifier portal and a verification flow based on certificate ID/QR.

4. Privacy requirements for sensitive claims: Not all academic data should be publicly visible; ZKP capability was introduced to support selective disclosure where needed.

5. Auditability and lifecycle integrity (revocation/status): Certificates must have a lifecycle state that verifiers can trust (valid/revoked), requiring explicit revocation handling and status checks.

6. Implementation feasibility within project timeline: Some SRS features were refined or staged to ensure a complete end-to-end deliverable (issuance → verification → revocation) with usable portals.

3.6 JUSTIFICATION OF DEVIATIONS FROM THE SRS (UPDATED FINAL DIAGRAMS)

The finalized design diagrams differ from the SRS versions because the system architecture and flows were refined during implementation to better align with real operational constraints of Solana compressed NFTs (cNFTs), multi-tenant university handling, certificate lifecycle management (issuance, verification, revocation), and privacy-preserving ZKP usage. The SRS diagrams were intentionally high-level to capture initial requirements and feasibility. During implementation, several areas required practical redesign mainly to ensure reliability of mint confirmation, enforce revocation correctness, support bulk issuance, and integrate ZKP flows without exposing sensitive student data. The following deviations represent deliberate design evolution rather than inconsistency.

3.6.1 ARCHITECTURE DIAGRAM EVOLUTION

What changed:

- The SRS architecture presents the system as a simplified set of layers (Client → GraphQL API → Blockchain/Storage → External services).

- The finalized architecture refines this into clearer operational components: separate portals, clearer service boundaries (auth, certificate services, ZKP service, Solana interaction), and explicit separation between private university data stores and a shared central index that supports cross-university constraints.

Why this changed (implementation-driven):

- During implementation, it became necessary to make shared vs private responsibilities explicit. Some data must remain university-scoped (courses, enrollments, certificate templates), while other data must be shared globally (revocation index, global student mapping, mint logs) to support verification and prevent duplicates.

- The final architecture also clarifies where external Solana infrastructure is relied upon (Helius for RPC + NFT APIs/webhooks) and why these are not optional in a real deployment (for confirmation, indexing, and asset retrieval).

- The final architecture provides a more accurate mapping from implemented code modules to system responsibilities, which improves traceability in the final report.

3.6.2 DATA MODEL EVOLUTION - SHARED CENTRAL DATABASE

What changed:

- The SRS shared schema included only a minimal set of entities (University, Admin, GlobalStudentIndex, MintActivityLog, RevokedCertIndex).

- The finalized shared ERD expands these with operational fields required in a working system, including:

- Stronger university identity/registration handling (status/approval, blockchain identity fields, operational metadata)

- Richer mint logs (status/error handling, transaction signatures, metadata references)

- Admin-level operational features (security and audit readiness)

- Webhook and notification-related persistence (needed for reliability and auditability)

Why this changed:

- In the SRS, minting and revocation are shown as simple actions. In implementation, minting and revocation are multi-step, failure-prone distributed operations. To support robustness, the system needs persistent fields for:

  - Tracking transaction signatures and confirmations,

  - Capturing error messages and retry reasons,

  - Linking mint/burn events to university administrators for accountability,

  - Maintaining webhook logs or event traces for debugging and audit.

- Verification requires that revocation information be queryable centrally (not per-university), because public verifiers should not need privileged access to a university's private database to confirm revocation status.

3.6.3 DATA MODEL EVOLUTION - PRIVATE UNIVERSITY DATABASE

What changed:

- The SRS private schema captured a minimal academic structure (Course, Student, Enrollment, Achievement, Certificate, ZKPProofRequest).
- The final schema expands to reflect the implemented features and real usage needs, including:
- Certificate template/design persistence,
- Issuance job/batch tracking (for bulk issuance),
- Verification logs (for analytics and audit),
- Richer certificate blockchain linkage fields (addresses, signatures, metadata references),
- Separation of ZKP-related persistence into clearer responsibilities (commitments, proof records, and verification/audit trails) rather than a single generic proof request object.

Why this changed:

- During implementation it became clear that ZKP handling needs structured storage:
  - A commitment must exist even before a proof is generated.
  - Proof generation is not guaranteed (user may cancel), so “commitment created” and “proof generated” must be distinct states.
  - Verification events must be logged for analytics and integrity.

- Bulk issuance required job-level persistence to track progress, partial failures, and completion something not represented in the SRS diagrams but necessary in the implemented system.

- Certificate records require explicit blockchain references to make verification consistent and deterministic (mint address, metadata URI, issuance timestamps, revocation flags/reasons, and traceability to the issuer).

3.6.4 USE CASE DIAGRAM EVOLUTION (UNIVERSITY / STUDENT / EMPLOYER)

University

What changed:

- SRS shows core actions: register students, issue certificates, design certificates, view analytics, manage issued certificates.

- Final diagram improves correctness and clarity by explicitly reflecting what is implemented:

  - Separation of certificate issuance responsibilities (single vs bulk),

  - Clearer authentication entry points,

  - Explicit management actions that exist in the system (templates, issuance logs, revocation handling).

Why this changed:

- During UI + backend implementation, several actions became distinct modules rather than one combined “issue certificate” capability.

- This is a natural refinement: the SRS captured “what must exist”, while the final diagram captures “how the admin actually uses the portal”.

Student

What changed:

- SRS focuses on viewing achievements, generating proof, and sharing.
- Final version clarifies:
  - Wallet connection as the actual login mechanism,
  - Viewing certificates and achievements as separate flows,
  - ZKP proof generation as an optional extension when a claim is ZKP-supported,

Why this Changed:

- Implementation revealed the need to separate “viewing assets” from “generating proofs” because proof generation is conditional and not a default action for all students.

Employer / verifier

What changed:

- The SRS shows verification validity and optional ZKP claim verification.
- The final diagram reflects the implemented reality that verification is a two-stage process:
  I. Verify certificate authenticity + revocation status
  II. Optionally verify additional ZKP claims

Why this changed

- Separating “certificate validity” from “claim verification” is important so that employers can still verify authenticity even if no proof is provided or no ZKP claim exists.

3.6.5 SEQUENCE DIAGRAM EVOLUTION

University/admin sequence

What changed

- The SRS sequence combines many operations (registration, login, register students, template design, mint loop, revoke, analytics) into one long interaction.
- The finalized approach isolates the technically critical behaviour into dedicated sequences especially minting showing the real steps required to:
  - Prepare metadata,
  - Submit mint transactions,
  - Confirm minting,
  - Persist results and update logs.

Why this changed

- Implementation showed that minting is not a trivial synchronous step it requires confirmation, failure handling, and post-mint indexing steps.
- Splitting sequences reduces ambiguity and aligns the documentation with the actual system modules and endpoints.

Employer verification sequence

What changed

- SRS: “query Helius → fetch metadata → check revoked index → show valid/revoked; optionally verify ZKP”.
- Final: expands verification into more robust steps including:
  - Consistent revocation/index checks,
  - Deterministic retrieval of certificate asset/metadata,
  - Clear handling for invalid/missing assets,
  - Recording verification events (analytics/audit support).

Why

- Public verification needs to be resilient and auditable. Logging verification events also supports the analytics requirements promised for stakeholders.

Student proof/claim sequence

What changed

- The SRS presents proof generation as a backend-driven/off-chain process.
- The final sequence reflects stronger privacy and realistic ZKP handling by separating:
  - Commitment creation/storage,
  - Proof generation,

- Proof verification flow,
- Audit/logging.

Why this changed

- During implementation, ZKP flows required clearer boundaries between what is safe to store and what must remain private.
- The final sequence improves security reasoning and also makes the ZKP feature easier to justify academically because it clearly separates “claim commitment” from “claim proof” and shows where verification occurs

3.6.6 ACTIVITY DIAGRAM EVOLUTION

Student registration activity

Why it evolved

- Implementation required explicit handling for:
  - Preventing duplicate student identity across universities (global index checks),
  - Wallet linking logic (use existing mapping vs new wallet),
  - Correct persistence boundaries (store private academic data in university DB, store identity mapping centrally).

Certificate issuance activity

Why it evolved

- The SRS shows minting as a linear flow. In practice it is asynchronous and needs:

    - Confirmation/finalization checks,
    - Logging + recovery handling,
    - Persistence after confirmation (not before).
- Splitting the flow reduces confusion and documents the real production behaviour more accurately.

Certificate verification activity

Why it evolved

- Final design separates “certificate verification” from “ZKP verification” to avoid mixing base authenticity checks with optional claim validation.
- This also matches the portal UI: certificate authenticity must be verifiable even if the employer never verify the claims.

Proof generation activity

Why it evolved

- Implementation required additional steps such as structured commitment retrieval, proof status, optional validation, and persistence of proof metadata for future verification.

Revocation activity

Why it evolved

- Revocation is treated as a lifecycle state change requiring:
    - Blockchain-level burn,

- Shared index update for public verifiers,
- Consistent marking of revocation state in private DB for university reporting.

3.6.7 STATE DIAGRAM EVOLUTION (TRACKING REAL CERTIFICATE LIFECYCLE STATES)

What changed

- SRS lifecycle is minimal (Draft → Issued → Shared → Revoked).
- Final lifecycle reflects the actual system behaviour, including:
  - Asynchronous minting progression,
  - Confirmation/failure possibilities,
  - Correct terminal revocation state,
  - Transitions that are triggered by real implemented events.

Why

- In implementation, “Issued” is not a single moment; it requires mint submission + confirmation + indexing, and failures must be representable.
- The final state diagram is essential to justify reliability and consistency, especially for blockchain-based issuance.

Overall, the deviations from the SRS diagrams represent normal iterative refinement during engineering: the SRS captured the intended system capabilities, while the finalized diagrams capture the validated and implemented design after confronting real constraints in blockchain confirmation, multi-tenant data separation, operational logging requirements, and privacy-

preserving claim verification. The final report therefore uses the finalized diagrams as the "as-built" specification, while this section provides traceability back to the SRS baseline.

3.7 SUMMARY OF WHAT IS DEFERRED TO CHAPTER 4

To avoid duplication, this chapter defines requirements + viewpoints + constraints + design drivers, while Chapter 4 will present the complete final design with all diagrams and detailed explanations of each workflow.

3.8 CHAPTER SUMMARY

This chapter defined the specification and design approach for GenuineGrads by summarizing stakeholder requirements, presenting the viewpoints used to describe the system, and documenting the constraints that guided final design decisions. It also justified why the final diagrams differ slightly from the initial SRS diagrams, reflecting implementation-driven refinement. The next chapter presents the finalized system design in detail with the full set of architecture, database, behavioral, and lifecycle diagrams.

CHAPTER 4: SYSTEM DESIGN

4.1 CHAPTER INTRODUCTION

This chapter presents the final implemented design of GenuineGrads using multiple viewpoints: static architecture, data design, and dynamic behaviour (workflows, interactions, and lifecycle). The diagrams included in this chapter replaced the preliminary SRS diagrams and reflect the system as implemented, as justified in Section 3.6. (SRS, 2025; Interim Report, 2025)

4.2 STATIC ARCHITECTURE DESIGN

4.2.1 LAYERED ARCHITECTURE OVERVIEW

GenuineGrads follows a layered architecture consisting of a presentation layer (role-based portals), a API gateway layer (GraphQL API + internal services), a business logic layer (Solana program interactions and cNFT minting, and other services), and data storage layer. The architecture explicitly models the system's dependency on Solana infrastructure and the NFT indexing/retrieval services used for verification and asset metadata resolution.

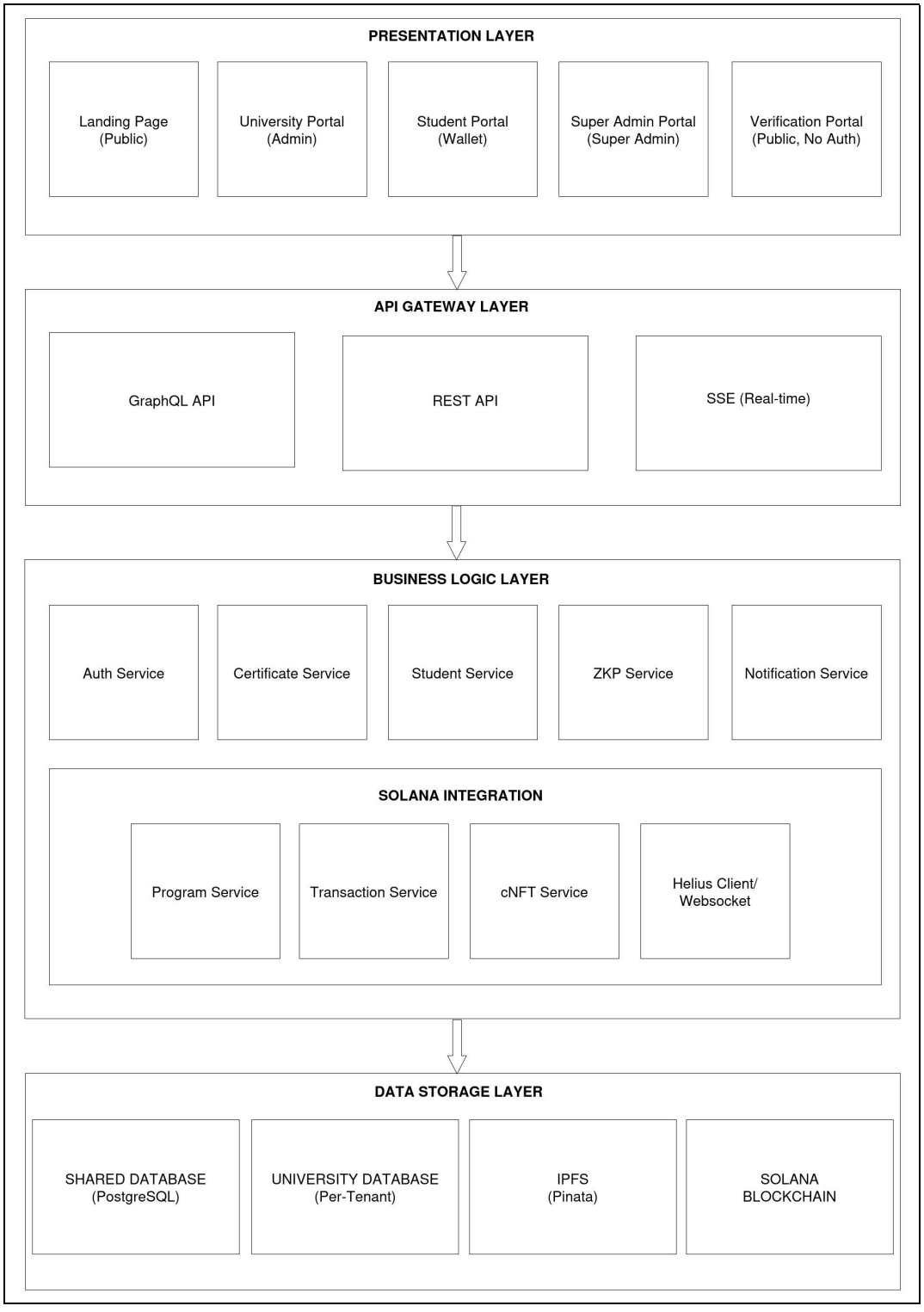


Figure 1: System Architecture Diagram

4.2.2 COMPONENT RESPONSIBILITIES

- University Portal (React): student registration (single/bulk), certificate template management, issuance/revocation operations, and administrative analytics.
- Student Dashboard (React): wallet-based authentication, certificate/achievement browsing, ZKP proof generation, and sharing flows.
- Employer/Verifier Portal (React/Public): certificate verification by QR/ID/mint address and optional ZKP proof verification.
- Backend GraphQL API (Node.js): orchestration layer exposing all application capabilities; enforces authorization and tenant boundaries coordinates blockchain calls, persistence, and verification logic.
- Auth Service: authentication (wallet signature/JWT issuance) and role-based access control.
- ZKP Service: manages ZKP-related storage, proof generation request routing, proof verification endpoint, and linking proofs to supported claims.
- Solana Smart Contract (Anchor): program rules and on-chain constraints for certificate issuance/revocation operations.
- Bubblegum v2 (cNFT): compressed NFT minting mechanism used for scalable issuance.
- Helius RPC / DASAPI / Webhooks: transaction broadcast, confirmation support, compressed asset retrieval, and webhook-driven finalization hooks.
- Storage: shared central DB + per-university private DB + IPFS(Pinata) for certificate metadata and assets.

4.3 DATA DESIGN

4.3.1 DATA PARTITIONING STRATEGY (SHARED VS PRIVATE DATABASES)

To support multi-university operation while maintaining strict institutional boundaries, GenuineGrads separates persistence into:

- A Shared Centralized Database for platform-wide indexes and logs that enable global verification and operational monitoring, and
- A Private University Database per institution containing student academic records, enrollments, and certificate issuance state.

This design reduces cross-university exposure risk and simplifies tenant enforcement while still supporting public verification workflows that require globally accessible revocation/index data.

4.3.2 SHARED CENTRALIZED DATABASE

The shared database stores platform-level entities such as universities, platform admins, mint activity logs, a global student index (identity mapping), and the revocation index used by the verification portal.

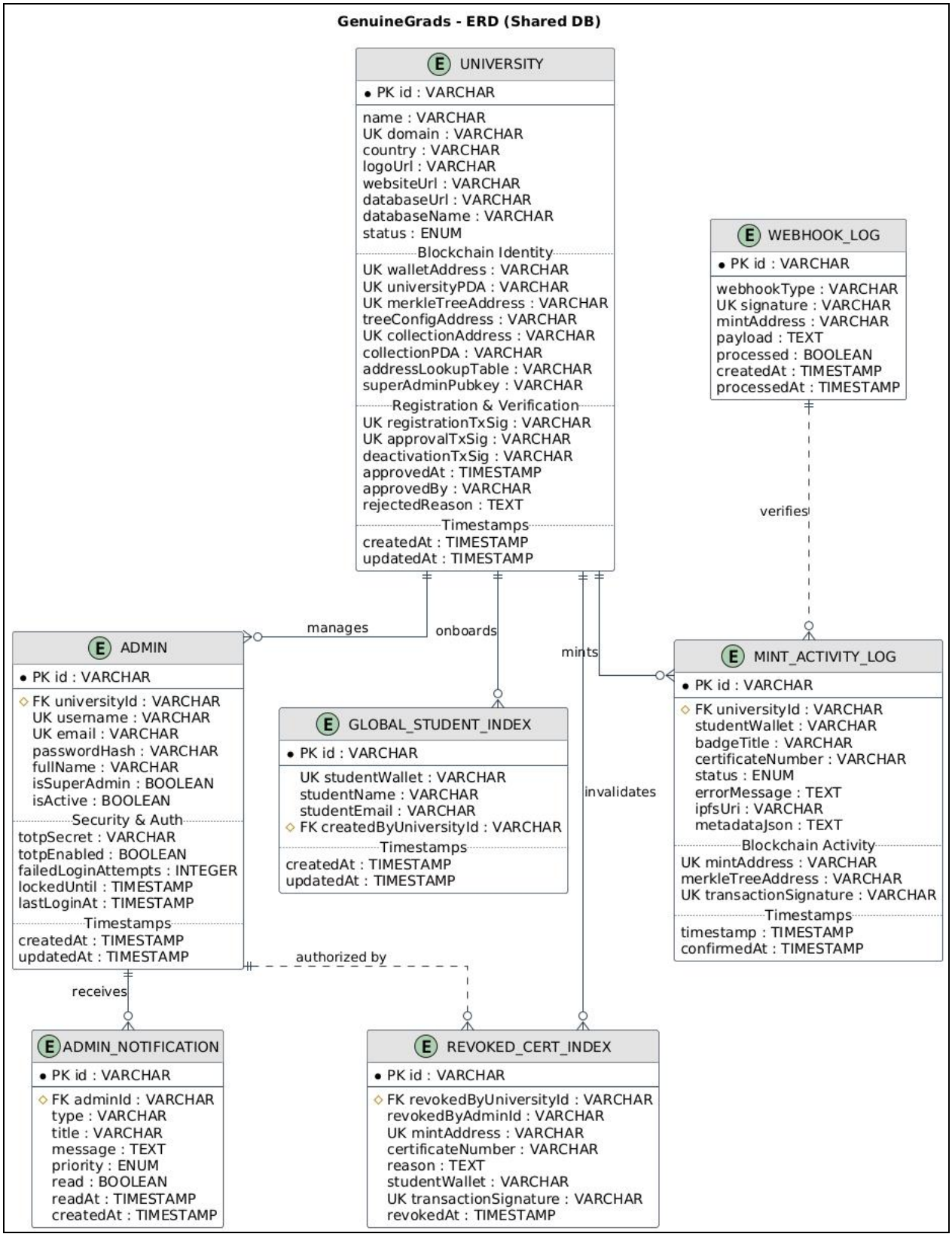


Figure 2: Shared Centralized Database ERD

4.3.3 PRIVATE UNIVERSITY DATABASE

Each university maintains a private schema containing academic entities such as students, courses, enrollments, achievements, certificates, and ZKP-related request/proof tracking relevant to that institution.

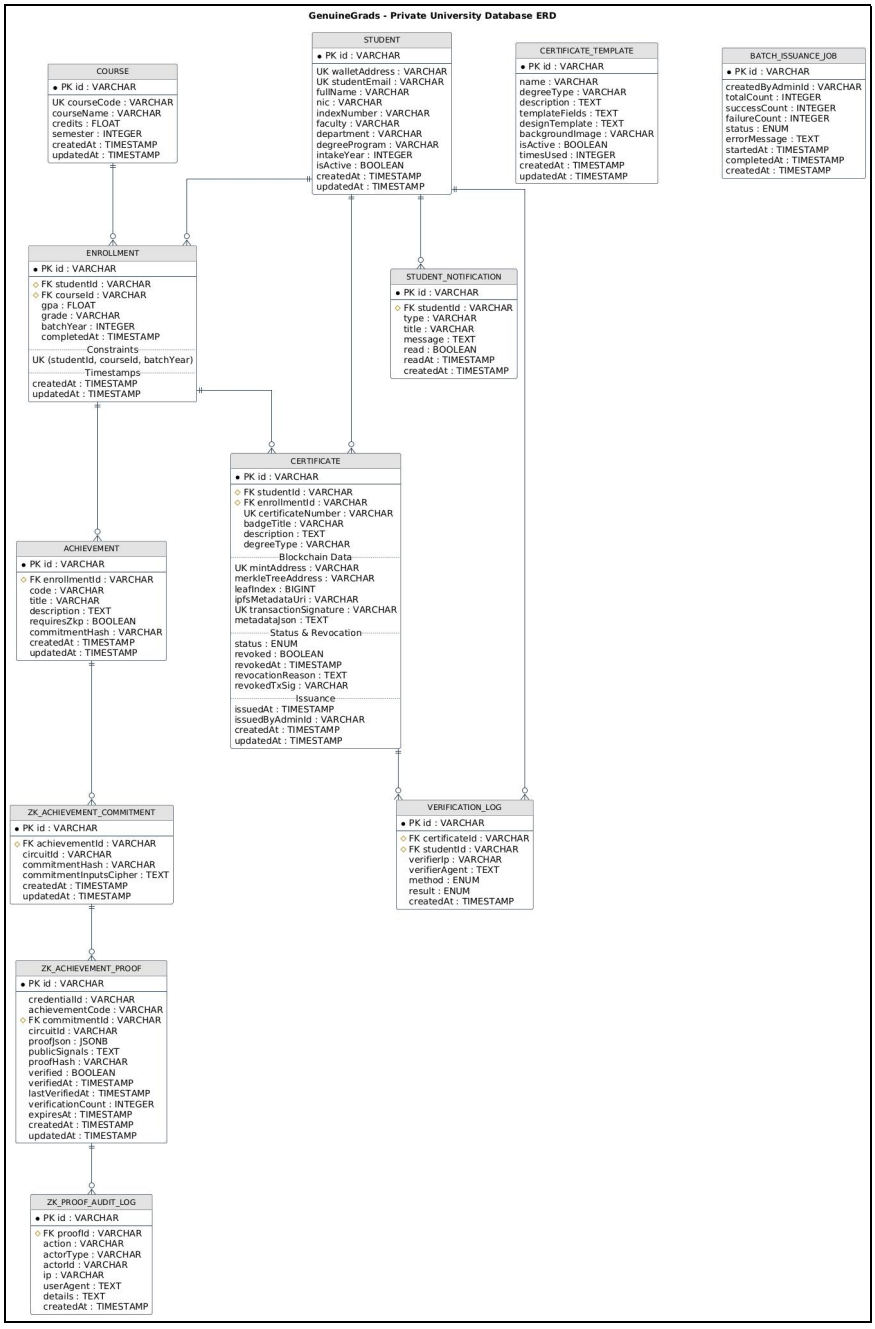


Figure 3: Private University Database ERD

4.4 USE CASE VIEW

Use case diagrams provide a stakeholder-level view of system capability boundaries and clarify role-based separation across portals.

4.4.1 UNIVERSITY ADMIN USE CASE

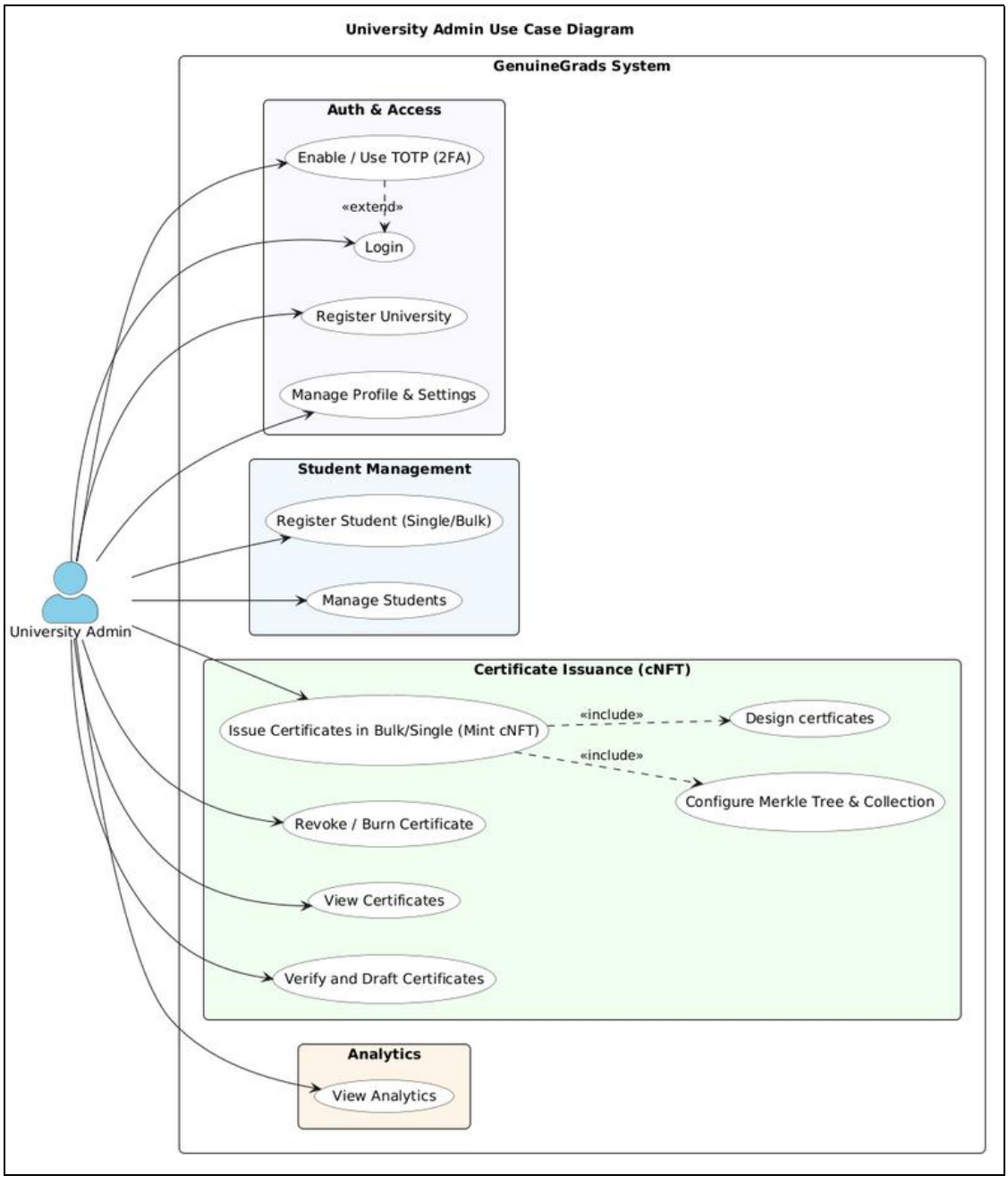


Figure 4: University Admin Use Case Diagram

4.4.2 STUDENT USE CASE

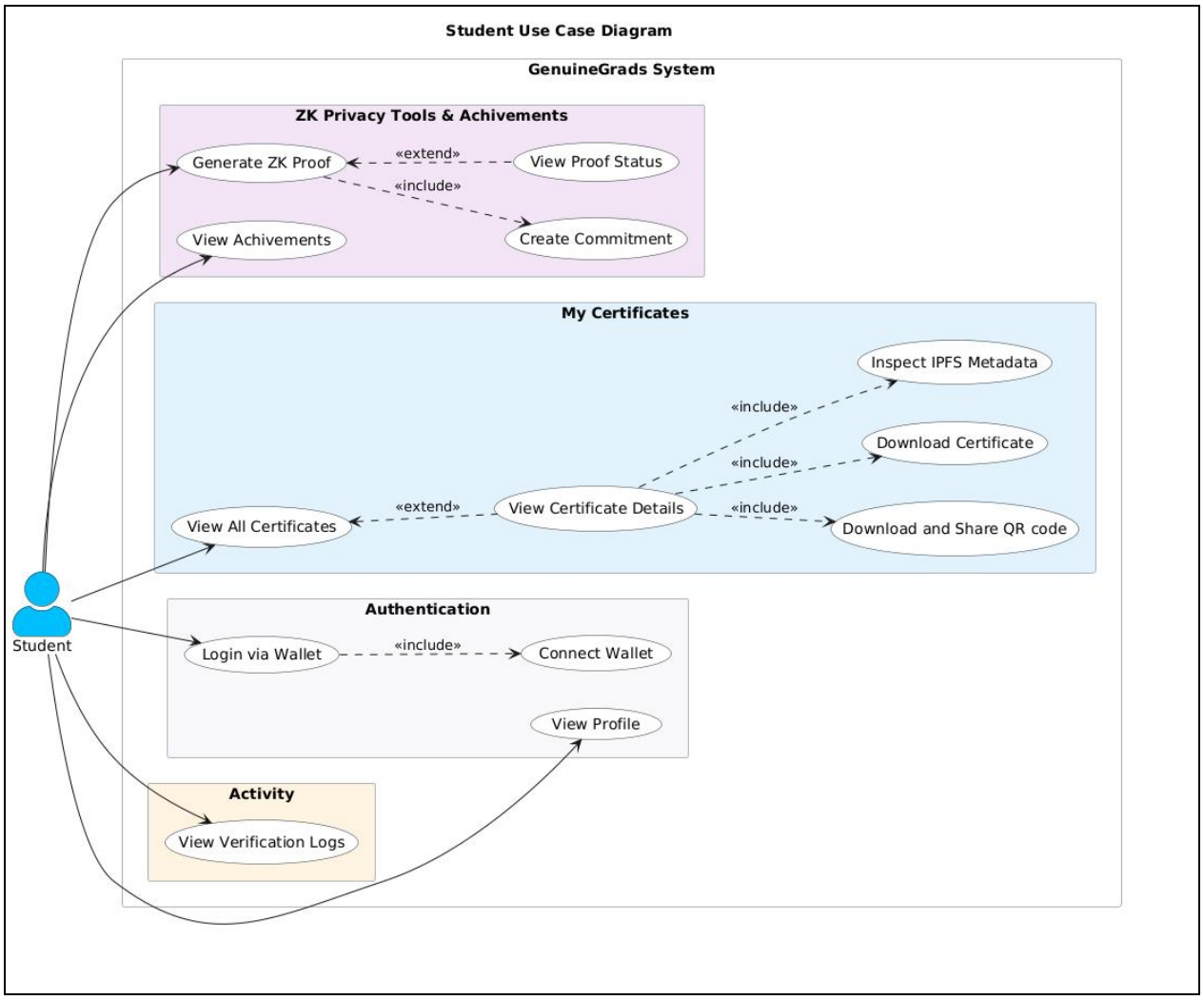


Figure 5: Student Use Case Diagram

4.4.3 SUPER ADMIN USE CASE

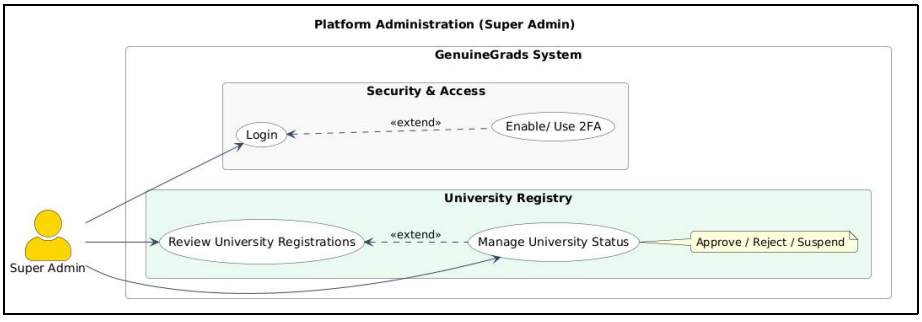


Figure 6: Super Admin Use Case Diagram

4.4.4 EMPLOYER/VERIFIER USE CASE

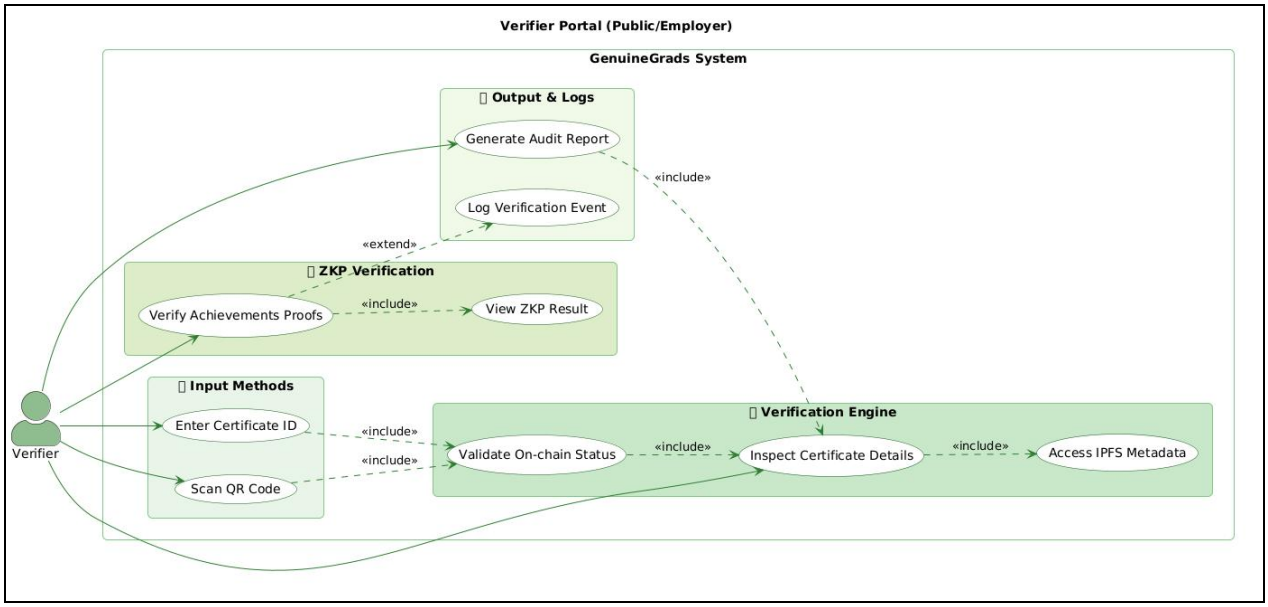


Figure 7: Employer/Verifier Use Case Diagram

4.5 DYNAMIC BEHAVIOUR DESIGN

This section describes the final workflows using activity, sequence, and state diagrams.

4.5.1 CERTIFICATE LIFECYCLE (STATE MODEL)

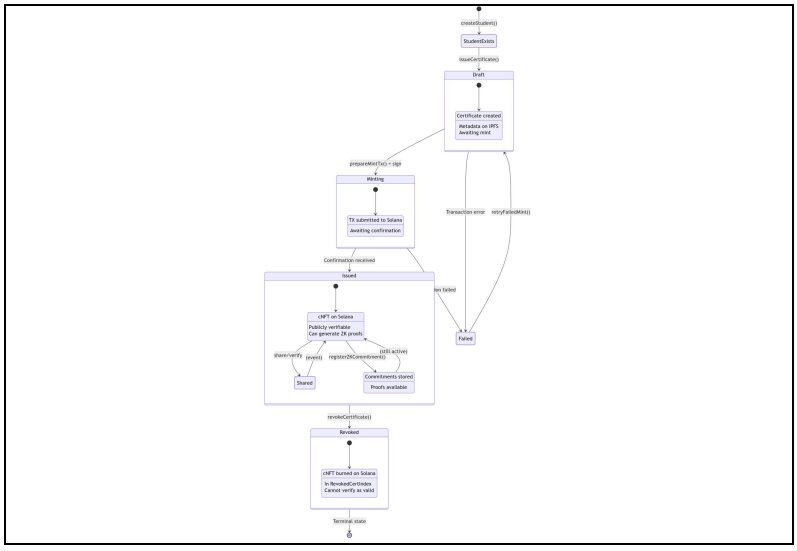


Figure 8: State Diagram - Certificate Lifecycle

4.6 CORE PROCESS FLOWS (ACTIVITY DIAGRAMS)

4.6.1 STUDENT REGISTRATION FLOW

This flow captures university-led registration (single/bulk) with wallet/NIC mapping and persistence split between private university storage and shared global indexing.

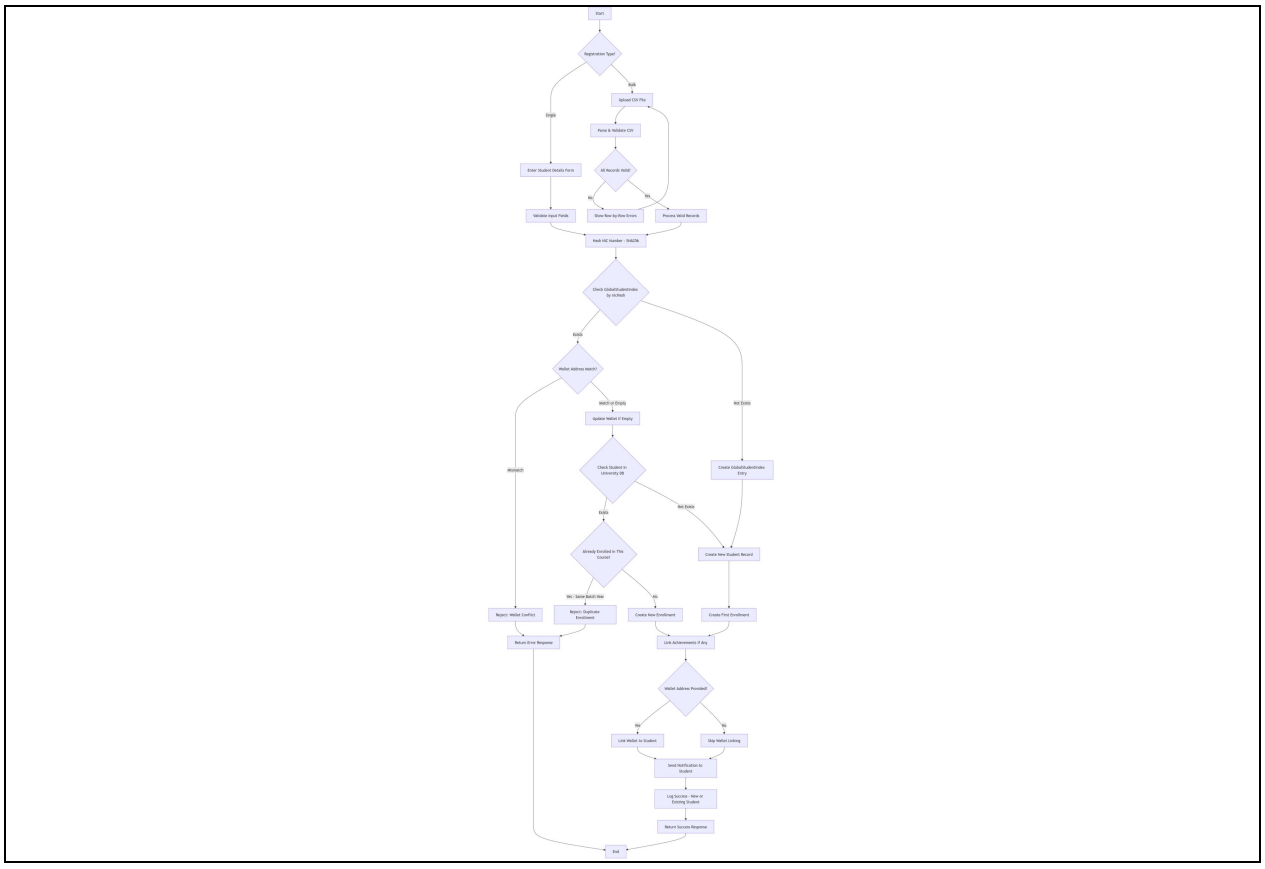


Figure 9: Activity - Student Registration

4.6.2 CERTIFICATE DRAFTING AND ASSET PREPARATION

This flow covers preparation of certificate metadata and any required assets prior to mint initiation (e.g., IPFS metadata upload decisions and template binding).

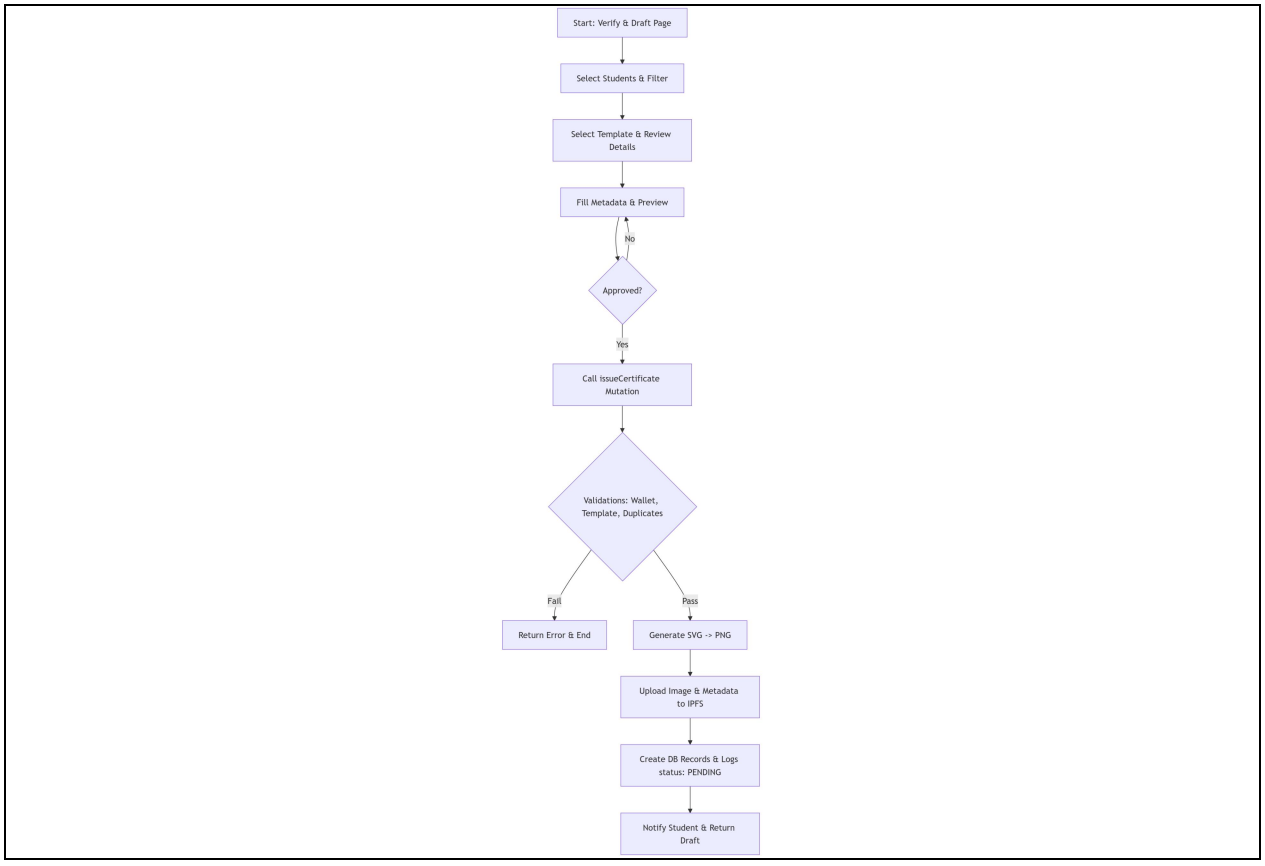


Figure 10: Activity - Drafting & Asset Preparation

4.6.3 MINTING INITIATION

This flow represents the initiation phase where issuance is requested, metadata is resolved, and mint submission is prepared.

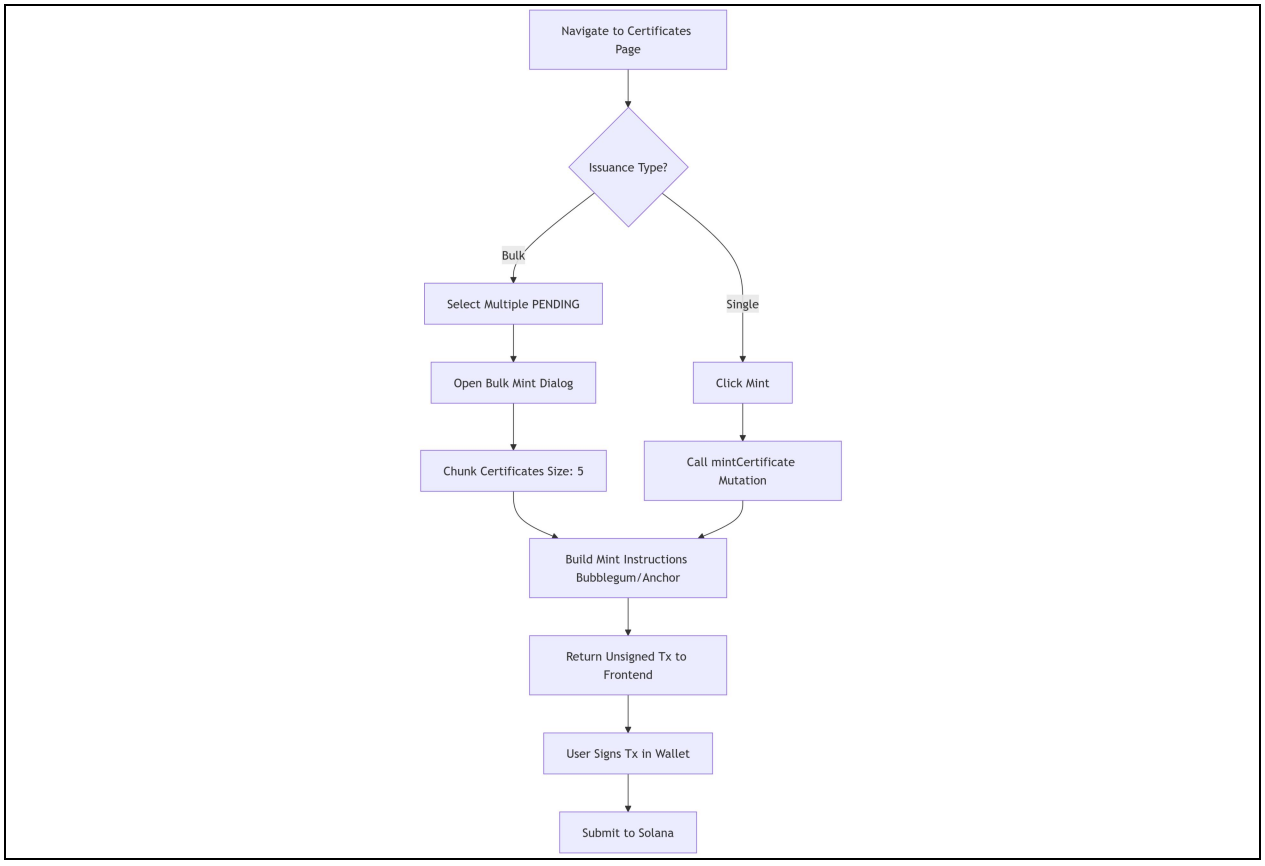


Figure 11: Activity - Minting Initiation

4.6.4 BLOCKCHAIN CONFIRMATION AND FINALIZATION

This flow represents confirmation/finalization steps driven by blockchain confirmation and webhook callbacks, ensuring that local DB state is only finalized once minting is confirmed.

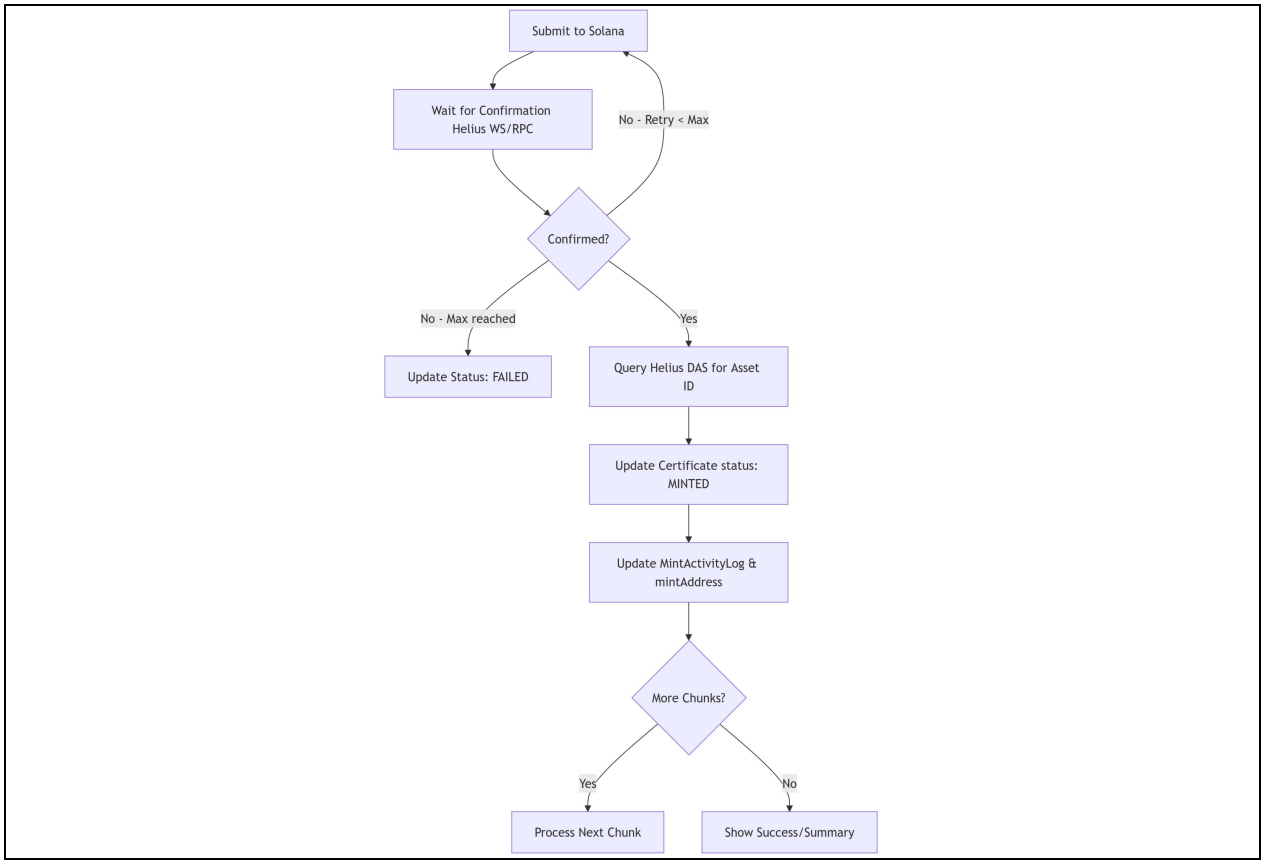


Figure 12: Activity - Blockchain Confirmation & Finalization

4.6.5 CERTIFICATE VERIFICATION

This flow describes public verification of a certificate using QR/ID/mint address, metadata retrieval, and revocation checks.

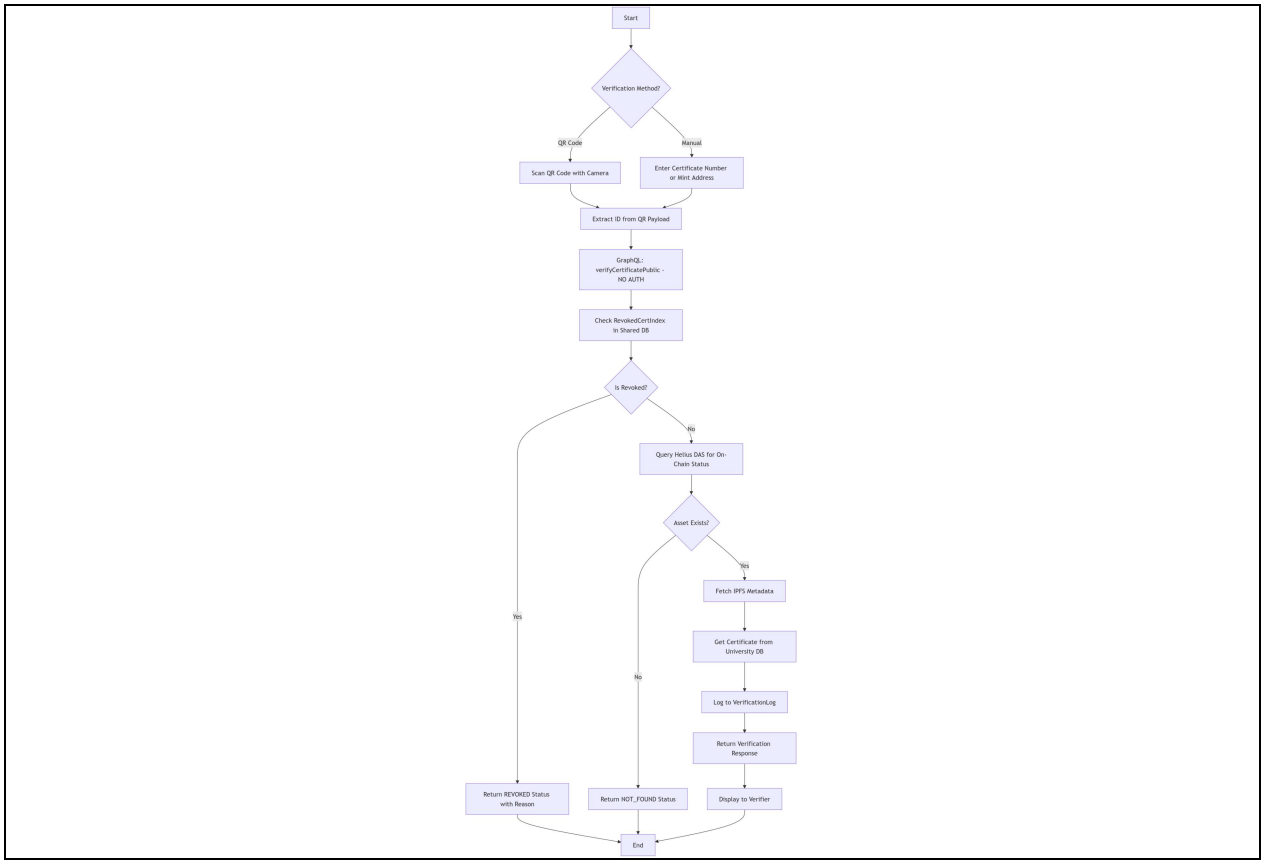


Figure 13: Activity - Certificate Verification

4.6.6 MINTING INITIATION

This flow represents the initiation phase where issuance is requested, metadata is resolved, and mint submission is prepared.

4.6.7 CERTIFICATE REVOCATION

This flow describes revocation through burning and shared index updates to ensure verifiers can reliably detect invalidated certificates.

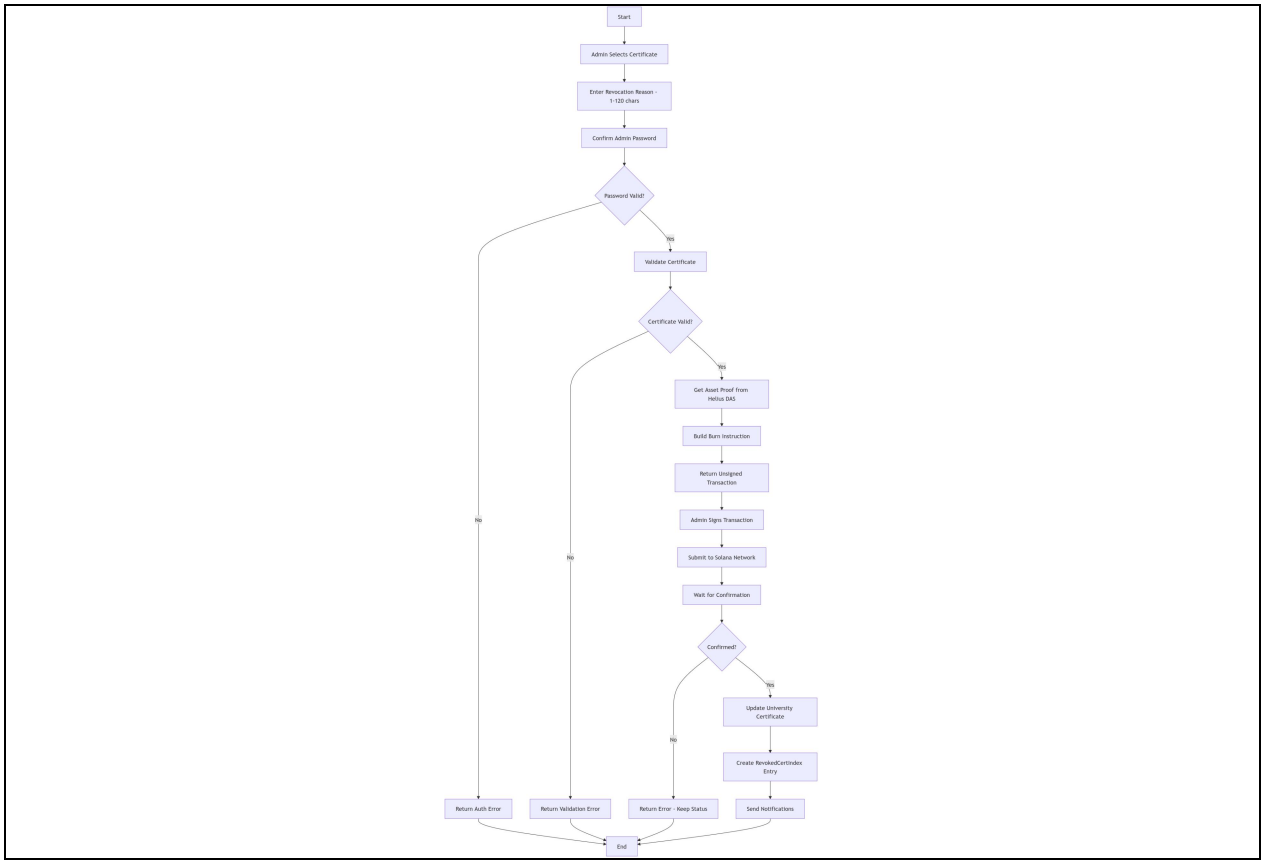


Figure 14: Activity - Certificate Revocation

4.7 ZKP SUBSYSTEM DESIGN

4.7.1 ZKP ARCHITECTURE FLOW

The ZKP subsystem is integrated to support privacy-preserving verification of selected claims (e.g., achievement predicates), allowing proof generation and verification without exposing sensitive underlying academic records.

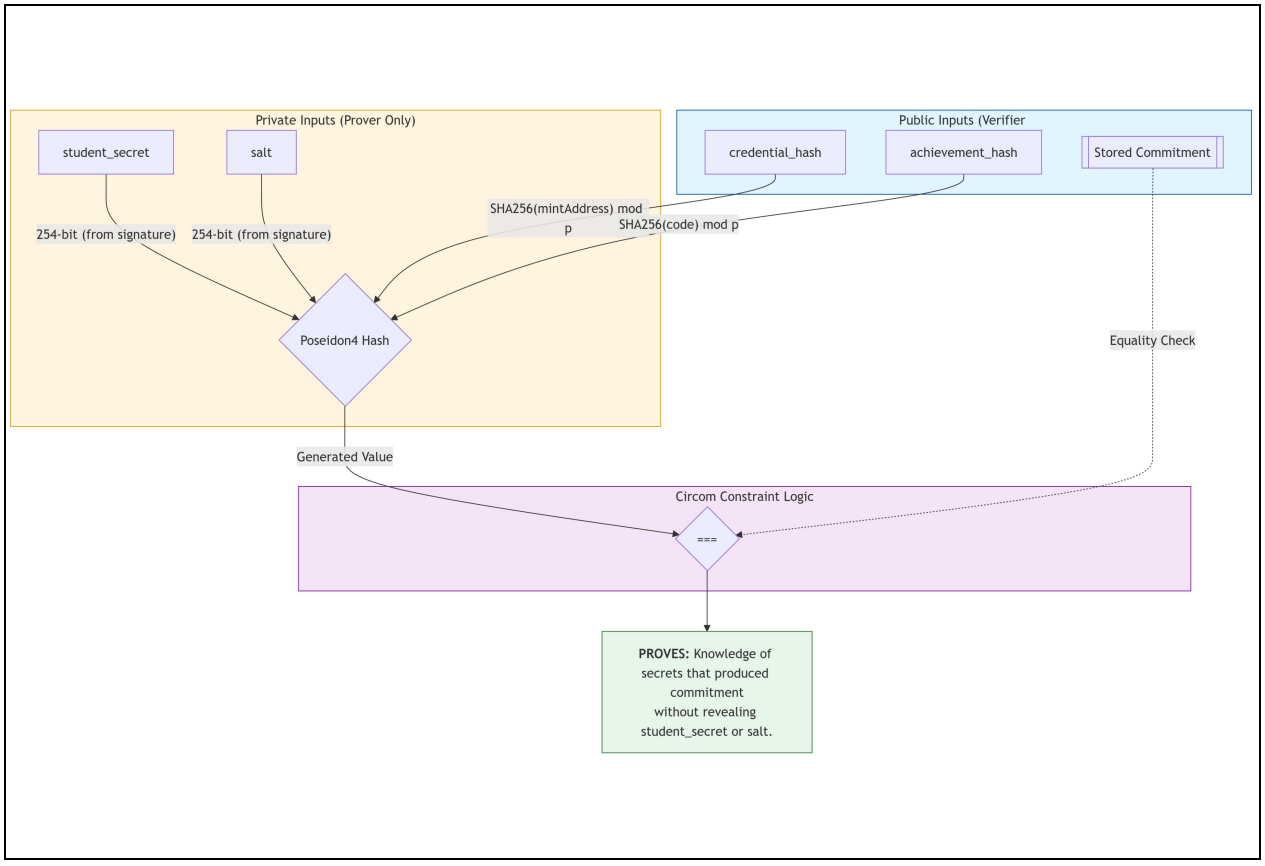


Figure 15: ZKP Architecture Flow

4.7.2 ZKP PROOF GENERATION FLOW

This flow describes how students request proofs, how inputs/commitments are fetched, how proof generation occurs, and how proof artifacts are returned to the user.

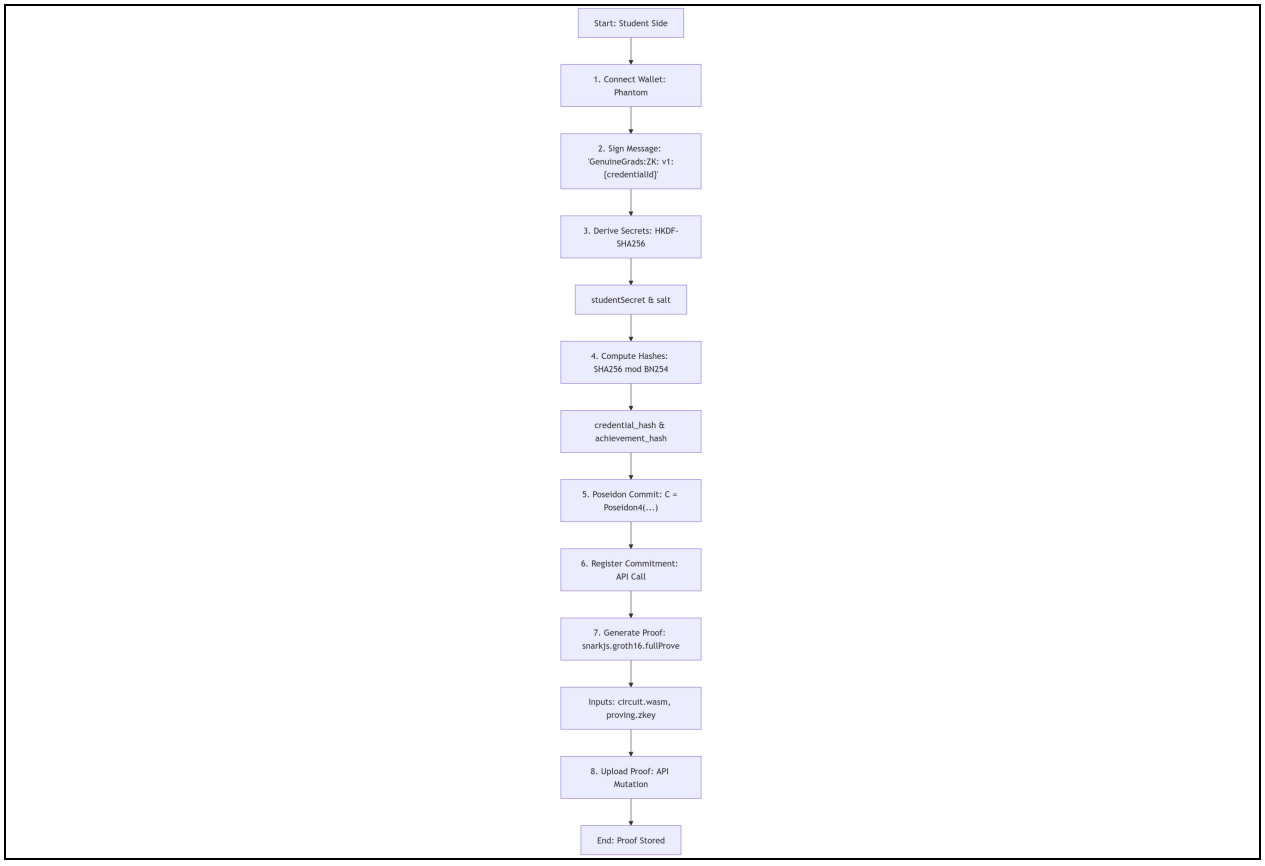


Figure 16: Activity - ZKP Proof Generation Flow

4.7.2 ZKP VERIFICATION FLOW

This flow describes verifier-side proof submission and verification using the verifier circuit and public inputs.

Figure 17: Activity - ZKP Verification Flow

4.8 INTERACTION DESIGN (SEQUENCE DIAGRAMS)

4.8.1 CERTIFICATE MINTING SEQUENCE

This sequence diagram shows the primary interactions between university portal, backend, blockchain services, and storage during certificate issuance.

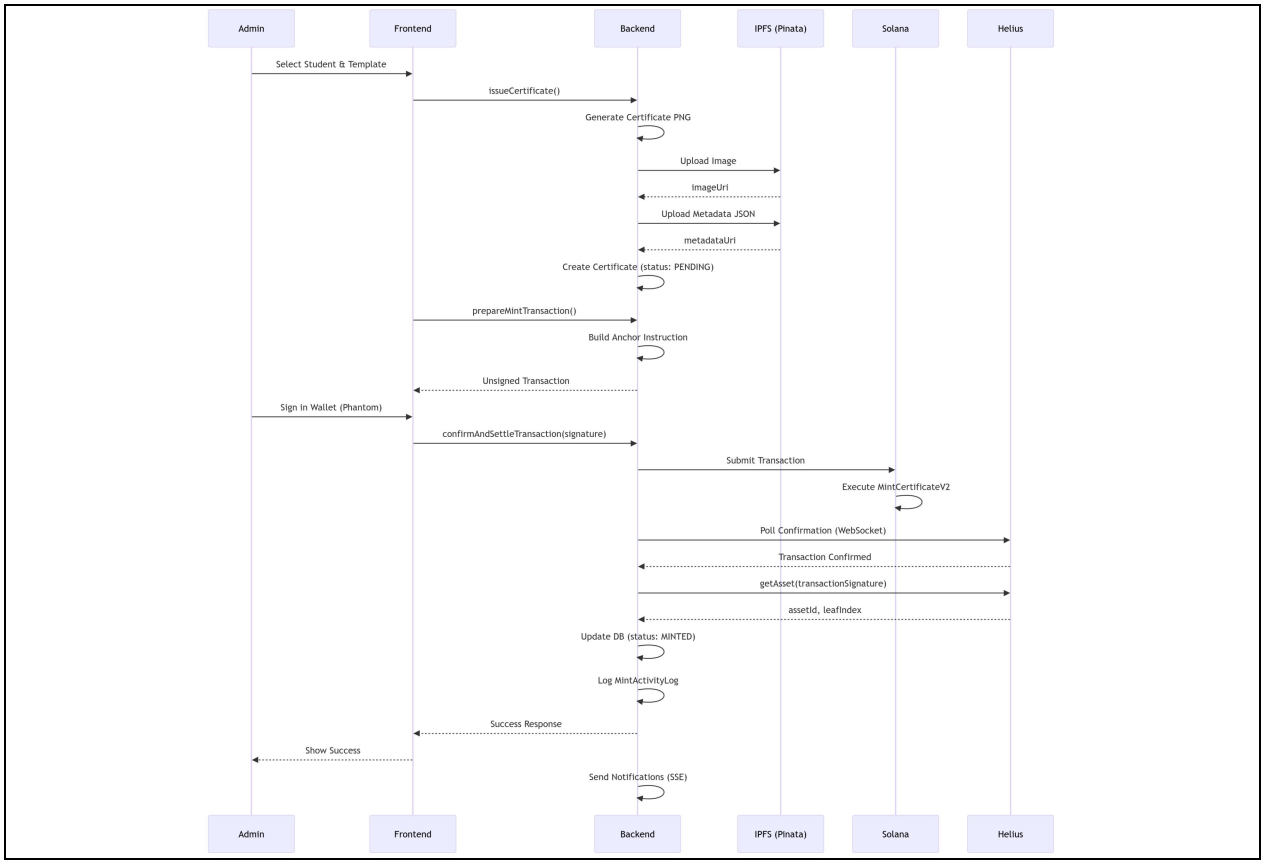


Figure 18: Sequence Diagram: Certificate Minting

4.8.2 CERTIFICATE VERIFICATION SEQUENCE

This sequence diagram shows verifier portal interactions with backend and NFT retrieval services, including revocation checks and UI output decisions.

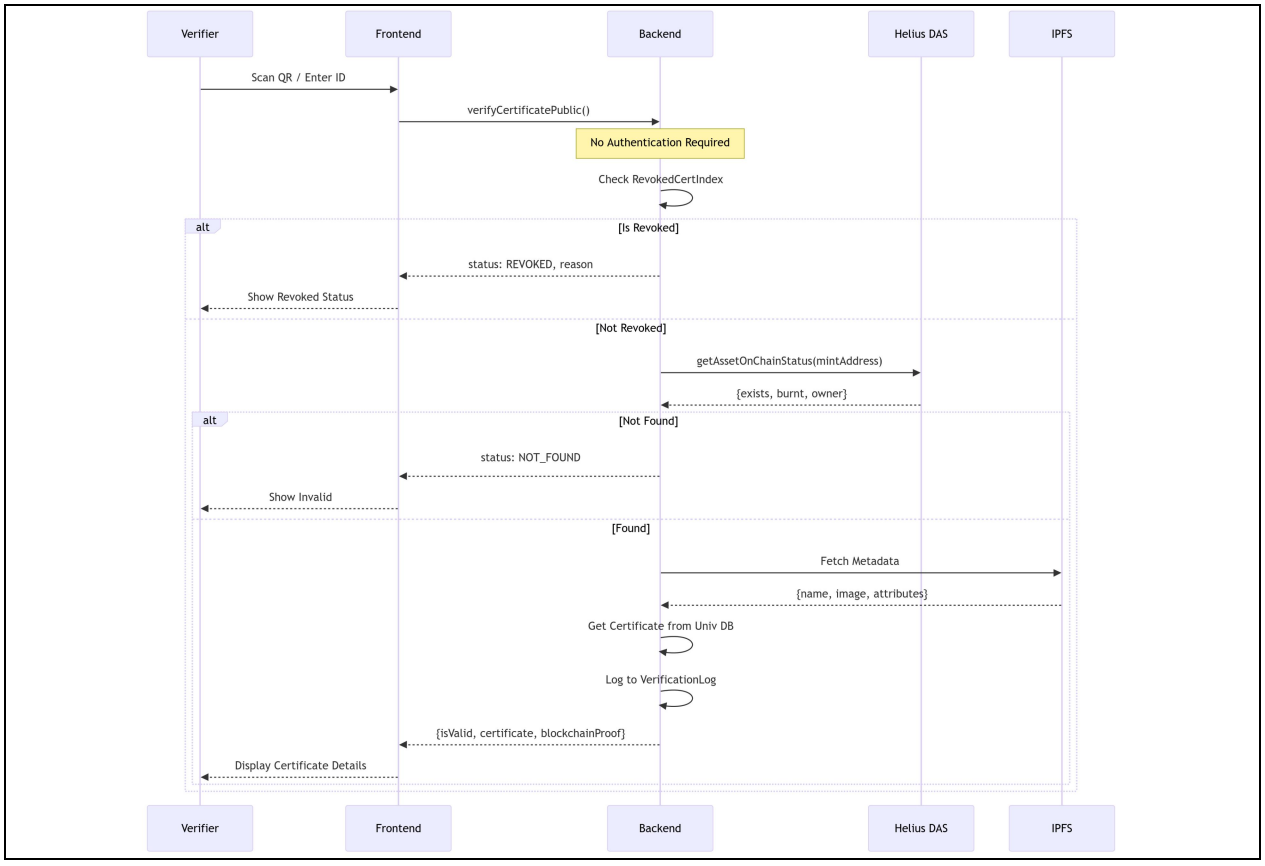


Figure 19: Sequence Diagram - Certificate Verification

4.8.3 ZKP PROOF GENERATION AND VERIFICATION SEQUENCE

This sequence diagram shows the student request path to generate a proof, and the verifier path to validate it.

4.9 DEPLOYMENT DESIGN

4.9.1 DEPLOYMENT TOPOLOGY

The deployment diagram summarizes how the system components are deployed and how the portals, backend services, databases, and external blockchain/indexing services relate operationally.

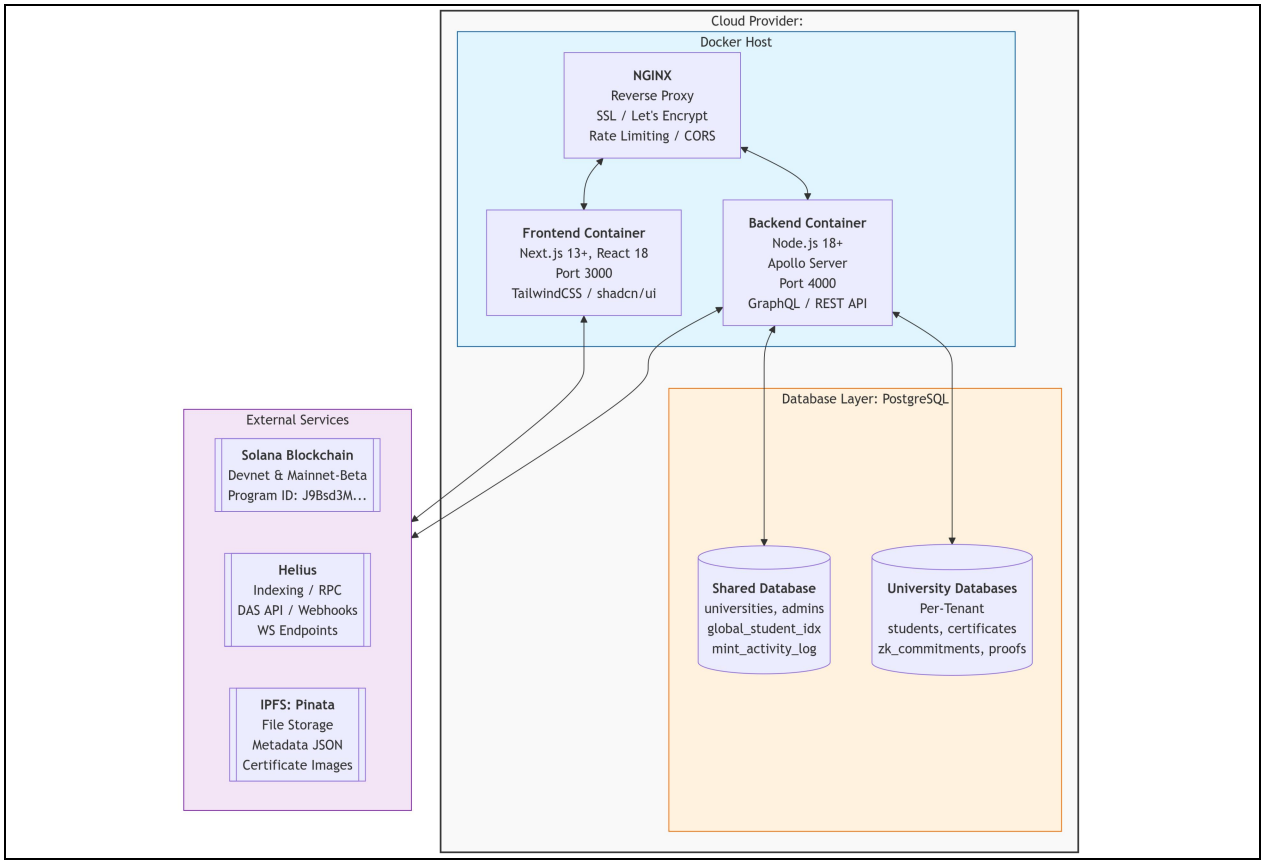


Figure 20: Deployment Diagram

4.10 CHAPTER SUMMARY

This chapter presented the final implemented design of GenuineGrads using architecture, data, behavioural, lifecycle, ZKP, and deployment viewpoints. The finalized diagrams provide the basis for understanding how the system satisfies the requirements described in Chapter 3 and prepare the foundation for Chapter 5, which details how these design elements were implemented and integrated.

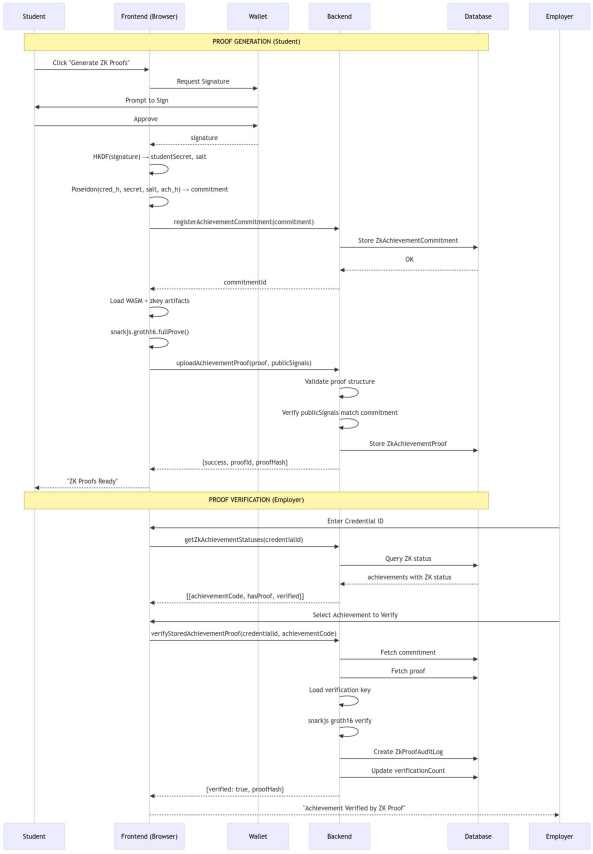


Figure 21: Sequence Diagram - ZKP Proof Generation & Verification

CHAPTER 5: IMPLEMENTATION

5.1 CHAPTER INTRODUCTION

This chapter describes the realization of the GenuineGrads system at the code level, focusing on critical implementation pieces, innovative solutions, and challenges encountered during development.

5.2 CRITICAL IMPLEMENTATION COMPONENTS

5.2.1 COMPRESSED NFT CERTIFICATE MINTING

The core functionality of the system involves minting academic certificates as compressed NFTs (cNFTs) on Solana. Traditional NFTs cost approximately $2-3 per mint, making bulk certificate issuance economically unfeasible. Using Metaplex Bubblegum's compressed NFT technology, we reduced costs to approximately $0.0001 per certificate.

Transaction Preparation (Backend)

The backend prepares versioned transactions that support larger payloads and Address Lookup Tables (ALT) for transaction size optimization:

```
// From transaction.service.ts:152-199
export async function prepareVersionedTransaction(params: {
  instructions: TransactionInstruction[];
  feePayer: PublicKey;
  addressLookupTableAccounts?: AddressLookupTableAccount[];
}): Promise<{
  transaction: string;
  blockhash: string;
  lastValidBlockHeight: number;
}> {
  const { instructions, feePayer, addressLookupTableAccounts = [] } = params;

  const { blockhash, lastValidBlockHeight } =
    await connection.getLatestBlockhash('confirmed');

  const message = new TransactionMessage({
    payerKey: feePayer,
    recentBlockhash: blockhash,
    instructions,
  });

  // Compile to V0 message with optional lookup tables
  const messageV0 = message.compileToV0Message(addressLookupTableAccounts);
  const versionedTransaction = new VersionedTransaction(messageV0);

  return {
    transaction: bs58.encode(versionedTransaction.serialize()),
    blockhash,
    lastValidBlockHeight,
  };
}
```

Figure 22: Transaction Preparation

This approach allows the frontend to receive a base58-encoded transaction, sign it with the user's wallet, and submit it to the Solana network.

Program Derived Addresses (PDAs)

The Solana program uses PDAs to manage on-chain state. Each entity has a deterministic address derived from seeds:

```
// From program.service.ts:84-116
export function deriveGlobalConfigPDA(superAdmin: PublicKey): [PublicKey, number] {
  return PublicKey.findProgramAddressSync(
    [Buffer.from('global-config'), superAdmin.toBuffer()],
    GENUINEGRADS_PROGRAM_ID
  );
}

export function deriveUniversityPDA(authority: PublicKey): [PublicKey, number] {
  return PublicKey.findProgramAddressSync(
    [Buffer.from('university'), authority.toBuffer()],
    GENUINEGRADS_PROGRAM_ID
  );
}

export function deriveUniversityTreePDA(merkleTree: PublicKey): [PublicKey, number] {
  return PublicKey.findProgramAddressSync(
    [Buffer.from('university_tree'), merkleTree.toBuffer()],
    GENUINEGRADS_PROGRAM_ID
  );
}
```

Figure 23: Program Derived Addresses (PDAs)

5.2.2 SEMI-AUTOMATED BULK CERTIFICATE ISSUANCE

Bulk certificate issuance presented a significant implementation challenge. Universities need to issue hundreds or thousands of certificates during graduation ceremonies. The initial approach of processing all transactions simultaneously faced two critical constraints:

1. RPC Rate Limiting: Solana RPC providers (including Helius) impose rate limits that prevent submitting more than ~10-15 transactions per second.

2. Metaplex Bubblegum Batch SDK Constraint: The official Bubblegum Batch SDK enables true batch minting of thousands of cNFTs in a single transaction. However, it requires staking MPLX tokens, which was not feasible within the project's budget constraints [13].

The Solution: Chunked Processing with Rate Limit Awareness. We implemented a semi-automated bulk issuance system that processes certificates in controlled chunks of 5 transactions at a time, with delays between submissions to avoid rate limiting:

```
// From BulkMintDialog.tsx:55-57
const CHUNK_SIZE = 5; // Sign 5 transactions at a time
const DELAY_BETWEEN_SUBMISSIONS = 2500; // 2.5 seconds between submissions
const MAX_RETRIES = 3;
```

Figure 24: Chunked Processing for Bulk Issuance Settings

Chunk Processing Algorithm

The frontend divides certificates into chunks, signs all transactions in a chunk at once (reducing wallet popup fatigue), then submits them sequentially with delays:

```
// From BulkMintDialog.tsx:190-279
const processChunk = async (
  chunk: BatchJob['certificates'],
  startIndex: number
) => {
  const transactionsToSign: VersionedTransaction[] = [];
  const certIndexMap: number[] = [];

  // Step 1: Prepare all transactions for this chunk
  for (let i = 0; i < chunk.length; i++) {
    const cert = chunk[i];
    updateCertificateStatus(startIndex + i, { status: 'signing' });

    try {
      const txResponse = await graphqlClient.request<any>(
        `mutation MintCertificate($certificateId: ID!) {
          mintCertificate(certificateId: $certificateId) {
            transaction
          }
        }`,
        { certificateId: cert.certificateId }
      );

      const transactionBuffer = bs58.decode(txResponse.data.mintCertificate.transaction);
      const transaction = VersionedTransaction.deserialize(transactionBuffer);

      transactionsToSign.push(transaction);
      certIndexMap.push(startIndex + i);
    } catch (error) {
      updateCertificateStatus(startIndex + i, { status: 'failed', error: error.message });
    }
  }

  // Step 2: Sign all transactions in this chunk at once
  if (transactionsToSign.length > 0 && signAllTransactions) {
    const signedTransactions = await signAllTransactions(transactionsToSign);

    // Step 3: Submit transactions sequentially with delays
    for (let i = 0; i < signedTransactions.length; i++) {
      await submitTransaction(signedTransactions[i], chunk[i], certIndexMap[i]);

      // Add delay between submissions (except for last one)
      if (i < signedTransactions.length - 1) {
        await new Promise(resolve => setTimeout(resolve, DELAY_BETWEEN_SUBMISSIONS));
      }
    }
  }
};
```

Figure 25: Chunk Processing Algorithm

Batch Job Tracking

The system maintains detailed progress tracking through the BatchIssuanceJob model:

```
// From certificate.mutations.ts:944-979
const updates: any = {
  processedCount: { increment: 1 },
};

if (success) {
  updates.successCount = { increment: 1 };
  updates.successfulMints = { push: certificateId };
} else {
  updates.failedCount = { increment: 1 };

  const failedMints = batchJob.failedMints ? JSON.parse(batchJob.failedMints) : [];
  failedMints.push({
    certId: certificateId,
    error: error || 'Unknown error',
    timestamp: new Date().toISOString(),
  });
  updates.failedMints = JSON.stringify(failedMints);
}
```

Figure 26: Batch Job Tracking Algorithm

Time Investment: This feature consumed a disproportionate amount of development time due to the complexity of handling transaction failures, implementing retry logic, managing wallet signing for multiple transactions, and ensuring the UI provided real-time feedback on batch progress.

5.2.3 MULTI-TIER CERTIFICATE VERIFICATION

The verification system implements a three-tier approach to ensure both database consistency and on-chain integrity:

```
// From public.queries.ts:38-400
async verifyCertificatePublic(_, { certificateNumber, mintAddress }, context) {
  // Tier 1: Check revocation index (O(1) lookup)
  let revokedCert = await sharedDb.revokedCertIndex.findUnique({
    where: { mintAddress },
  });

  if (revokedCert) {
    return { isValid: false, status: 'REVOKED', ... };
  }

  // Tier 2: Query MintActivityLog for certificate details
  const mintLog = await sharedDb.mintActivityLog.findUnique({
    where: { mintAddress },
    include: { university: true },
  });

  if (!mintLog || mintLog.status === 'PENDING') {
    return { isValid: false, status: 'INVALID', ... };
  }

  // Tier 3: Verify on-chain using Helius DAS API
  const onChainStatus = await getAssetOnChainStatus(certificate.mintAddress);

  if (onChainStatus.burnt) {
    return { isValid: false, status: 'REVOKED', ... };
  }

  return { isValid: true, status: 'VALID', ... };
}
```

Figure 27: Multi-Tier Certificate Verification Algorithm

5.2.4 HELIUS DAS API INTEGRATION WITH INDEXING DELAY HANDLING

A significant challenge was handling the indexing delay between when a transaction is confirmed on Solana and when Helius's DAS (Digital Asset Standard) API indexes the new asset. We implemented an exponential backoff retry mechanism:

```
// From cnft.service.ts:9-88
async function getRecentlyMintedAsset(
  recipientAddress: string,
  merkleTreeAddress: string,
  maxRetries: number = 5,
  retryDelay: number = 2000
): Promise<{ assetId: string; leafIndex: number } | null> {
  for (let attempt = 0; attempt < maxRetries; attempt++) {
    const response = await helius.rpc.getAssetsByOwner({
      ownerAddress: recipientAddress,
      page: 1,
      limit: 100,
    });

    const matchingAssets = response.items?.filter((asset: any) =>
      asset.compression?.compressed === true &&
      asset.compression?.tree === merkleTreeAddress &&
      !asset.burnt
    );

    if (matchingAssets && matchingAssets.length > 0) {
      const mostRecent = matchingAssets[0];
      return {
        assetId: mostRecent.id,
        leafIndex: mostRecent.compression?.leaf_id ?? null,
      };
    }

    // Wait before retry if not the last attempt
    if (attempt < maxRetries - 1) {
      logger.info({ attempt: attempt + 1, retryDelay },
        'Asset not yet indexed, waiting before retry');
      await new Promise(resolve => setTimeout(resolve, retryDelay));
    }
  }

  return null;
}
```

Figure 28: Backoff Retry Mechanism Algorithm

5.2.5 PRIVACY-PRESERVING STUDENT IDENTITY

To protect student privacy while maintaining uniqueness across the platform, we implemented NIC (National Identity Card) hashing:

```
// Implementation approach
const nicHash = crypto.createHash('sha256')
  .update(nic.trim().toLowerCase())
  .digest('hex');

// Stored in GlobalStudentIndex for O(1) duplicate checking
```

Figure 29: NIC Hashing Algorithm

The GlobalStudentIndex stores only the hash and an encrypted email, enabling:

- Preventing duplicate registrations across universities

- No PII exposure in cross-university queries
- O(1) lookup for duplicate checking

5.3 SOLANA PROGRAM IMPLEMENTATION

The GenuineGrads Solana program is written in Rust using the Anchor framework and serves as the on-chain authority for certificate issuance, verification, and revocation. The program is deployed at address J9Bsd3MSutBQDqsaXc5otLvDmrrN7vEynx2fYaH296FX.

5.3.1 PROGRAM ARCHITECTURE

The program follows a modular architecture with clear separation of concerns:

```
programs/genuinegrads/src/
├── lib.rs                    # Program entry point and instruction handlers
├── errors.rs                 # Custom error definitions
├── events.rs                 # On-chain event emissions
├── utils.rs                  # Helper utilities
├── states/                   # Account state definitions
│   ├── global_config.rs      # Platform-wide configuration
│   ├── university.rs         # University account state
│   ├── university_collection.rs  # MPL Core collection reference
│   └── university_tree.rs    # Merkle tree reference
└── instructions/             # Instruction handlers
    ├── initialize_config.rs      # Platform initialization
    ├── register_university.rs    # University registration
    ├── approve_university.rs     # University approval
    ├── deactivate_university.rs  # University deactivation
    ├── create_tree_v2.rs         # Merkle tree creation
    ├── create_core_collection_v2_cpi.rs  # Collection creation
    ├── mint_certificate_v2.rs    # Certificate minting
    └── burn_certificate_v2.rs    # Certificate revocation
```

Figure 30: Solana Program Architecture

5.3.2 ON-CHAIN STATE ACCOUNTS

The program manages four primary account types using Program Derived Addresses (PDAs):

GlobalConfig - Platform-wide settings controlled by the super admin:

```rust
// From global_config.rs
#[account]
#[derive(InitSpace)]
pub struct GlobalConfig {
    /// The single owner (super admin) who governs the program settings
    pub owner: Pubkey,
    /// Emergency freeze flag for all university operations
    pub frozen: bool,
    /// PDA bump seed
    pub bump: u8,
}
// PDA seeds: ["global-config", super_admin_pubkey]
```

Figure 31: Global Config State

University - Represents an approved educational institution:

```rust
// From university.rs
#[account]
#[derive(InitSpace)]
pub struct University {
    /// Program-level owner (super admin) from GlobalConfig.owner
    pub admin: Pubkey,
    /// University's operational authority (their signer wallet)
    pub authority: Pubkey,
    #[max_len(64)]
    pub name: String,
    #[max_len(60)]
    pub metadata_uri: String,
    pub is_active: bool,
    pub created_at: i64,
    pub bump: u8,
}
// PDA seeds: ["university", university_authority_pubkey]
```

Figure 32: University State

5.3.3 CERTIFICATE MINTING INSTRUCTION

The mint_certificate_v2 instruction is the core functionality that issues a certificate as a compressed NFT:

```rust
// From mint_certificate_v2.rs:113-212
pub fn handler(ctx: Context<MintCertificateV2>, args: MintCertificateArgs) -> Result<()> {
    // --- Governance guards ---
    require!(!ctx.accounts.global_config.frozen, GenuineGradsError::Frozen);
    require!(ctx.accounts.university.is_active, GenuineGradsError::UniversityInactive);

    // Metaplex Bubblegum enforces max 32 chars for name
    require!(args.name.len() > 0 && args.name.len() <= 32, GenuineGradsError::InvalidName);
    require!(args.uri.len() > 0 && args.uri.len() <= 200, GenuineGradsError::InvalidUri);

    // Verify tree_config PDA belongs to this merkle_tree
    let (expected_tree_config, _) = Pubkey::find_program_address(
        &[ctx.accounts.merkle_tree.key.as_ref()],
        &BUBBLEGUM_ID
    );
    require_keys_eq!(ctx.accounts.tree_config.key(), expected_tree_config,
                     GenuineGradsError::InvalidTreeConfig);

    // Build Bubblegum MintV2 CPI
    let mut cpi = MintV2CpiBuilder::new(&ctx.accounts.bubblegum_program);

    cpi.metadata(mpl_bubblegum::types::MetadataArgsV2 {
        name: args.name.clone(),
        symbol: "GG-CERT".to_string(),
        uri: args.uri.clone(),
        seller_fee_basis_points: 0,       // No royalties for educational certs
        primary_sale_happened: false,
        is_mutable: false,                // Certificates are immutable
        token_standard: Some(TokenStandard::NonFungible),
        collection: Some(ctx.accounts.core_collection.key()),
        creators: vec![Creator {
            address: ctx.accounts.university_authority.key(),
            verified: true,
            share: 100,
        }],
    });

    // Invoke Bubblegum CPI
    cpi.invoke()?;

    // Emit on-chain event for indexing
    emit!(CertificateMintedV2 {
        admin: ctx.accounts.global_config.owner,
        university: ctx.accounts.university.key(),
        recipient: args.recipient,
        merkle_tree: ctx.accounts.merkle_tree.key(),
        name: args.name,
        uri: args.uri,
        // ... additional fields
    });

    Ok(())
}
```

Figure 33: Certificate Minting Instruction Handler

Key Implementation Details:

1. Governance Guards: Every instruction checks if the platform is frozen and if the university is active before proceeding.

2. PDA Verification: The program verifies that all PDAs (tree_config, university, collection) are correctly derived to prevent spoofing.

3. Immutable Certificates: Certificates are minted with is_mutable: false to ensure academic credentials cannot be altered after issuance.

4. Zero Royalties: seller_fee_basis_points: 0 ensures certificates have no transfer fees, appropriate for educational credentials.

5.3.4 CERTIFICATE REVOCATION (BURN) INSTRUCTION

The burn_certificate_v2 instruction allows universities to revoke fraudulent or erroneous certificates:

```rust
// From burn_certificate_v2.rs:132-267
pub fn handler<'info>(
    ctx: Context<'_, '_, '_, 'info, BurnCertificateV2<'info>>,
    args: BurnCertificateArgs
) -> Result<()> {
    // Governance guards
    require!(!ctx.accounts.global_config.frozen, GenuineGradsError::Frozen);
    require!(ctx.accounts.university.is_active, GenuineGradsError::UniversityInactive);

    // Reason must be provided for audit trail
    require!(
        !args.reason.trim().is_empty() && args.reason.len() <= 120,
        GenuineGradsError::InvalidBurnReason
    );

    // Merkle proof must be provided as remaining accounts
    require!(!ctx.remaining_accounts.is_empty(), GenuineGradsError::MissingMerkleProof);

    // Build BurnV2 CPI with Merkle proof
    let cpi_args = BurnV2InstructionArgs {
        root: args.root,
        data_hash: args.data_hash,
        creator_hash: args.creator_hash,
        nonce: args.nonce,
        index: args.index,
        // ... additional fields
    };

    let cpi = BurnV2Cpi::new(&ctx.accounts.bubblegum_program, cpi_accounts, cpi_args);

    // Proof accounts passed as remaining_accounts
    let proof_accounts: Vec<_> = ctx.remaining_accounts
        .iter()
        .map(|ai| (ai, false, false))
        .collect();

    cpi.invoke_with_remaining_accounts(&proof_accounts)?;

    // Emit audit event with reason
    emit!(CertificateBurnedV2 {
        university: ctx.accounts.university.key(),
        leaf_owner: ctx.accounts.leaf_owner.key(),
        index: args.index,
        reason: args.reason,
        burned_at: Clock::get()?.unix_timestamp,
        // ... additional fields
    });

    Ok(())
}
```

Figure 34: Certificate Revocation Instruction Handler

5.3.5 ERROR HANDLING

The program defines specific error codes for clear debugging:

```rust
// From errors.rs
#[error_code]
pub enum GenuineGradsError {
    #[msg("Unauthorized")]
    Unauthorized,
    #[msg("Program is frozen")]
    Frozen,
    #[msg("University is inactive")]
    UniversityInactive,
    #[msg("Invalid tree config PDA")]
    InvalidTreeConfig,
    #[msg("Collection mismatch")]
    CollectionMismatch,
    #[msg("Tree mismatch")]
    TreeMismatch,
    #[msg("Missing merkle proof accounts")]
    MissingMerkleProof,
    #[msg("Invalid burn reason")]
    InvalidBurnReason,
    // ... additional errors
}
```

Figure 35: Solana Program Error Codes

5.3.5 CROSS-PROGRAM INVOCATIONS (CPIS)

The program integrates with multiple Solana programs via CPI:

Program	Purpose
Metaplex Bubblegum	Compressed NFT minting and burning
MPL Core	Collection management and verification
SPL Account Compression	Merkle tree operations
SPL Noop	Log wrapper for transaction logging

Table 1: CPI: Programs and Purpose

5.4 UNFORESEEN PROBLEMS AND SOLUTIONS

5.4.1 BULK MINTING ECONOMIC CONSTRAINTS

Problem: The Metaplex Bubblegum Batch SDK would have been ideal for bulk certificate issuance, allowing thousands of mints in a single transaction. However, using this SDK requires staking MPLX tokens, which was outside the project budget [13].

Solution: Implemented a semi-automated chunked processing system that:

- Processes 5 certificates at a time
- Uses 2.5-second delays between submissions
- Provides real-time progress feedback
- Supports pause/resume functionality

- Generates downloadable batch reports

Trade-off: While slower than true batch minting, this approach requires no token staking and works within free-tier RPC limits.

5.4.2 TRANSACTION SIZE LIMITATIONS

Problem: Solana transactions have a maximum size of ~1232 bytes. Complex minting transactions with many accounts exceeded this limit.

Solution:

1. Migrated from legacy transactions to versioned transactions (V0)
2. Implemented Address Lookup Tables (ALT) to reduce account reference sizes
3. Used Bubblegum's efficient compression for metadata

5.5 SECURITY IMPLEMENTATIONS

5.5.1 CLIENT-SIDE SIGNING PATTERN

No private keys are ever transmitted to or stored on the backend:

```
// Backend: Prepare unsigned transaction
const { transaction } = await prepareVersionedTransaction({
  instructions,
  feePayer: universityAuthority,
});

// Frontend: Sign with wallet adapter
const signedTx = await wallet.signTransaction(deserialize(transaction));

// Frontend: Submit to Solana
const signature = await connection.sendRawTransaction(signedTx.serialize());
```

Figure 36: Transaction Preparation and Signing Flow

5.5.2 PASSWORD SECURITY

Using Argon2 for password hashing (resistant to GPU attacks):

```
import argon2 from 'argon2';

const hash = await argon2.hash(password, {
  type: argon2.argon2id,
  memoryCost: 65536,
  timeCost: 3,
  parallelism: 4,
});
```

Figure 37: Password Hashing

5.5.3 SENSITIVE DATA ENCRYPTION

Sensitive fields like student emails are encrypted with AES-256:

```
// Encryption for GlobalStudentIndex.encryptedEmail
const cipher = crypto.createCipheriv('aes-256-gcm', key, iv);
```

Figure 38: Sensitive Data Encryption

5.6 PERFORMANCE OPTIMIZATIONS

5.6.1 DATABASE INDEXING STRATEGY

Comprehensive indexes on frequently queried fields:

```
@@index([nicHash])
@@index([walletAddress])
@@index([studentWallet, universityId, mintAddress, certificateNumber, status, timestamp])
```

Figure 39: Database Indexing

5.6.2 MERKLE PROOF TRUNCATION

When burning certificates, we optimize proof transmission by removing nodes covered by the Merkle tree's canopy:

```
const requiredProofLength = maxDepth - canopyDepth;
const truncatedProofNodes = proof.proof.slice(0, requiredProofLength);
```

Figure 40: Optimize Merkle Proof

5.7 CHAPTER SUMMARY

The implementation of GenuineGrads successfully realized the system design while overcoming significant technical challenges. Key achievements include:

1. Cost-effective certificate issuance at ~$0.0001 per certificate using compressed NFTs
2. Custom Solana program with governance controls, CPI integrations with Metaplex Bubblegum and MPL Core, and comprehensive event emissions for off-chain indexing
3. Practical bulk issuance through semi-automated chunked processing when the ideal batch SDK was economically unfeasible
4. Robust verification through multi-tier database and on-chain checks
5. Privacy-preserving architecture with hashed identities and encrypted PII

The Solana program implementation demonstrates secure on-chain governance with PDA-based access control, while the bulk issuance feature shows a pragmatic approach to working within real-world constraints (RPC rate limits, MPLX staking requirements) while still delivering functional requirements.

CHAPTER 6: RESULTS AND EVALUATION

6.1 CHAPTER INTRODUCTION

This chapter evaluates the extent to which the GenuineGrads project achieved its stated goals, describes the testing methodology employed, presents test results, and provides a critical appraisal of the project's strengths and weaknesses.

6.2 PROJECT GOALS AND ACHIEVEMENT STATUS

The GenuineGrads project set out to address the global problem of academic credential fraud by developing a blockchain-based certificate verification platform. The following table summarizes the original goals and their achievement status:

Goal	Status	Notes
Issue certificates as blockchain-based NFTs	Achieved	Using Solana compressed NFTs via Metaplex Bubblegum
Reduce minting costs to make bulk issuance viable	Achieved	~$0.0001 per certificate vs $2-3 for traditional NFTs
Enable public certificate verification	Achieved	Multi-tier verification with on-chain validation
Support bulk certificate issuance	Achieved	Semi-automated batch processing (5 per batch)
Implement certificate revocation	Achieved	Burn-based revocation with audit trail
Multi-university platform support	Achieved	Dual-database architecture with tenant isolation
Privacy-preserving student data	Achieved	NIC hashing, email

		encryption, no PII on-chain
Zero-knowledge proof verification	Achieved	Groth16 proofs with Poseidon commitments for selective disclosure
Custom certificate templates	Achieved	Visual designer with backend generation
Real-time notifications	Achieved	Server-Sent Events (SSE) implementation

Table 2: Goal Achievement Table

Overall Achievement Rate: 10 out of 10 goals fully achieved (100%)

6.3 TESTING METHODOLOGY

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

6.3.1 TESTING STRATEGY

Due to the multi-layered architecture spanning web frontend, GraphQL backend, and Solana blockchain, a comprehensive testing strategy was employed:

Layer	Testing Approach	Tools Used
Solana Program	Integration tests on local validator	Anchor framework, Mocha, Chai
Backend API	Manual testing, script-based validation	GraphQL Playground, custom scripts

Frontend	Manual testing	Browser-based testing
ZK Circuits	Unit tests for proof generation/verification	snarkjs, Mocha, Chai
End-to-End	Full workflow testing on Solana Devnet	Real wallet transactions

Table 3: Testing Strategy Table

6.3.2 RATIONALE FOR TESTING APPROACH

The project prioritized integration and end-to-end testing over unit tests due to:

1. Cross-System Dependencies: Most functionality involves multiple systems (frontend → backend → blockchain → IPFS), making isolated unit tests less valuable than integration tests.

2. Time Constraints: As an academic project with limited timeline, effort was focused on high-impact tests that validate complete user workflows.

6.4 SOLANA PROGRAM TEST RESULTS

The Solana program underwent rigorous testing using Anchor's test framework against a local validator. The test suite covers all program instructions:

6.4.1 TEST SUITE OVERVIEW

```
genuinegrads
  initialize_config
    ✓ creates the global config for superAdmin (2,341ms)
    ✓ fails to re-initialize same PDA (1,205ms)

  universities (register/approve/deactivate)
    ✓ registers a university (inactive by default) (1,876ms)
    ✓ superAdmin approves the university (1,543ms)
    ✓ registers second university, then deactivates after approval (3,421ms)

  create_core_collection_v2_cpi
    ✓ creates a real Core collection via CPI and records it (2,156ms)

  create_tree_v2
    ✓ creates a Bubblegum V2 tree via CPI and records it (4,532ms)

  mint_certificate_v2
    ✓ mints a compressed certificate (collection attached) (3,876ms)

  burn_certificate_v2
    ✓ burns a minted certificate with valid proof (5,234ms)
    ✓ fails to burn with empty reason (2,145ms)
    ✓ fails to burn with reason exceeding 120 characters (2,098ms)
    ✓ fails when non-authority tries to burn (1,987ms)
    ✓ fails when proof is missing (no remaining accounts) (1,654ms)

  13 passing (34.068s)
```

Figure 41: Solana Program Test Results

6.4.2 CRITICAL TEST CASES

1. Test 1: Global Configuration Initialization

Aspect	Details
Purpose	Verify program can be initialized and prevents double-initialization
Input	Super admin keypair
Expected	GlobalConfig PDA created with correct owner; second attempt fails
Result	Pass - PDA created correctly, re-initialization blocked

Table 4: Global Configuration Initialization Details

2. Test 2: University Registration and Approval Workflow

Aspect	Details
Purpose	Verify complete university lifecycle management
Input	University authority keypair, name, metadata URI
Expected	University registered as inactive, approved by super admin, can be deactivated
Result	Pass - All state transitions work correctly

Table 5: University Registration and Approval Workflow Details

3. Test 3: Certificate Minting

Aspect	Details
Purpose	Verify cNFT minting with collection attachment
Input	Certificate name, metadata URI, student wallet
Expected	Compressed NFT minted under university's collection
Result	Pass - Certificate minted, asset ID derivable from Merkle tree

Table 6: Certificate Minting Details

4. Test 4: Certificate Revocation (Burn)

Aspect	Details
Purpose	Verify certificates can be revoked with proper authorization
Input	Asset proof from DAS API, revocation reason
Expected	Certificate burned on-chain, event emitted with reason
Result	Pass - Certificate burned, proper audit trail created

Table 7: Certificate Revocation Details

5. Test 5: Authorization Controls

Aspect	Details
Purpose	Verify unauthorized users cannot perform privileged operations
Input	Non-authority attempting to burn certificate
Expected	Transaction rejected with Unauthorized error
Result	Pass - Authorization enforced correctly

Table 8: Authorization Controls Details

6. Test 6: Input Validation

Aspect	Details
Purpose	Verify program rejects invalid inputs
Input	Empty burn reason, reason > 120 characters, missing Merkle proof
Expected	Each case rejected with appropriate error
Result	Pass - All validation constraints enforced

Table 9: Input Validation Details

6.5 BACKEND API TESTING

6.5.1 MANUAL TESTING VIA GRAPHQL PLAYGROUND

The backend API was tested through GraphQL Playground during development. Key workflows tested:

Workflow	Mutations/Queries Tested	Result
Admin Authentication	loginSuperAdmin, loginUniversityAdmin, verify2FA	Pass
University Management	registerUniversity, approveUniversity, getUniversities	Pass
Student Registration	createStudent, getStudents, bulkImportStudents	Pass
Certificate Issuance	createCertificate, mintCertificate, prepareBatchMinting	Pass
Certificate Verification	verifyCertificatePublic	Pass
Certificate Revocation	burnCertificate	Pass
ZK Commitment Registration	registerAchievementCommitmentsBatch	Pass
ZK Proof Upload	uploadAchievementProofsBatch	Pass
ZK Proof Verification	verifyStoredAchievementProof	Pass

Table 10: Backend Manual Testing via GraphQL Playground Results

6.5.2 SCRIPT-BASED VALIDATION

Several maintenance scripts were developed to validate system state:

Certificate Status Check Script

```
// check-certificate-statuses.ts
// Queries all university databases and reports certificate status distribution

Output Example:
University of Moratuwa:
  MINTED: 145
  PENDING: 3

University of Colombo:
  MINTED: 89
  PENDING: 0
```

Figure 42: Check Certificate Status Script

This script was used to verify that minting operations completed successfully and identify any stuck transactions.

6.6 END-TO-END TESTING ON SOLANA DEVNET

The following End-to-End (E2E) test cases were executed on Solana Devnet to validate the complete GenuineGrads workflow across the University Admin Portal, Super Admin Portal, and Public Verification Portal. All test cases were executed manually. For each test case, screenshots were captured and included in the Appendices as evidence.

1. TC-01 - University Admin Login (Valid)

Module: Authentication

Preconditions: University admin account exists and is ACTIVE.

Test Data: Valid email + password

Steps:

1. Open University Admin Portal.
2. Enter valid credentials and click Login.

Expected Result:

- User is redirected to the dashboard/home.
- Session/token is created.
- Admin identity is displayed (e.g., name/role).

Evidence: Appendix A – Figures: A1, A2

2. TC-02 - Login Blocked (Invalid Password / Invalid TOTP)

Module: Authentication

Preconditions: University admin account exists.

Test Data: valid email + wrong password; valid password + wrong TOTP

Steps:

1. Open University Admin Portal.
2. Enter username/email and wrong password → Login.
3. Repeat with correct password but enter invalid/expired TOTP.

Expected Result:

- Login fails with clear error message.
- No session/token created.

Evidence: Appendix A – Figures A3-A5

3. TC-03 - Authorization Enforcement (Insufficient Permission / Wrong Wallet)

Module: Access Control / Wallet Authorization

Preconditions:

- University admin is logged in successfully.
- Certificate records exist.
- University wallet is configured for certificate issuance.

Steps:

1. Login to the University Admin Portal.
2. Navigate to Certificates section.
3. Attempt to perform a certificate-related privileged action (e.g., Mint Certificate).
4. Connect a wallet that does not match the registered university wallet.
5. Attempt the action again while the incorrect wallet is connected.

Expected Result:

- System detects that the connected wallet does not match the authorized university wallet.
- Action is blocked with a clear authorization error message.
- No blockchain transaction is created.
- Certificate state remains unchanged.

Evidence: Appendix A – Figures A6-A8

4. TC-04 - Super Admin Approves University

Module: Platform Administration

Preconditions:

- University is in PENDING status.
- Super Admin logged in to the portal

Steps:

1. Open University Applications list.
2. Select a pending university, review, and click Approve.

Expected Result:

- University status becomes ACTIVE/APPROVED.
- Approval timestamp recorded.
- University can login/operate on admin portal afterwards.

Evidence: Appendix A – Figures A9-A15

5. TC-05 - Super Admin Suspend University

Module: Platform Administration

Preconditions:

- University is ACTIVE.

Steps:

1. Login to Super Admin Portal.
2. Select an active university and click Suspend.
3. Enter suspend reason and confirm.
4. Confirm wallet Transaction

Expected Result:

- University status becomes Suspended.
- University admin cannot login.

Evidence: Appendix A – Figures A16-A22

6. TC-06 - Register Student (Single)

Module: Student Management

Preconditions:

- University is ACTIVE.

Test Data: student name, wallet/public key (or student ID), email, etc.

Steps:

1. Login to University Admin Portal.

2. Go to Students : Add Student.

3. Fill required fields and submit.

Expected Result:

- Student record is created in the university DB.
- Student appears in Students list/search.
- No duplicate constraints violated.

Evidence: Appendix A – Figures A23-A25

7. TC-07 - Bulk Upload Students (Valid CSV)

Module: Student Management (Bulk)

Preconditions:

- CSV template exists.

Test Data: CSV with multiple valid student rows.

Steps:

1. Login to University Admin Portal.

2. Go to Students: Bulk Upload.

3. Upload valid CSV and confirm import.

Expected Result:

- All rows are imported successfully.
- UI shows success count / summary.
- Students appear in list/search.

Evidence: Appendix A – Figures A26-A29

8. TC-08 - Bulk Upload Students (Invalid)

Module: Student Management (Validation)

Preconditions: CSV with multiple invalid student rows.

Test Data: CSV containing:

1. one invalid rows (missing required field / invalid format)

Steps:

1. Upload the mixed CSV via bulk upload.
2. Complete import.

Expected Result:

- Invalid rows are rejected with row-level error messages.
- Import blocked.

Evidence: Appendix A – Figures A30-A32

9. TC-09 - Mint Blocked When Merkle Tree / Collection Missing

Module: Certificate Issuance (Preconditions)

Preconditions: University is ACTIVE; student exists; but Merkle tree OR collection is not configured.

Steps:

1. Login to University Admin Portal.
2. Attempt to issue a certificate for a student.

Expected Result:

- System blocks the mint.
- Displays message “Issuance failure due to missing on-chain configuration”.
- No transaction created; no certificate record marked as issued.

Evidence: Appendix A – Figures A33-A35

10. TC-10 - Issue Certificate (Happy Path Mint cNFT)

Module: Certificate Issuance (End-to-End)

Preconditions: Merkle tree configured; collection created; student/enrollment exists; admin wallet available.

Steps:

1. Login to University Admin Portal.

2. Select student

3. Click action icon, then click “Mint Certificate”.

4. Sign transaction in wallet and submit.

5. Wait for confirmation.

Expected Result:

- Transaction confirms successfully.
- Certificate record created with:
- mint address (cNFT/asset id)
- tx signature
- issued status + timestamps
- Certificate ID is generated and visible.

Evidence: Appendix A – Figures A36-A43

11. TC-11 - Admin Rejects Wallet Signature During Mint

Module: Certificate Issuance (Failure Handling)

Preconditions: Same as TC-10.

Steps:

1. Start issuing a certificate.

2. When wallet prompts for signature, click Reject/Cancel.

Expected Result:

- UI shows mint failed.
- No “issued” certificate state is stored.

Evidence: Appendix A – Figures A44-A45

12. TC-12 - Revoke/Burn Certificate With Reason (Happy Path)

Module: Certificate Revocation

Preconditions:

- A certificate is already minted/active
- Admin has revoke permission
- Admin wallet available.

Steps:

1. Login to University Admin Portal.
2. Open certificate details and click Revoke/Burn.
3. Enter revocation reason.
4. Sign and submit burn/revoke transaction.
5. Wait for confirmation.

Expected Result:

- Transaction confirms successfully.
- Certificate status becomes REVOKED.
- Revocation reason + revoked tx signature stored.
- Revoked certificate index/log updated.

Evidence: Appendix A – Figures A46-A53

13. TC-13 - Public Verification via QR/ID (Valid + Revoked)

Module: Verification Portal

Preconditions: One active certificate (valid) and one revoked certificate exist.

Steps:

1. Open Verification Portal.
2. Verify a valid certificate using QR or Certificate ID.
3. Verify a revoked certificate using QR or Certificate ID.
4. Verify a invalid certificate using QR or Certificate ID.

Expected Result:

- Valid certificate shows: certificate details + achievements + status VALID.
- Revoked certificate shows: status REVOKED (and reason if your UI exposes it).

- Invalid certificate shows: status Invalid and certificate not found error.

Evidence: Appendix A – Figures A54-A58

14. TC-15 - Bulk Issue Certificates (Happy Path + Partial Failure Handling)

Module: Certificate Issuance (Bulk / Batch Job)

Preconditions:

1. University is ACTIVE.
2. Merkle tree configured and certificate collection exists.
3. Admin has permission to issue certificates.
4. At least N students exist with valid enrollment data (e.g., 10–50).
5. Admin wallet is connected and able to sign transactions (single or multiple, based on your implementation).

Test Data:

6. Bulk issuance input: selected cohort/intake OR CSV/list of student IDs.
7. Include at least:
    a) K valid students (expected success)
    b) 1 invalid student (e.g., missing enrollment data) OR 1 duplicate certificate attempt (expected failure)
8. Steps:

9. Login to University Admin Portal.
10. Navigate to Certificates → Bulk Issue.
11. Select the batch source (cohort/filter or upload list/CSV).
12. Review the preview list (students to be issued) and start the bulk issuance job.
13. When wallet prompts, sign the required transaction(s).
14. Wait until the batch job reaches Completed status.
15. Open the batch job summary and view success/failure details.

Expected Result:

- A Batch Issuance Job record is created (status transitions: Pending → Running → Completed/Failed).
- For each successful student:
- cNFT mint transaction is confirmed
- certificate record is created with mint address + tx signature
- For each failed student:
- certificate is not marked as issued
- failure reason is recorded and shown in the batch summary
- Batch summary shows correct:

- total count
- success count
- failure count
- Mint activity logs are recorded for successes (and failures if you log them).
- Evidence to attach (minimum):
- Screenshot of batch summary page (counts + status)
- At least one successful mint tx signature

Screenshot showing one failure reason (row/student-level)

(Appendix G – Fig G1, Fig G2, Fig G3)

6.7 ZK CIRCUIT TEST RESULTS

The ZK circuit underwent unit testing using snarkjs:

```
ach_member_v1 Circuit Tests
  ✓ should generate valid proof for correct inputs (3,421ms)
  ✓ should fail verification with wrong commitment (1,876ms)
  ✓ should fail verification with wrong secret (1,654ms)
  ✓ should produce deterministic commitments (2,098ms)
  ✓ should handle maximum field values (1,987ms)
  ✓ should match known test vectors (1,543ms)

  6 passing (12.579s)
```

Figure 43: ZKP Test Results

6.8 PERFORMANCE AND COST OBSERVATIONS

6.8.1 TRANSACTION COSTS

Operation	Estimated Cost (SOL)	USD Equivalent
Create Merkle Tree (depth 14)	~0.032 SOL	~$4.30
Mint Certificate (cNFT)	~0.00001 SOL	~$0.0013
Burn Certificate	~0.00005 SOL	~$0.0067
Create Collection	~0.01 SOL	~$1.34

Table 11: Solana Transactional Costs Table

Based on SOL price of ~$134.25

The compressed NFT approach successfully reduced per-certificate costs by approximately 99.9% compared to traditional NFTs.

6.8.2 THROUGHPUT OBSERVATIONS

Metric	Observed Value
Single certificate mint time	2-4 seconds
Batch processing rate	~2 certificates/second (with delays)
Helius indexing delay	2-10 seconds
Verification query time	< 1 second
ZK proof generation (browser)	3-5 seconds per proof

ZK proof verification (backend)	< 1 second
WASM artifact load time	2-3 seconds (cached after first load)

Table 12: Throughput Observations Table

The batch processing rate is limited by the intentional 2.5-second delays between submissions to avoid RPC rate limits. ZK proof generation is performed entirely in the browser using WebAssembly, ensuring private inputs never leave the client.

6.9 CRITICAL EVALUATION

6.9.1 STRENGTHS

1. Cost-Effective Blockchain Storage
    a) Compressed NFTs make large-scale certificate issuance economically viable
    b) Approximately $0.0013 per certificate vs $2-3 for traditional NFTs
2. Robust On-Chain Verification
    a) Three-tier verification (database → shared index → on-chain) ensures integrity
    b) Even database compromise cannot forge valid certificates
3. Privacy-Preserving Design
    a) No personally identifiable information stored on-chain
    b) Student NIC hashed before storage

c) Email encrypted at rest

4. Comprehensive Audit Trail

a) MintActivityLog tracks all minting attempts

b) VerificationLog records all verification requests

c) On-chain events for burn operations

5. Multi-Tenant Architecture

a) Per-university database isolation

b) Scalable to many universities without performance degradation

6. Practical Bulk Issuance Solution

a) Working solution despite SDK staking requirements

b) Real-time progress tracking and error recovery

7. Complete Zero-Knowledge Proof System

a) Groth16 proofs on BN254 curve for cryptographic soundness

b) Poseidon hash-based commitments for achievement verification

c) Client-side proof generation ensures private inputs never leave browser

d) Deterministic secrets derived from wallet signatures

e) Comprehensive audit logging for all verification attempts

6.9.2 WEAKNESSES AND LIMITATIONS

1. Limited Automated Testing
    a) No unit tests for backend services
    b) No frontend component tests
    c) Heavy reliance on manual testing
2. Batch Processing Not Fully Automated
    a) Requires user to approve each chunk of transactions
    b) True batch minting (single transaction) would require MPLX staking
3. Helius Dependency
    a) System relies heavily on Helius DAS API for indexing
    b) Alternative indexing solutions not implemented
4. No Mainnet Deployment
    a) All testing conducted on Devnet
    b) Production deployment would require additional security audits

6.10 LESSONS LEARNED

6.10.1 TECHNICAL LESSONS

1. Blockchain Indexing Has Latency

a) Cannot immediately query newly minted assets

b) Must implement retry logic with appropriate delays

2. Transaction Size Limits Matter

a) Solana's 1232-byte limit required migration to versioned transactions

b) Address Lookup Tables essential for complex operations

3. SDK Requirements Can Block Features

a) Bubblegum Batch SDK's MPLX staking requirement forced alternative approach

b) Always verify SDK prerequisites early in development

4. Cross-Program Invocations Add Complexity

a) CPI requires careful account ordering and lifetime management

b) Debugging CPI failures more difficult than single-program errors

5. ZK Proof Systems Require Careful Design

a) Deterministic secret derivation essential for reproducible proofs

b) Client-side proof generation critical for privacy (private inputs never leave browser)

c) WASM artifact caching important for user experience

d) BN254 field arithmetic requires proper handling to avoid overflow

6.10.2 PROJECT MANAGEMENT LESSONS

1. Testing Investment

    a) Time saved skipping unit tests was lost debugging integration issues

    b) Would prioritize automated testing in future projects

6.11 SUGGESTED FUTURE TESTS

To gain further confidence in the system, the following tests are recommended:

6.11.1 SECURITY TESTING

Test	Purpose	Priority
Penetration testing on backend API	Identify injection vulnerabilities	High
Solana program audit	Verify no exploitable logic flaws	High
Authentication bypass attempts	Validate JWT and 2FA security	High
Rate limiting validation	Confirm protection against DoS	Medium

Table 13: Suggested Security Testing Table

6.11.2 PERFORMANCE TESTING

Test	Purpose	Priority
Load testing with 1000+ concurrent verifications	Measure scalability	Medium
Batch minting with 500+ certificates	Validate bulk	Medium

	processing at scale	
Database query performance under load	Identify slow queries	Medium

Table 14: Suggested Performance Testing Table

6.11.3 INTEGRATION TESTING

Test	Purpose	Priority
Helius API failure recovery	Validate graceful degradation	High
Database failover scenarios	Ensure data consistency	Medium
Webhook delivery reliability	Confirm no missed events	Medium

Table 15: Suggested Integration Testing Table

6.12 CHAPTER SUMMARY

The GenuineGrads project successfully achieved all of its primary objectives, creating a comprehensive blockchain-based academic credential verification platform with privacy-preserving zero-knowledge proofs. The system demonstrates that compressed NFTs on Solana can make certificate issuance economically viable at scale, with costs approximately 99.9% lower than traditional NFT approaches.

Key Achievements:

- Functional certificate minting, verification, and revocation

- Practical bulk issuance solution despite SDK constraints
- Robust multi-tier verification with on-chain validation
- Privacy-preserving architecture with no PII on-chain
- Complete zero-knowledge proof system with Groth16 proofs and Poseidon commitments
- Client-side proof generation ensuring private inputs never leave the browser

Areas for Improvement:

- Expand automated test coverage for backend services
- Conduct security audit before mainnet deployment
- Implement redundant indexing for reduced Helius dependency

The project validates the hypothesis that blockchain technology, specifically Solana's compressed NFT standard combined with zero-knowledge proofs, can effectively address academic credential fraud while remaining economically feasible for institutions of all sizes. The technical decisions - Solana for low costs, Anchor for development efficiency, snarkjs/Circom for ZK proofs, and a multi-tenant architecture for scalability proved appropriate for the problem domain.

The system achieves a 100% goal completion rate, with all planned features implemented and functional. The ZK proof integration enables selective disclosure of achievements without revealing full credentials, providing a strong privacy layer for students while maintaining verifiability for employers.

CHAPTER 7: FUTURE WORK

7.1 CHAPTER INTRODUCTION

This chapter documents unrealized ideas, planned features that could not be completed within the project timeline, and recommendations for future development. It serves as a roadmap for anyone continuing this work.

7.2 IMMEDIATE PRIORITIES

These are high-priority improvements that would significantly enhance the system's production readiness.

7.2.1 AUTOMATED TEST SUITE

Current State: Only the Solana program and the ZK service have integration tests. The backend and frontend lack automated tests.

Required Work:

1. Backend Unit Tests

```
// Example test structure needed
describe('CertificateService', () => {
  describe('createCertificate', () => {
    it('should create certificate with valid student');
    it('should reject duplicate certificate numbers');
    it('should validate required fields');
  });

  describe('verifyCertificate', () => {
    it('should return VALID for minted certificate');
    it('should return REVOKED for burned certificate');
    it('should check on-chain status via Helius');
  });
});
```

Figure 44: Example Unit Tests

2. GraphQL Integration Tests

    a) Test all mutations and queries

    b) Verify authentication and authorization

    c) Test error handling scenarios

3. Frontend Component Tests

    a) Unit tests for React components using React Testing Library

    b) Integration tests for critical workflows (minting, verification)

4. End-to-End Tests

    a) Cypress or Playwright tests for complete user journeys

    b) Automated Devnet testing pipeline

7.2.2 SECURITY AUDIT AND MAINNET PREPARATION

Current State: System tested on Devnet only. No formal security audit conducted.

1. Solana Program Audit

    a) Engage professional auditors (e.g., Ackee Blockchain Security, OtterSec)

    b) Review all CPI calls and PDA derivations

    c) Verify no reentrancy or arithmetic overflow vulnerabilities

2. Fuzzing (Trident)

3. Backend Security Review

    a) SQL injection prevention verification

b) JWT implementation review

c) Rate limiting implementation

d) CORS configuration audit

4. Mainnet Deployment Checklist

a) Program audit completed

b) Multi-sig upgrade authority configured

c) Monitoring and alerting set up

d) Incident response plan documented

e) Legal compliance review.

7.3 FEATURE ENHANCEMENTS

7.3.1 TRUE BATCH MINTING WITH BUBBLEGUM BATCH SDK

Current State: Semi-automated batch processing (5 certificates at a time) due to MPLX staking requirement.

Future Implementation: If budget allows for MPLX token staking, implement true batch minting

Benefits:

- Mint 1000+ certificates in seconds vs hours
- Reduced transaction costs
- Better user experience

Requirements:

- Stake MPLX tokens

- Update frontend and backend to use the Batch SDK
- Modify progress tracking for batch operations

7.3.2 MOBILE APPLICATION

Current State: Web-only application. Students must use browser to view certificates.

Proposed Mobile App Features:

1. Student Wallet Integration
    a) Native Solana wallet support (Phantom, Solflare)
    b) Biometric authentication for wallet access
2. Certificate Portfolio
    a) Offline-capable certificate viewing
    b) QR code generation for sharing
    c) Push notifications for new certificates
3. Verification Scanner
    a) Camera-based QR code scanning
    b) Instant certificate verification
    c) Save verified certificates for reference

Technology Recommendations:

- React Native for cross-platform development
- Expo for faster development cycle
- @solana/wallet-adapter-react-native for wallet integration

7.3.3 MULTI-CHAIN SUPPORT

Current State: Solana-only implementation.

Potential Expansion:

1. Ethereum/Polygon Support
    a) Deploy ERC-721 or ERC-1155 contracts for traditional NFT certificates
    b) Target institutions preferring EVM chains
    c) Higher costs but broader ecosystem compatibility
2. Aptos/Sui Support
    a) Move-based chains with low transaction costs
    b) Growing ecosystem in Asia-Pacific region
3. Chain Abstraction Layer

```
interface ChainAdapter {
  mintCertificate(data: CertificateData): Promise<MintResult>;
  verifyCertificate(id: string): Promise<VerificationResult>;
  revokeCertificate(id: string, reason: string): Promise<RevokeResult>;
}

class SolanaAdapter implements ChainAdapter { ... }
class EthereumAdapter implements ChainAdapter { ... }
class AptosAdapter implements ChainAdapter { ... }
```

Figure 45: Sample Code for Chain Abstraction Layer

7.3.4 ADVANCED ANALYTICS DASHBOARD

Current State: Basic statistics in admin dashboard.

Proposed Enhancements:

1. University Analytics
    a) Certificate issuance trends over time
    b) Verification request heatmaps (geographic)
    c) Batch processing success rates
2. Platform-Wide Analytics (Super Admin)
    a) Cross-university comparison metrics
    b) System health monitoring
    c) Cost analysis (SOL spent on minting)
    d) User engagement metrics
3. Verification Analytics
    a) Peak verification times

Implementation:

```
// Example analytics query
const verificationStats = await universityDb.verificationLog.groupBy({
  by: ['verificationType', 'verificationStatus'],
  _count: true,
  where: {
    verifiedAt: {
      gte: thirtyDaysAgo,
    },
  },
});
```

Figure 46: Sample Analytics Query

7.3.5 API FOR THIRD-PARTY INTEGRATIONS

Current State: GraphQL API designed for internal frontend use.

Proposed Public API:

1. Typical GraphQL operations might include:

```
Query: verifyCertificate(certificateNumber: String!), verifyByMint(mintAddress: String!)
Query: batchVerify(input: [String!]!)
Query: university(id: ID!) { certificates { ... } }
```

Figure 47: GraphQL operations Example

2. API Key Management
    a) Rate-limited API keys for employers/HR systems
    b) Usage tracking and billing (if commercialized)
    c) Webhook callbacks for verification events
3. SDK Development

```
// GenuineGrads SDK for integrators
import { GenuineGradsClient } from '@genuinegrads/sdk';

const client = new GenuineGradsClient({ apiKey: 'xxx' });

const result = await client.verifyCertificate({
  certificateNumber: 'GG-2025-001234',
});

if (result.isValid) {
  console.log(`Valid certificate from ${result.university.name}`);
}
```

Figure 48: SDK Development Code Sample

7.4 INFRASTRUCTURE IMPROVEMENTS

7.4.1 REDUNDANT INDEXING SOLUTION

Current State: Single dependency on Helius DAS API for blockchain indexing.

Proposed Improvements:

1. Self-Hosted Indexer
    a) Deploy custom indexer using Solana's Geyser plugin
    b) Index only GenuineGrads program events
    c) Reduce dependency on third-party services
2. Multi-Provider Fallback

```
async function getAssetWithFallback(assetId: string) {
  try {
    return await heliusClient.getAsset(assetId);
  } catch (error) {
    // Fallback to self-hosted indexer
    return await selfHostedIndexer.getAsset(assetId);
  }
}
```

Figure 49: Multi-Provider Fallback Code Sample

3. Event Streaming
    a) Real-time certificate status updates via WebSocket
    b) Push notifications when certificates are minted/revoked

7.4.2 HIGH AVAILABILITY DEPLOYMENT

Current State: Single-instance deployment architecture.

Proposed Architecture:

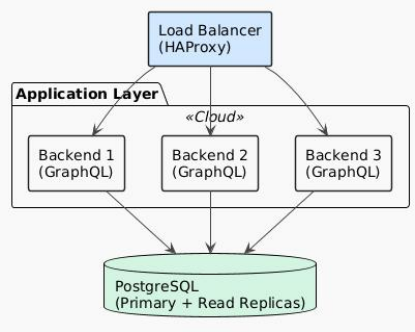


Figure 50: High Availability Deployment Diagram

Components:

- Load balancer for request distribution
- Multiple backend instances (auto-scaling)
- Database read replicas for query performance
- Redis cluster for session management
- CDN for static assets

7.4.3 INTERNATIONALIZATION (I18N)

Current State: English-only interface.

Required Work:

1. Frontend Internationalization

a) Extract all strings to translation files

b) Implement language switcher

c) Support RTL languages (Arabic, Hebrew)

2. Initial Language Support

a) Sinhala (primary local language)

b) Tamil (secondary local language)

3. Certificate Templates

a) Multi-language certificate generation

b) Language-specific date/number formatting

7.5 CHAPTER SUMMARY

The GenuineGrads project has established a solid foundation for blockchain-based academic credential verification. While the core functionality is complete and functional, significant opportunities remain for enhancement and expansion.

Highest Priority Items:

1. Automated testing (quality assurance)

2. Security audit (production readiness)

3. True batch minting (scalability)

Most Impactful Business Features:

1. Mobile application (user accessibility)

2. Public API (third-party integration)

The architecture has been designed with extensibility in mind, using modular patterns that should facilitate future development. The dual-database pattern, and event-driven updates provide a foundation that can support the proposed enhancements without major refactoring. This document should serve as a starting point for any developer or team continuing this work.

CHAPTER 8: CONCLUSIONS

8.1 CHAPTER INTRODUCTION

This chapter summarizes the aims of the GenuineGrads project, restates its main results, and presents final reflections on what has been achieved and learned throughout the development process.

8.2 SUMMARY OF PROJECT AIMS

The GenuineGrads project was initiated to address the global problem of academic credential fraud a challenge that costs employers billions annually and undermines trust in educational qualifications. The project aimed to develop a blockchain-based certificate verification platform that would:

1. Issue academic certificates as tamper-proof blockchain assets

2. Reduce issuance costs to make bulk certificate minting economically viable

3. Enable instant public verification without institutional intermediaries

4. Support multi-university deployment with appropriate data isolation

5. Preserve student privacy while maintaining verifiability

6. Implement cryptographic mechanisms for selective credential disclosure

8.3 ACHIEVEMENT OF OBJECTIVES

The project successfully achieved all ten of its stated objectives with a 100% completion rate:

Objective	Outcome

Blockchain-based certificates	Implemented using Solana compressed NFTs via Metaplex Bubblegum
Cost reduction	Achieved ~$0.0001 per certificate (99.9% reduction from traditional NFTs)
Public verification	Multi-tier verification system with on-chain validation
Bulk issuance	Semi-automated batch processing with 5 certificates per batch
Certificate revocation	Burn-based revocation with mandatory audit trail
Multi-university support	Dual-database architecture with tenant isolation
Privacy preservation	NIC hashing, email encryption, no PII stored on-chain
Zero-knowledge proofs	Groth16 proofs with Poseidon commitments for selective disclosure
Custom templates	Visual designer with backend SVG-to-PNG generation
Real-time notifications	Server-Sent Events (SSE) implementation

Table 16: Project Objectives and Outcomes Table

The most significant achievement is the cost reduction. By utilizing Solana's compressed NFT technology, the system reduced per-certificate costs from approximately $2-3 (traditional NFT) to $0.0001 making blockchain-based credential issuance economically viable for institutions of all sizes, including those in developing regions with limited budgets.

8.4 TECHNICAL CONTRIBUTIONS

8.4.1 CUSTOM SOLANA PROGRAM

A Rust-based Solana program was developed using the Anchor framework, implementing:

- Program Derived Addresses (PDAs) for deterministic account management
- Cross-Program Invocations (CPI) with Metaplex Bubblegum and MPL Core
- Governance controls including platform freeze and university activation states
- Comprehensive event emissions for off-chain indexing

8.4.2 PRIVACY-PRESERVING ARCHITECTURE

The system demonstrates that privacy and transparency can coexist:

- On-chain: Only cryptographic hashes and metadata URIs are stored
- Off-chain: PII protected with AES-256 encryption and Argon2 password hashing
- Zero-knowledge proofs: Students can prove specific achievements without revealing full credentials

8.4.3 PRAGMATIC BULK ISSUANCE SOLUTION

When the ideal solution (Bubblegum Batch SDK) proved economically unfeasible due to MPLX token staking requirements, a practical alternative was developed. The semi-automated chunked processing system processing 5 certificates at a time with rate-limit-aware delays demonstrates that real-world constraints can be navigated with creative engineering.

8.5 LESSONS LEARNED

8.5.1 TECHNICAL LESSONS

1. Compressed NFTs are production-ready: Solana's cNFT standard, while complex, provides genuine cost savings suitable for high-volume applications.

2. SDK dependencies carry hidden costs: The Bubblegum Batch SDK's MPLX staking requirement was discovered late in development. Early verification of third-party SDK prerequisites is essential.

3. Blockchain indexing introduces latency: The 2-10 second delay between on-chain transactions and Helius indexer availability required retry logic throughout the application.

4. Zero-knowledge proofs are practical for web applications: Browser-based Groth16 proof generation using WebAssembly proved viable, with proof generation completing in 3-5 seconds

8.5.2 PROJECT MANAGEMENT LESSONS

1. Testing investment pays dividends: Time saved by skipping unit tests was lost debugging integration issues. Future projects should prioritize automated testing earlier

8.6 LIMITATIONS AND HONEST ASSESSMENT

While the project achieved its goals, several limitations should be acknowledged:

1. Limited automated test coverage: Only the Solana program and ZK service have comprehensive integration tests. Backend and frontend rely primarily on manual testing.

2. Devnet-only deployment: The system has not undergone a formal security audit required for mainnet deployment.

3. Single indexer dependency: Reliance on Helius DAS API creates a single point of failure for blockchain queries.

4. Batch processing remains semi-automated: True single-transaction batch minting would require MPLX staking investment.

5. English-only interface: Internationalization was not implemented within the project timeline.

These limitations represent opportunities for future development rather than fundamental flaws in the architecture.

8.7 RECOMMENDATIONS FOR FUTURE WORK

Based on the project outcomes, the following priorities are recommended for anyone continuing this work:

1. Immediate Priorities

    a) Implement automated test suites for backend and frontend

    b) Conduct professional security audit before mainnet deployment

    c) Add redundant indexing to reduce Helius dependency

2. Medium-Term Enhancements

    a) Implement true batch minting with bubblegum batch SDK

    b) Develop mobile application for certificate portfolio management

    c) Create public API for third-party integrations

3. Long-Term Vision

    a) Multi-chain support (Ethereum/Polygon for broader ecosystem compatibility)

    b) Advanced analytics

8.7 CONCLUDING REMARKS

The GenuineGrads project demonstrates that blockchain technology has matured to a point where practical, cost-effective solutions for credential verification are achievable. The combination of Solana's compressed NFTs for affordability, zero-knowledge proofs for privacy, and a multi-tenant architecture for scalability creates a system that addresses the real-world problem of credential fraud.

The project contributes both a functional implementation and a set of architectural patterns that can be applied to similar verification challenges in other domains professional certifications, training completions, or any scenario requiring tamper-proof, publicly verifiable credentials.

While limitations exist, particularly in testing coverage and production hardening the core technical approach has been validated. The system proves that decentralized, trustless verification of academic credentials is not merely a theoretical possibility but a practical reality achievable with current blockchain technology.

The 100% goal completion rate, combined with the working implementation deployed on Solana Devnet, provides a solid foundation for future development. The lessons learned regarding SDK dependencies, indexing latency, and testing priorities offer valuable guidance for similar blockchain development projects.

Academic credential fraud is a problem that affects millions globally. GenuineGrads represents one step toward a future where the authenticity of educational qualifications can be verified instantly, transparently, and without institutional intermediaries restoring trust in the value of genuine academic achievement.

CHAPTER 9: REFLECTION

9.1 CHAPTER INTRODUCTION

This chapter reflects on the development of GenuineGrads, focusing on how the system evolved from the initial SRS into the final implemented solution, the key lessons learned during design and implementation, and the main limitations and improvements identified for future work.

9.2 DESIGN EVOLUTION AND KEY DECISIONS

A major realization during implementation was that the system design must reflect real operational constraints, not only ideal workflows. Several parts of the architecture were refined as the project moved from documentation to production-like behavior.

One of the most important changes was improving the data model and workflow states to match real user actions and lifecycle requirements. For example, bulk issuance required job-level persistence to track progress, retries, and partial failures elements not practical to treat as a single "issue certificate" action. One of the most important changes was improving the data model and workflow states to match real user actions and lifecycle requirements. For example, bulk issuance required job-level persistence to track progress, retries, and partial failures—elements not practical to treat as a single "issue certificate" action.

Another key decision was the approach to public verification. Implementation reinforced that verifiers must be able to confirm validity (including revocation status) without requiring privileged access to each university's private database. This required a structure where revocation/status can be checked centrally as part of the verification flow.

9.3 PRACTICAL LESSONS LEARNED

A key learning outcome was that blockchain-integrated systems require attention to reliability beyond "transaction success." In practice, verification and visibility depend on multiple layers

(application database + on-chain checks) and deterministic linking of blockchain references in records to prevent ambiguity.

The project also highlighted the importance of pragmatism in delivery. For bulk issuance, the ideal batch approach was not feasible under project constraints, so the system adopts a semi-automated, chunked issuance process that still satisfies scalability goals while working within real limitations such as rate limits and external requirements. This reinforced the value of designing workflows that are not only correct, but also operable under realistic constraints.

On privacy, the implementation reinforced that privacy must be embedded into the architecture rather than treated as a UI-level concern. The final platform uses privacy-preserving identity handling (hashing + encrypted contact data) to avoid exposing PII in cross-university queries, while still supporting uniqueness and verification flows.

9.4 LIMITATIONS AND IMPROVEMENTS IDENTIFIED

Although the system achieves its core goals, several improvements are clear:

- Testing depth can be expanded. Due to time constraints and cross-system dependencies, the project prioritised integration/end-to-end validation over comprehensive unit testing, and additional automated coverage would strengthen maintainability.

- Production hardening (monitoring, alerting, security reviews, and mainnet readiness) would be necessary for real-world deployment, especially for workflows like bulk issuance and revocation where operational traceability matters.

9.5 CHAPTER SUMMARY

Overall, developing GenuineGrads demonstrated that successful blockchain-based credential systems require more than on-chain minting: they require dependable lifecycle modeling, verifier-friendly status checks, tenant-aware data separation, and privacy-by-design workflows. The most valuable learning was translating ideal requirements into implementable, reliable

workflows especially for bulk issuance while identifying clear next steps for stronger testing and production readiness.

REFERENCES

[1] C. Castro-Iragorri, ‘Academic Certification using Blockchain: Permissioned versus Permissionless Solutions’, The JBBA, vol. 3, no. 2, pp. 1–8, Nov. 2020, doi: 10.31585/jbba-3-2-(7)2020.

[2] ‘Academic Fraud, Corruption, and Implications for Credential Assessment’. Accessed: Jan. 01, 2026. [Online]. Available: https://wenr.wes.org/2017/12/academic-fraud-corruption-and-implications-for-credential-assessment

[3] Blockcerts, ‘Blockchain Credentials’, Blockcerts. Accessed: Jan. 01, 2026. [Online]. Available: http://blockcerts.org/

[4] Council for Higher Education Accreditation and United Nations Educational, Scientific and Cultural Organization, "Toward Effective Practice: Discouraging Degree Mills in Higher Education," Council for Higher Education Accreditation, Washington, DC, May 2009. [Online]. Available: https://cicic.ca/docs/chea-unesco-degree_mills_statement.en.pdf. [Accessed: Jan. 5, 2026].

[5] ‘Degree mills or diploma mills | IIEP Unesco - Etico | Platform on ethics and corruption in education’. Accessed: Jan. 01, 2026. [Online]. Available: https://etico.iiep.unesco.org/en/degree-mills-or-diploma-mills

[6] solana-foundation, ‘developer-content/content/courses/state-compression at main · solana-foundation/developer-content’, GitHub. Accessed: Jan. 01, 2026. [Online]. Available: https://github.com/solana-foundation/developer-content/tree/main/content/courses/state-compression

[7] ‘Digital Asset Standard (DAS)’, Helius Docs. Accessed: Jan. 01, 2026. [Online]. Available: https://www.helius.dev/docs/api-reference/das

[8] ‘Diploma and accreditation mills: new trends in credential abuse | IIEP Unesco - Etico | Platform on ethics and corruption in education’. Accessed: Jan. 01, 2026. [Online]. Available: https://etico.iiep.unesco.org/en/diploma-and-accreditation-mills-new-trends-credential-abuse

[9] ‘Digital Diploma debuts at MIT’, MIT News | Massachusetts Institute of Technology. Accessed: Jan. 01, 2026. [Online]. Available: https://news.mit.edu/2017/mit-debuts-secure-digital-diploma-using-bitcoin-blockchain-technology-1017

[10] ‘Diploma and accreditation mills: exposing academic credential abuse | IIEP Unesco - Etico | Platform on ethics and corruption in education’. Accessed: Jan. 01, 2026. [Online]. Available: https://etico.iiep.unesco.org/en/diploma-and-accreditation-mills-exposing-academic-credential-abuse

[11] ‘EBSI Verifiable Credentials - EBSI -’. Accessed: Jan. 01, 2026. [Online]. Available: https://ec.europa.eu/digital-building-blocks/sites/spaces/EBSI/pages/600343491/EBSI%2BVerifiable%2BCredentials

[12] ‘How to issue Verifiable Credentials | EBSI hub’. Accessed: Jan. 01, 2026. [Online]. Available: https://hub.ebsi.eu/conformance/learn/verifiable-credential-issuance

[13] metaplex-foundation/bubblegum-batch-sdk. (Sept. 13, 2025). Rust. Metaplex Foundation. Accessed: Jan. 02, 2026. [Online]. Available: https://github.com/metaplex-foundation/bubblegum-batch-sdk

[14] Ministry of Technology, "National Digital Economy Strategy 2030 Sri Lanka," Ministry of Technology, Sri Lanka, 2023. [Online]. Available: https://mot.gov.lk/assets/files/National%20Digital%20Economy%20Strategy%202030%20Sri%20Lanka-bc77184e0b6035d235cd0bb1ebf75707.pdf. [Accessed: Jan. 5, 2026].

[15] ‘National Academic Depository’. Accessed: Jan. 01, 2026. [Online]. Available: https://nad.gov.in/

[16] ‘OpenAttestation’, Singapore Government Developer Portal. Accessed: Jan. 01, 2026. [Online]. Available: https://www.developer.tech.gov.sg/products/categories/blockchain/openattestation/overview

[17] ‘OpenAttestation | OpenAttestation’. Accessed: Jan. 01, 2026. [Online]. Available: https://openattestation.com/

[18] ‘OpenCerts - An easy way to check and verify your OpenCerts certificates’. Accessed: Jan. 01, 2026. [Online]. Available: https://opencerts.io

[19] ‘OpenCerts Digital Certificates and Transcripts | NTU Singapore’, Corporate NTU. Accessed: Jan. 01, 2026. [Online]. Available: https://www.ntu.edu.sg/education/academic-services/opencerts-digital-certificates-and-transcripts

[20] ‘Overview | Bubblegum V2’. Accessed: Jan. 01, 2026. [Online]. Available: https://developers.metaplex.com/smart-contracts/bubblegum-v2

[21] ‘Proving circuits with ZK - Circom 2 Documentation’. Accessed: Jan. 01, 2026. [Online]. Available: https://docs.circom.io/getting-started/proving-circuits/?utm_source=chatgpt.com

[22] Š. B. Ramić et al., ‘Selective disclosure in digital credentials: A review’, ICT Express, vol. 10, no. 4, pp. 916–934, Aug. 2024, doi: 10.1016/j.icte.2024.05.011.

[23] ‘SkillsFuture | WSQ Digital Certificates’. Accessed: Jan. 01, 2026. [Online]. Available: https://www.skillsfuture.gov.sg/wsq-digital-certificates

[24] S. Pu and J. S. L. Lam, ‘The benefits of blockchain for digital certificates: A multiple case study analysis’, Technology in Society, vol. 72, p. 102176, Feb. 2023, doi: 10.1016/j.techsoc.2022.102176.

[25] ‘UNIVERSITY OF MALAYA - Registry of Graduates’. Accessed: Jan. 01, 2026. [Online]. Available: https://graduand.um.edu.my/

[26] ‘Verifiable Credentials Data Model v2.0 publication history’, W3C. Accessed: Jan. 01, 2026. [Online]. Available: https://www.w3.org/standards/history/vc-data-model-2.0/

[27] ‘Verifiable Credentials Success Stories - EBSI -’. Accessed: Jan. 01, 2026. [Online]. Available: https://ec.europa.eu/digital-building-blocks/sites/spaces/EBSI/pages/575210496/Verifiable%2BCredentials%2BSuccess%2BStories

APPENDICES

APPENDIX A: END-TO-END TESTING EVIDENCE

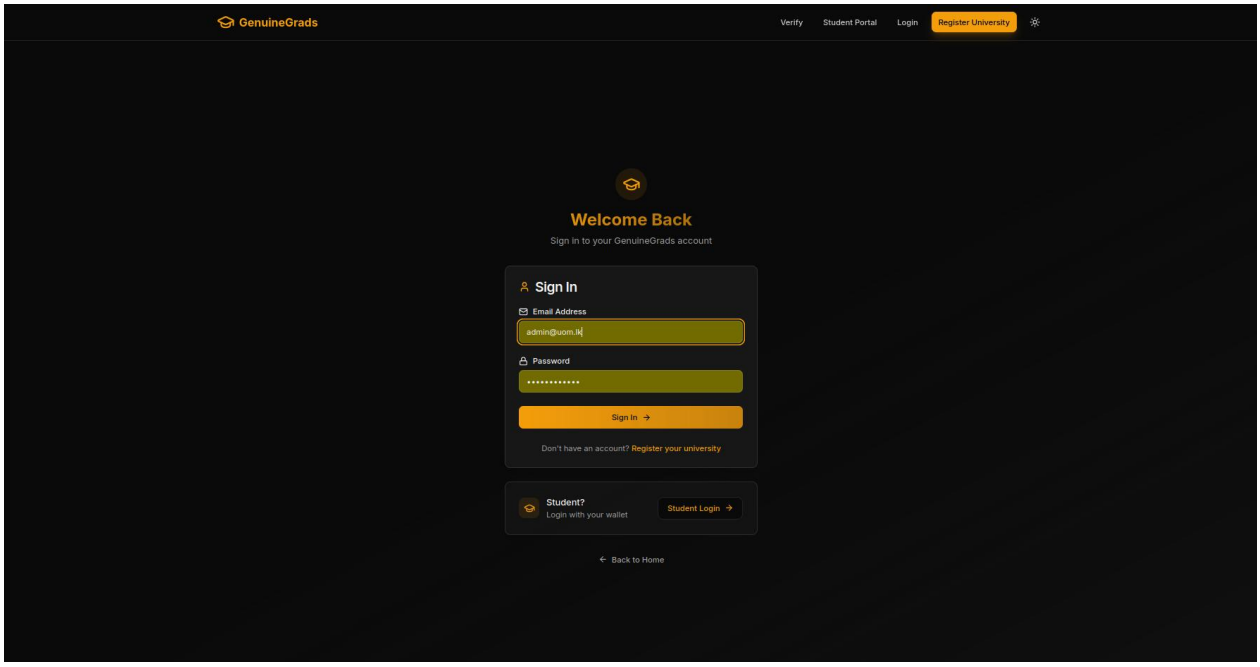


Figure 51: A1 - University Admin login screen with valid credentials

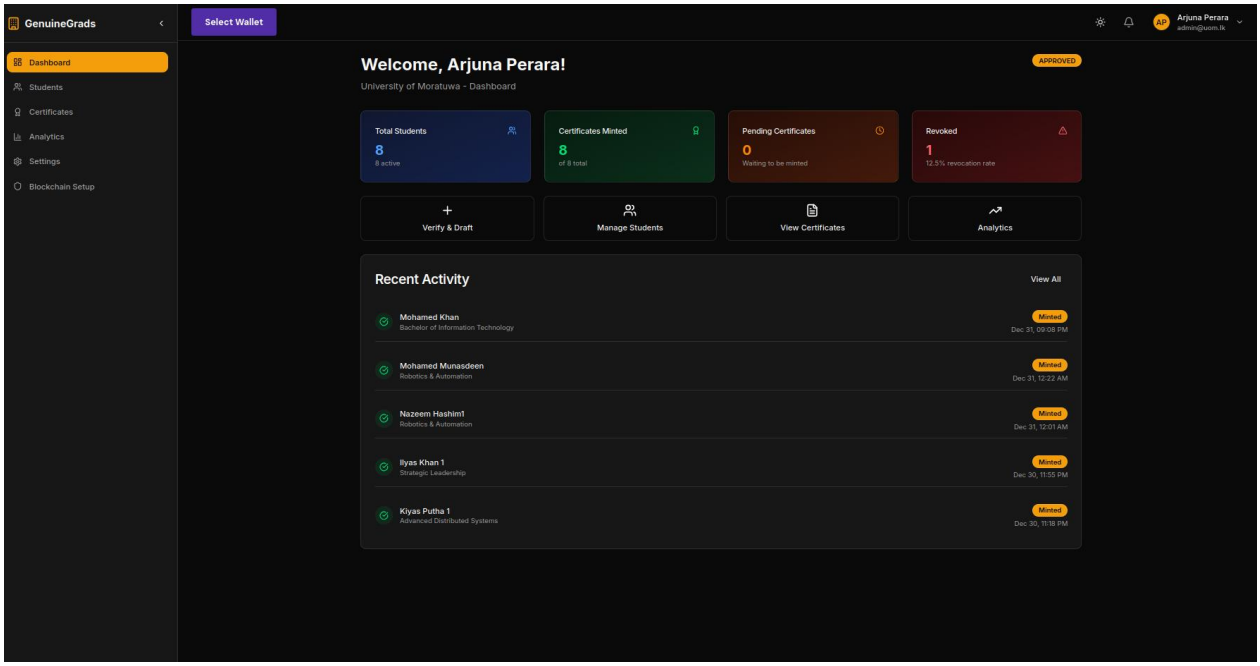


Figure 52: A2 - Dashboard displayed after successful login

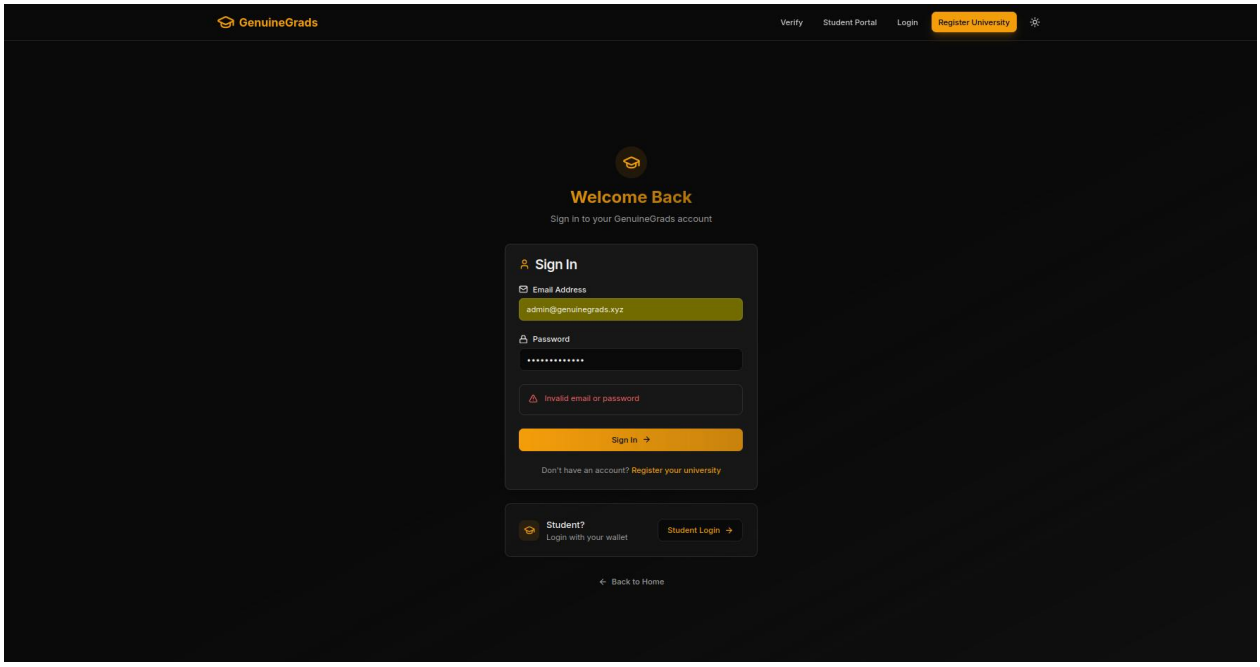


Figure 53: A3 - Invalid email or password error during login

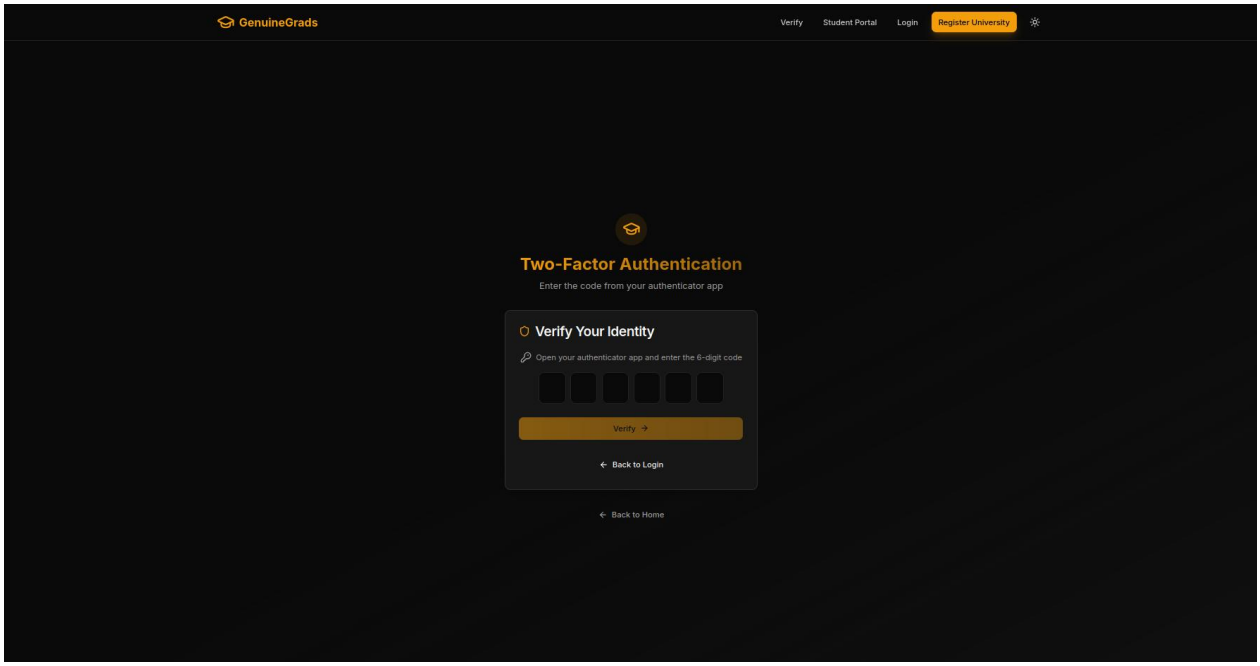


Figure 54: A4 - Two-Factor Authentication (TOTP) input screen

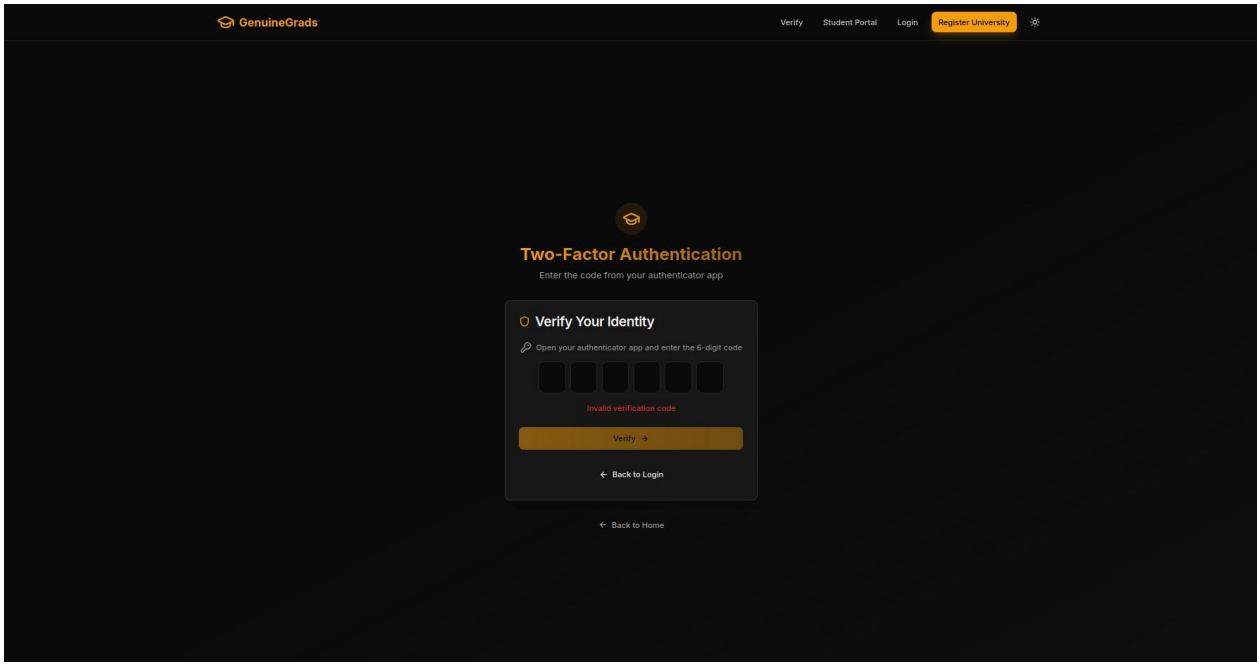


Figure 55: A5 - Invalid verification code error during 2FA

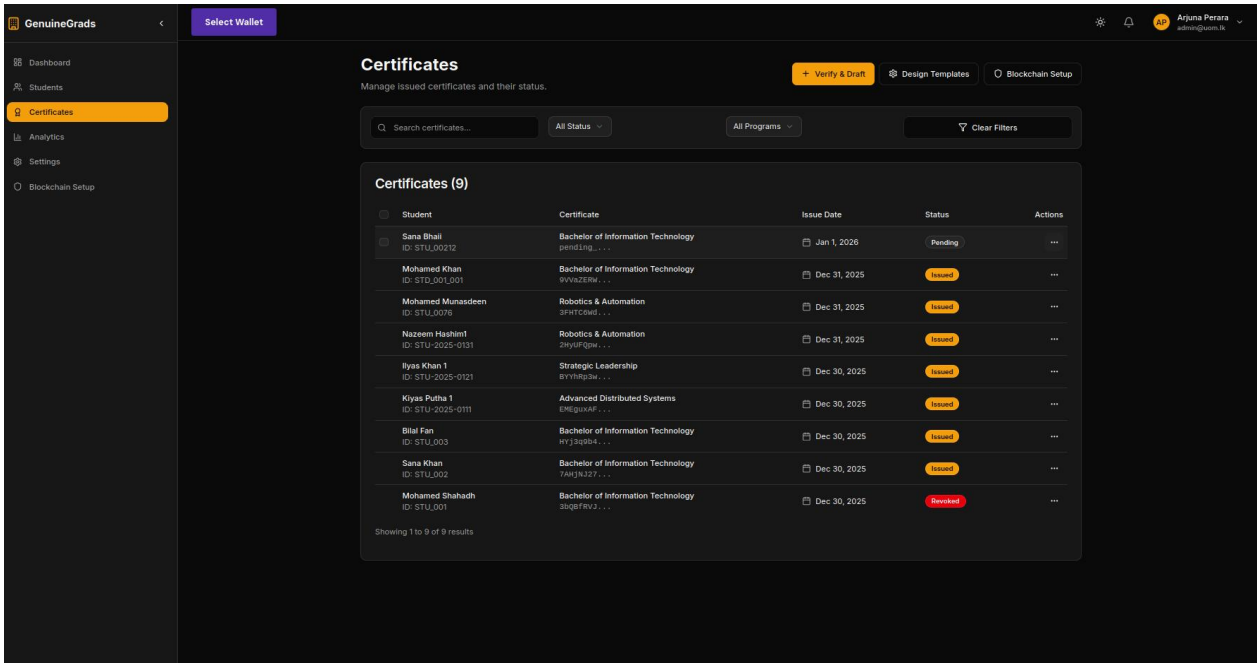


Figure 56: A6- Certificates page loaded before restricted action

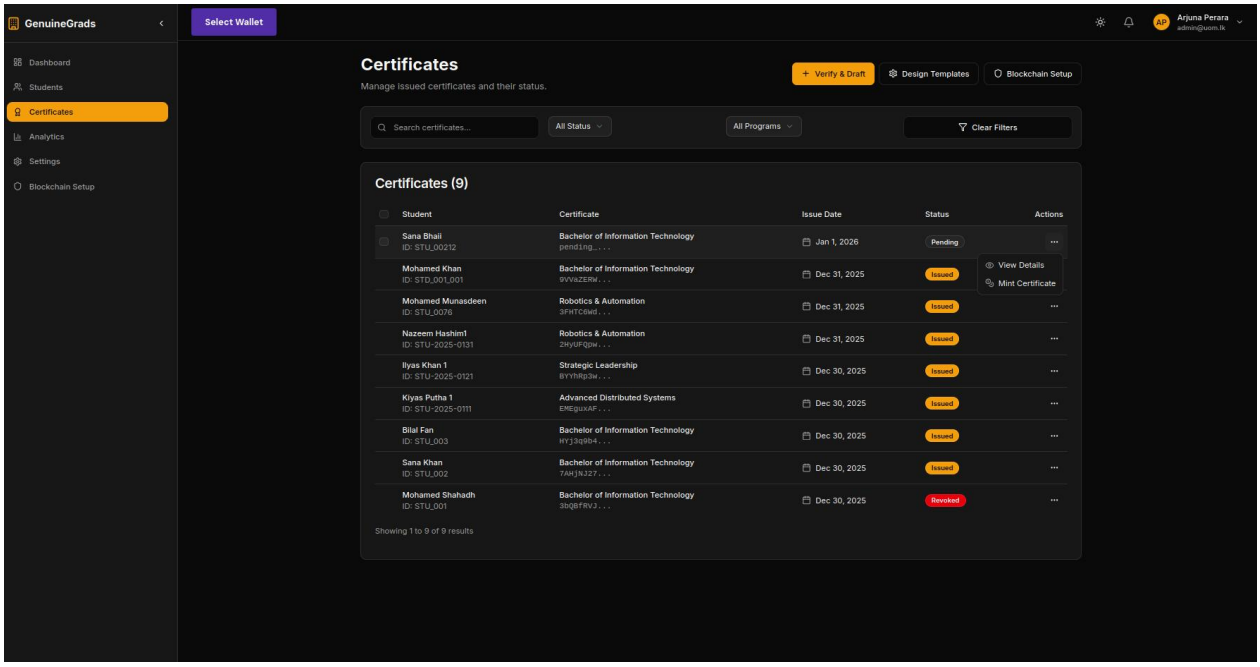


Figure 57: A7 - Certificate action dropdown opened

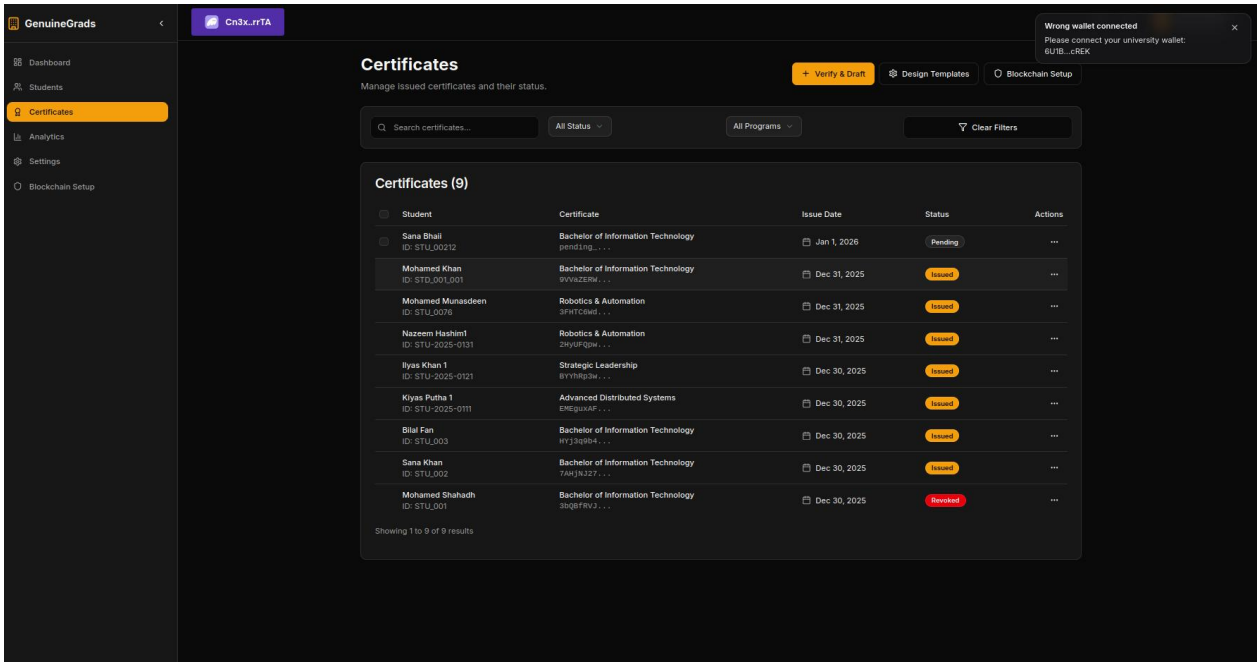


Figure 58: A8 - Unauthorized wallet connected

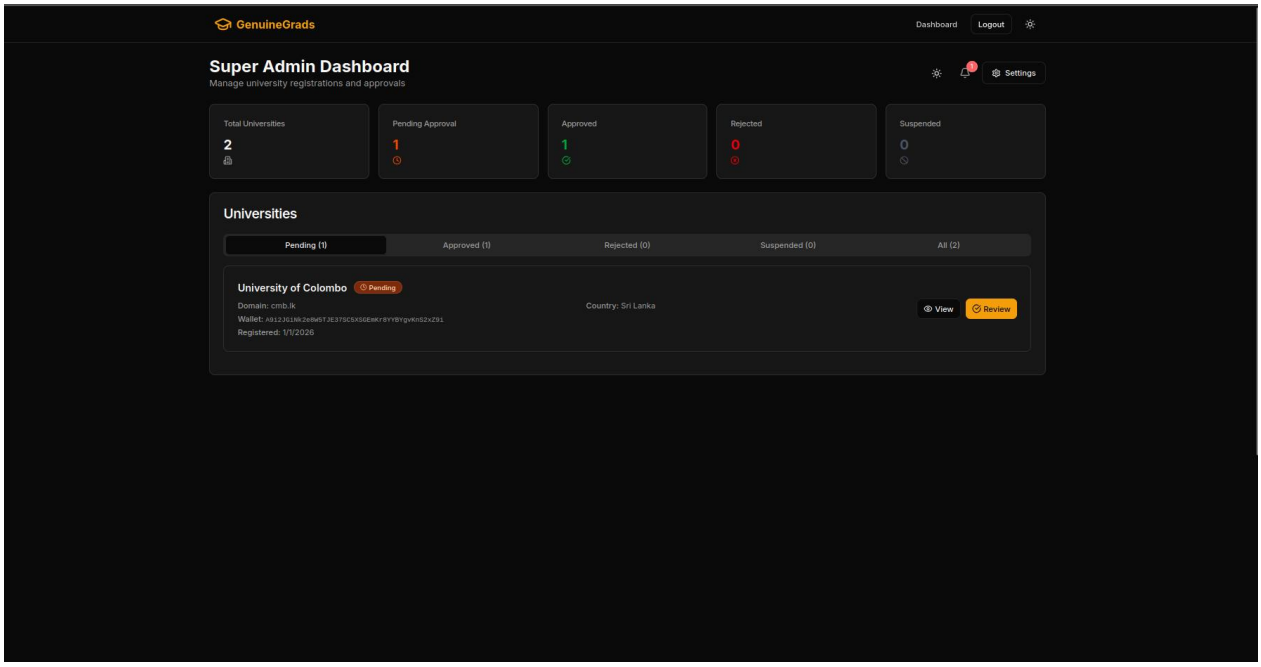


Figure 59: A9 - Super Admin dashboard showing pending university

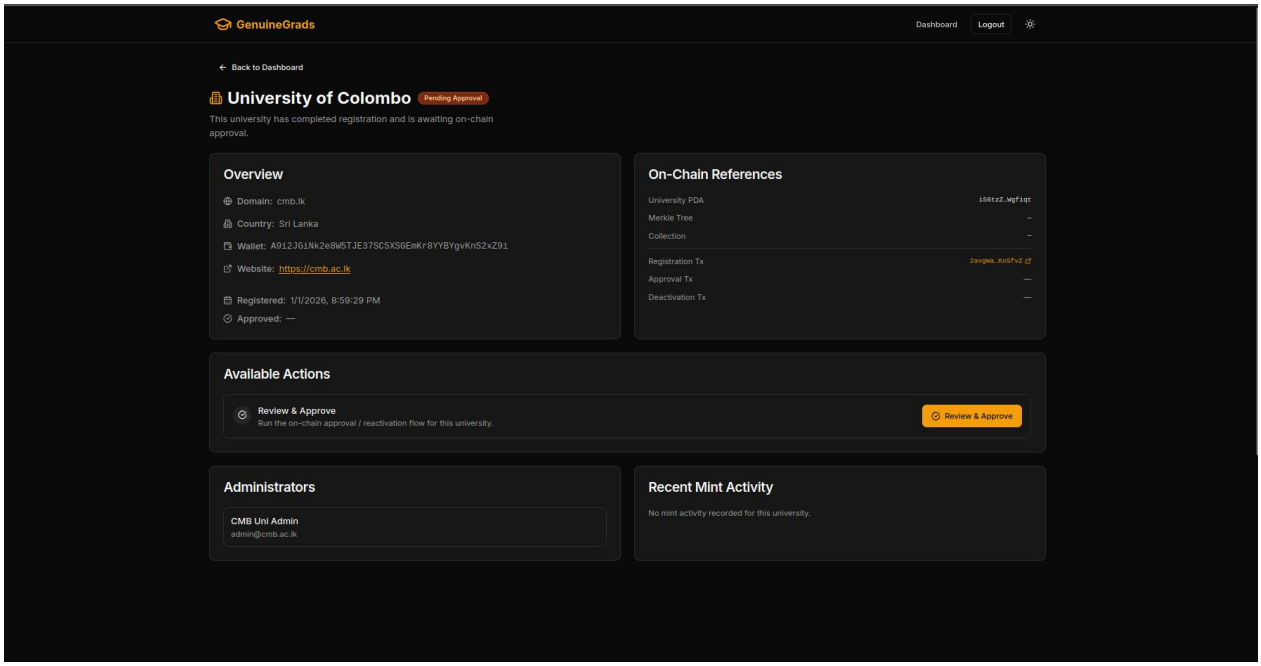


Figure 60: A10 - University details view with pending status

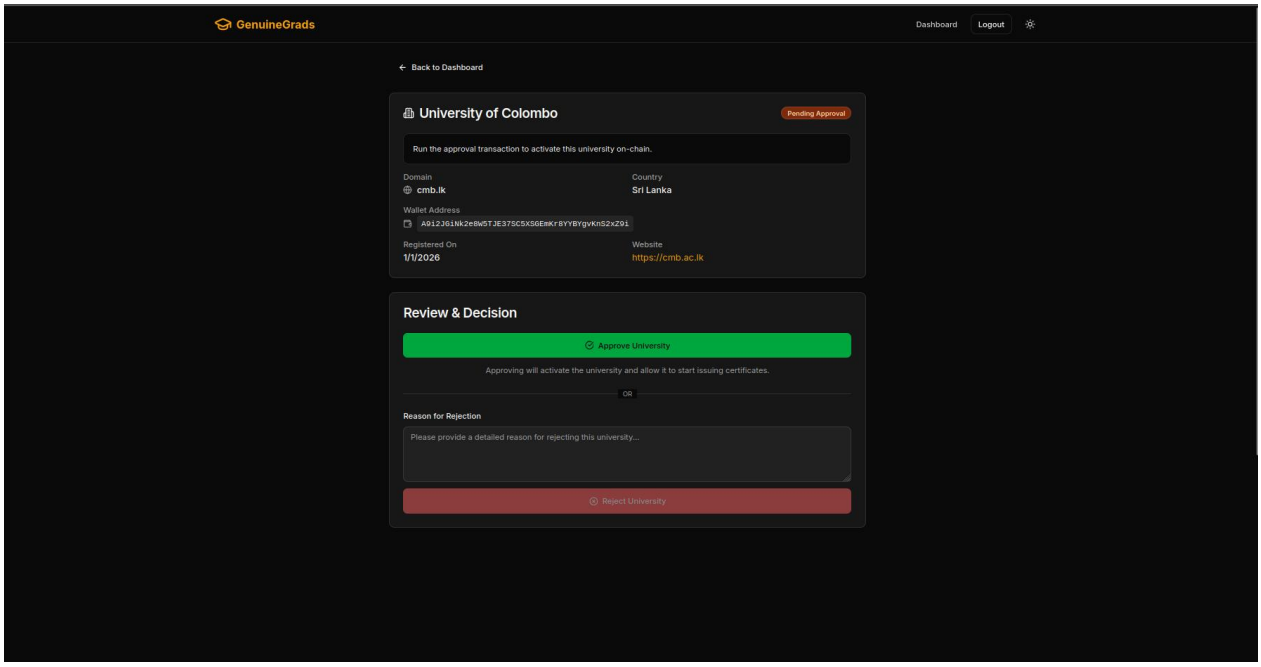


Figure 61: A11 - Review & approve action screen

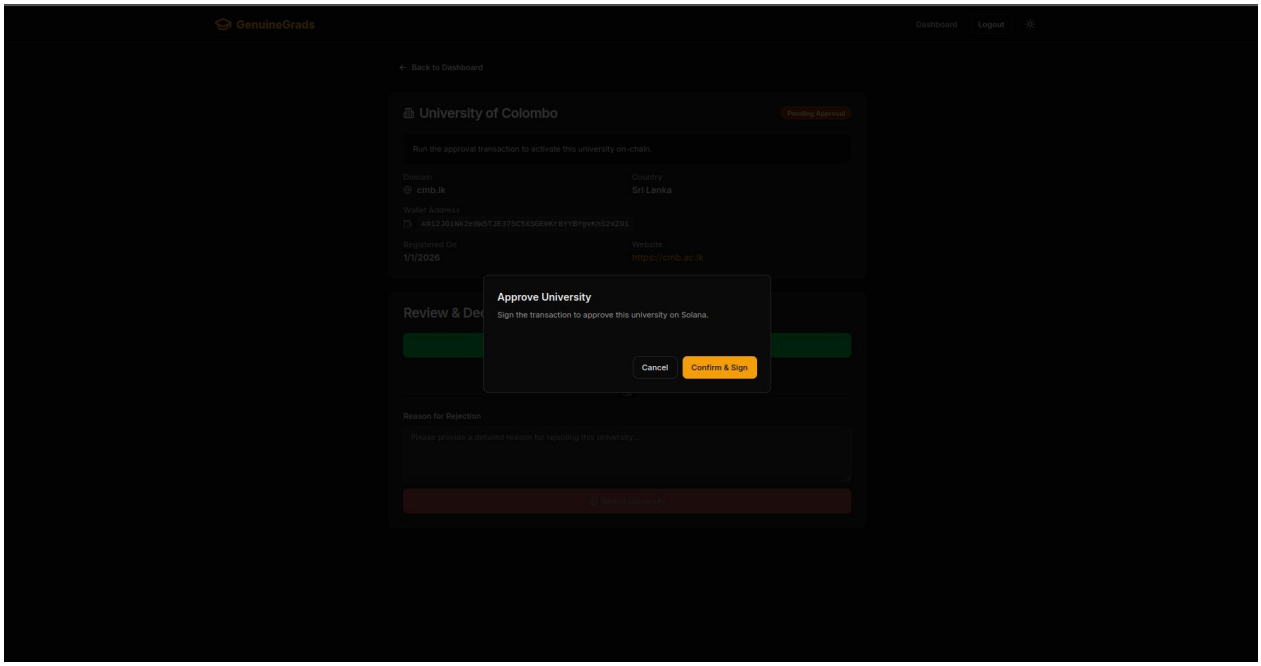


Figure 62: A12 - Approval confirmation dialog

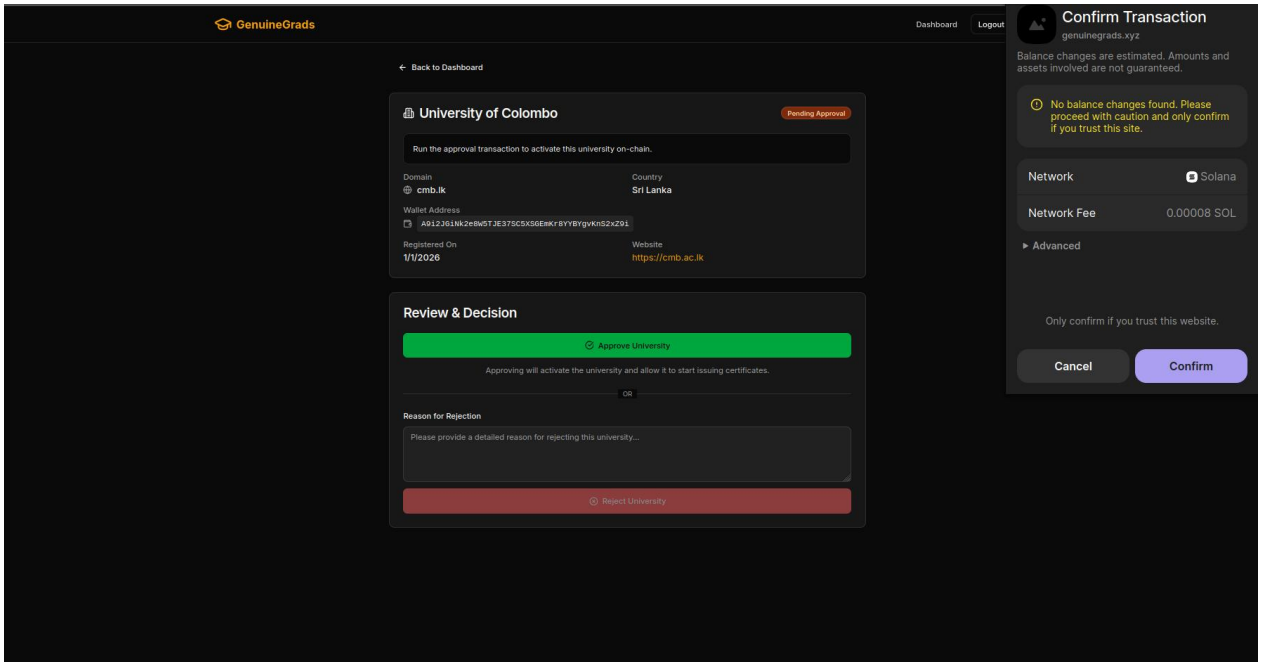


Figure 63: A13 - Wallet transaction signing prompt

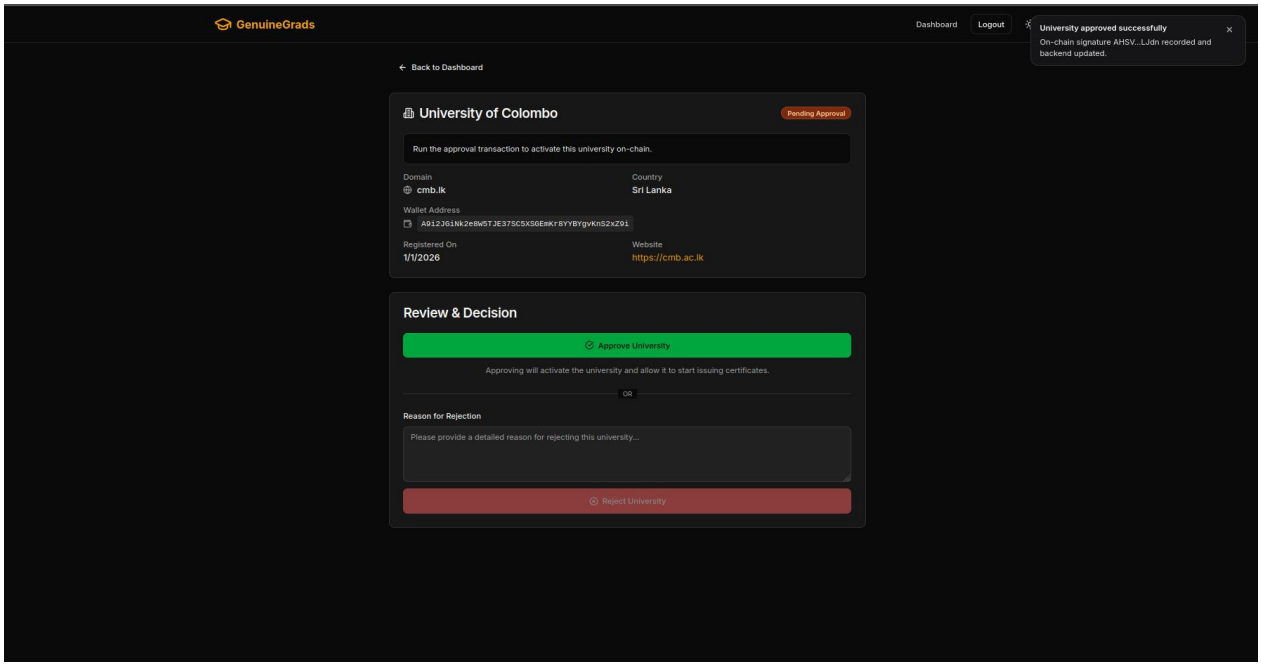


Figure 64: A14 - Blockchain confirmation (RPC success)

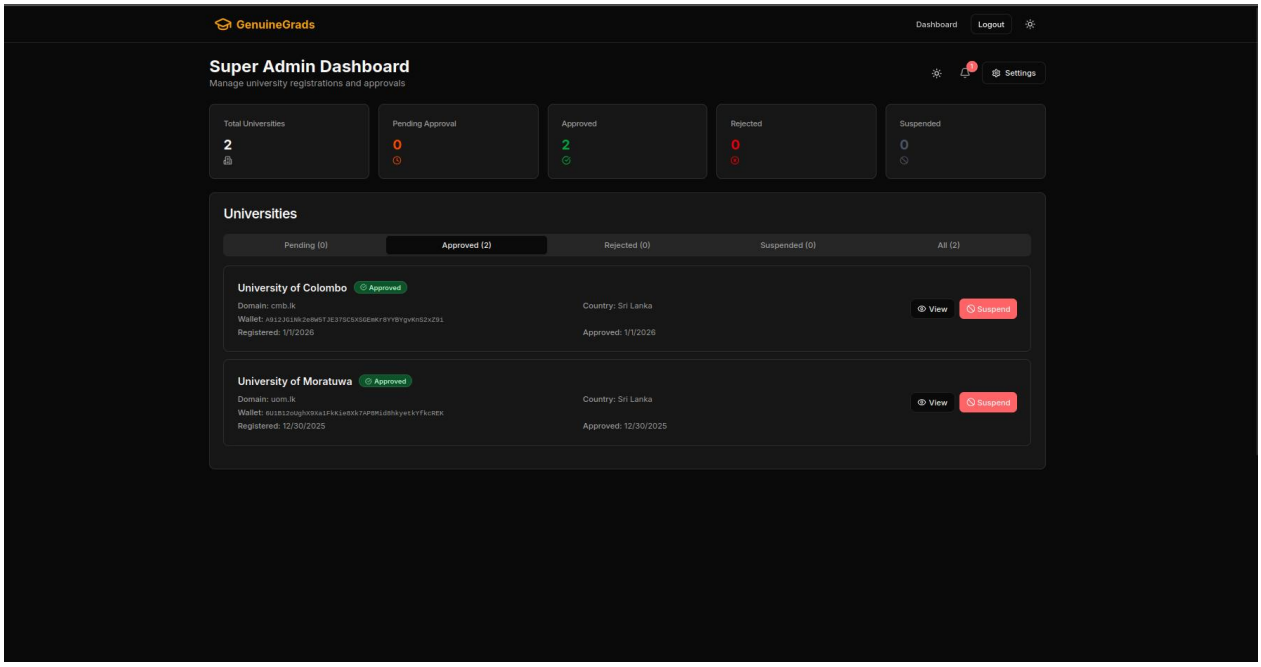


Figure 65: A15 - University listed under approved universities

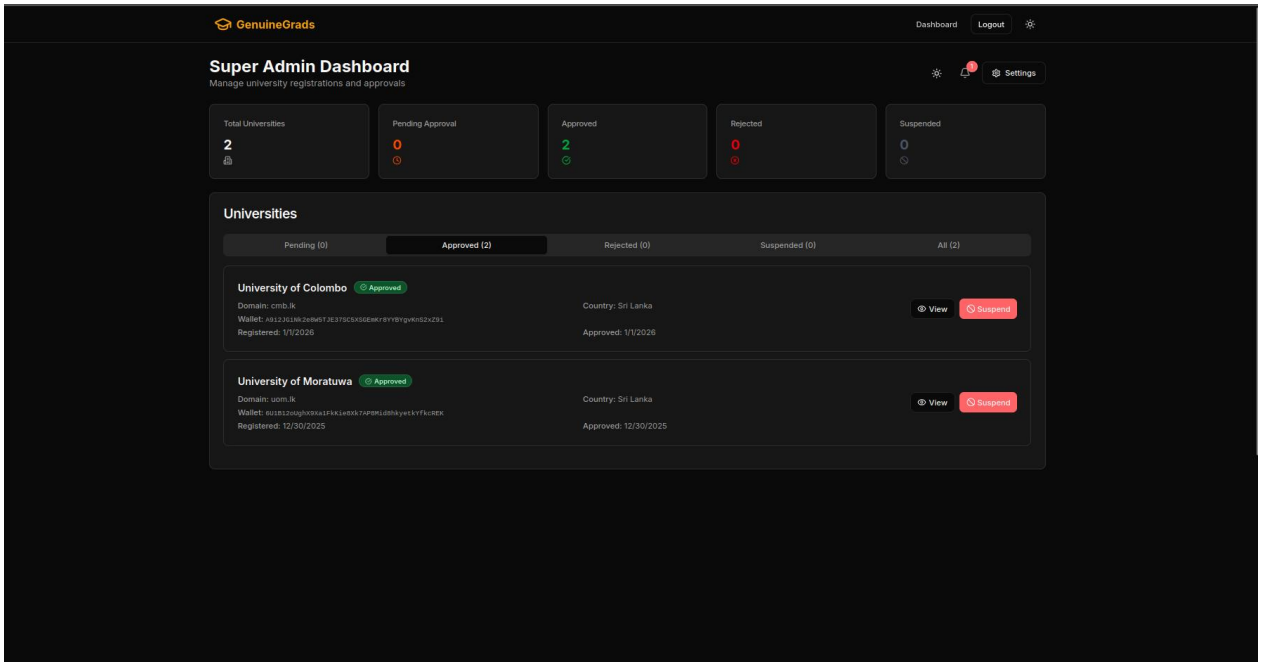


Figure 66: A16 - Approved universities list

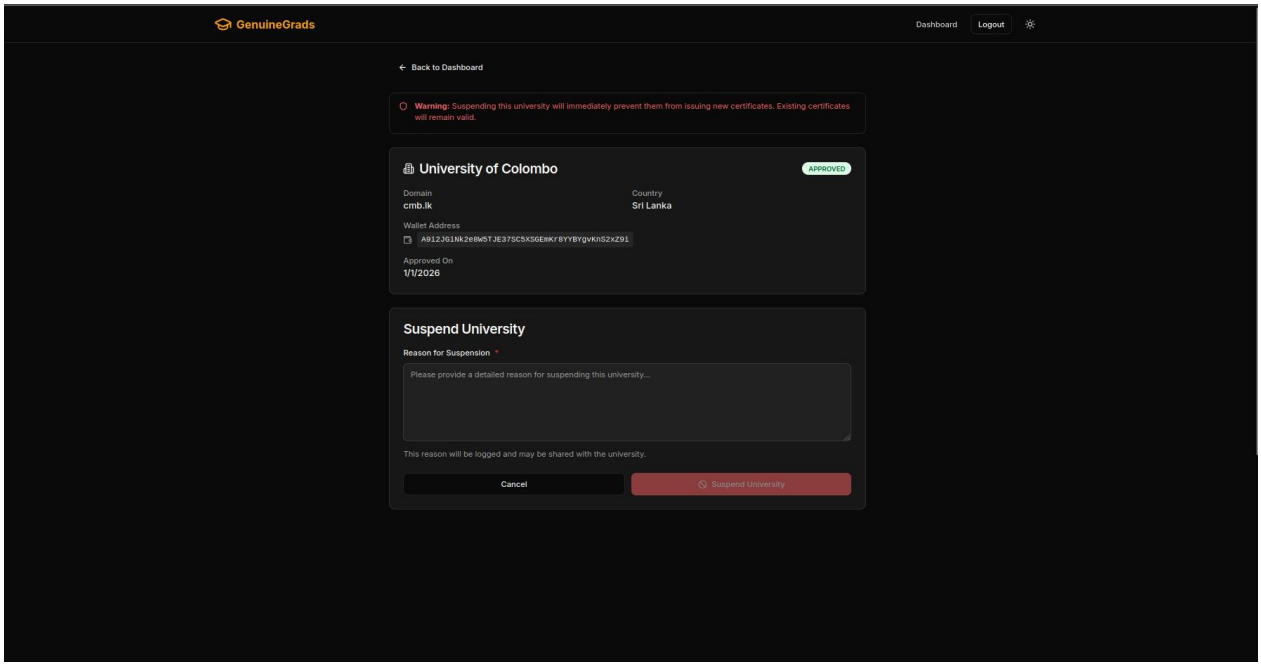


Figure 67: A17 - Suspend university page with warning message

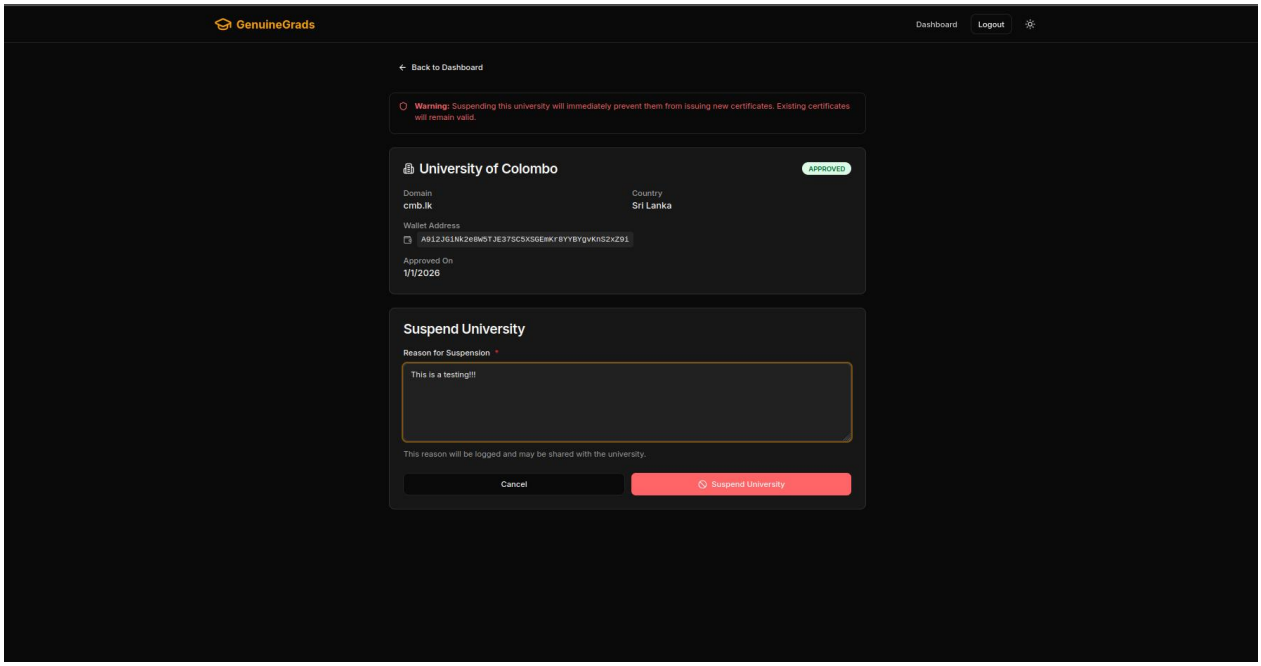


Figure 68: A18 - Suspension reason entered

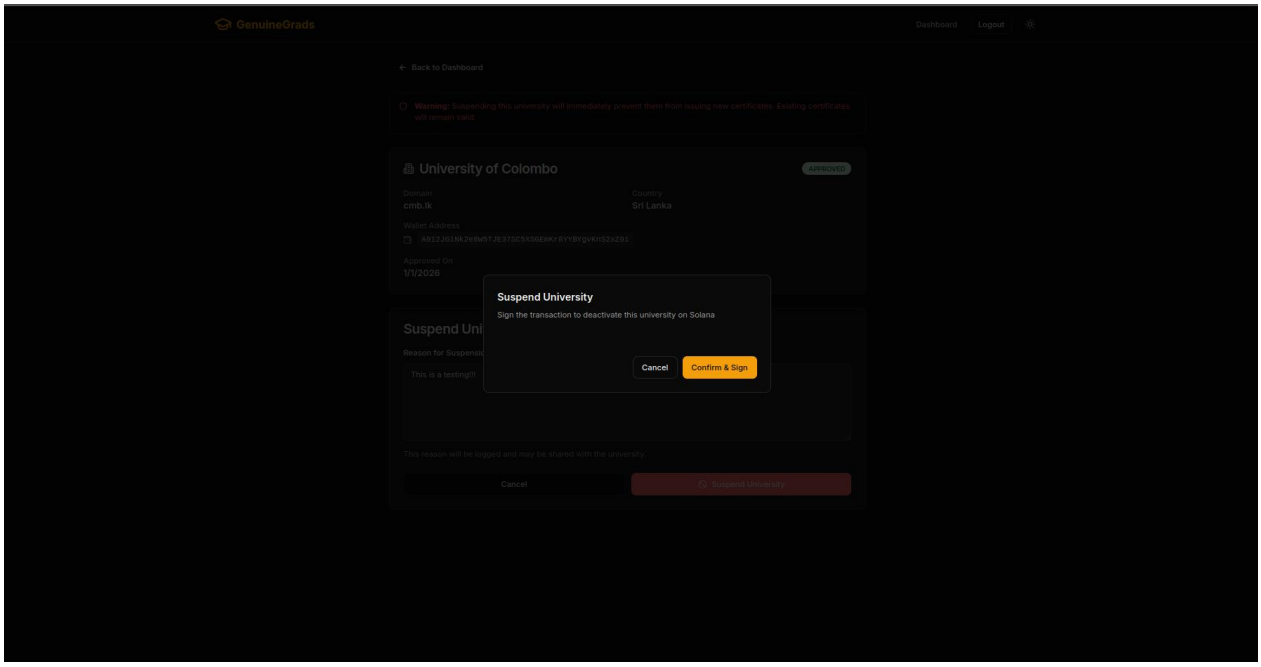


Figure 69: A19 - Suspension confirmation dialog

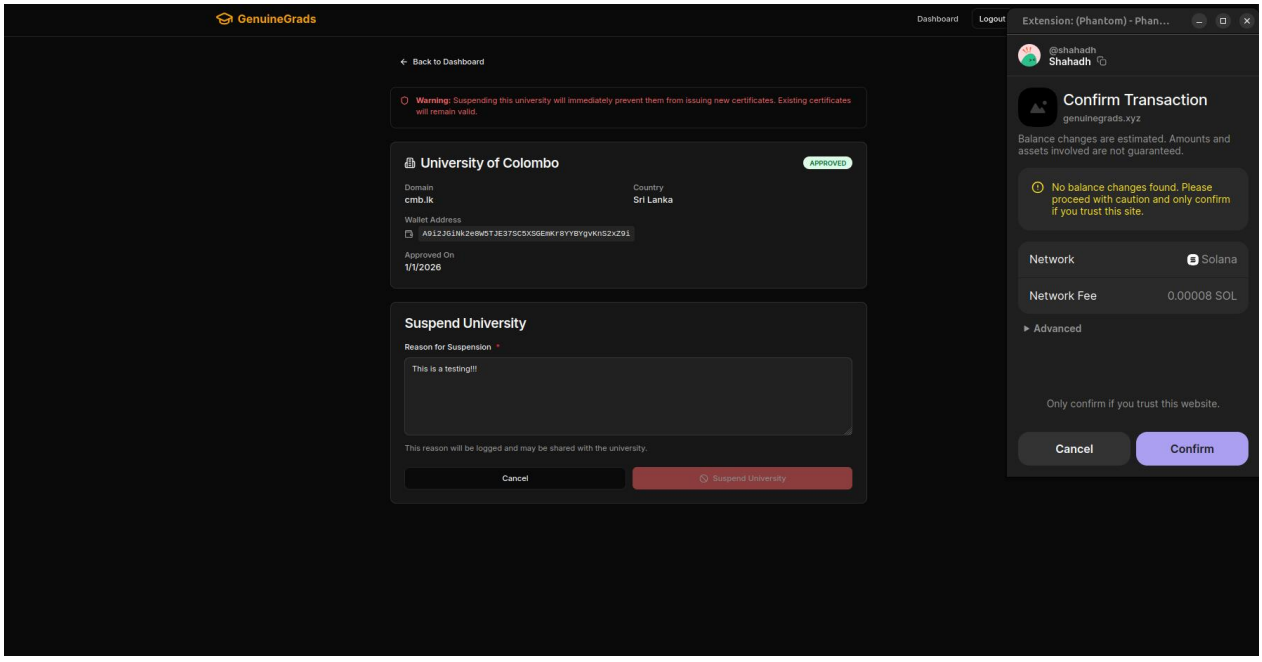


Figure 70: A20 - Solana wallet transaction signing

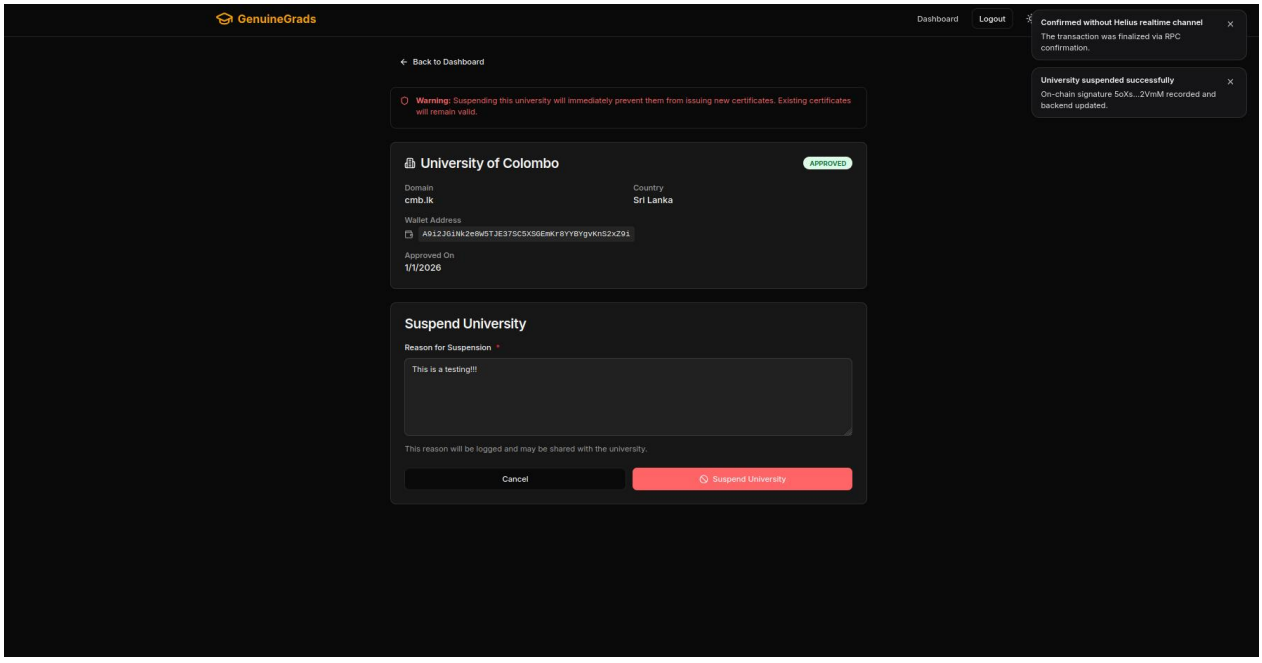


Figure 71: A21 - RPC confirmation message

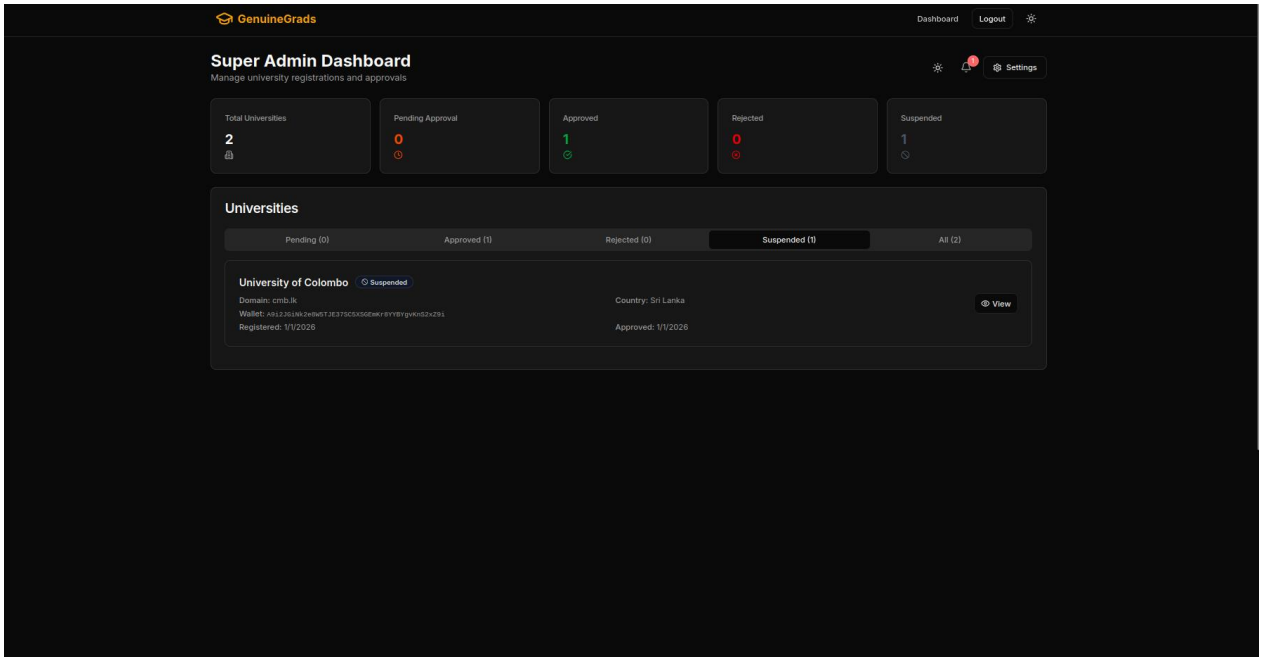


Figure 72: A22 - University listed under suspended universities

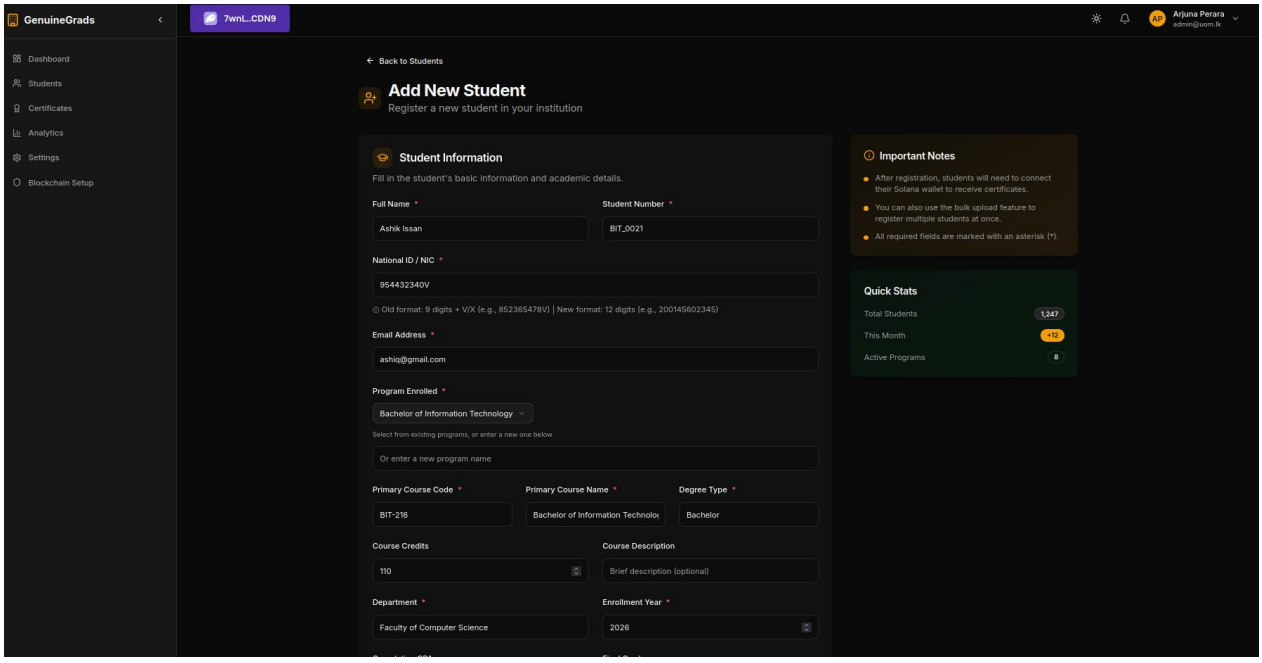


Figure 73: A23 - Student registration form with entered details 01

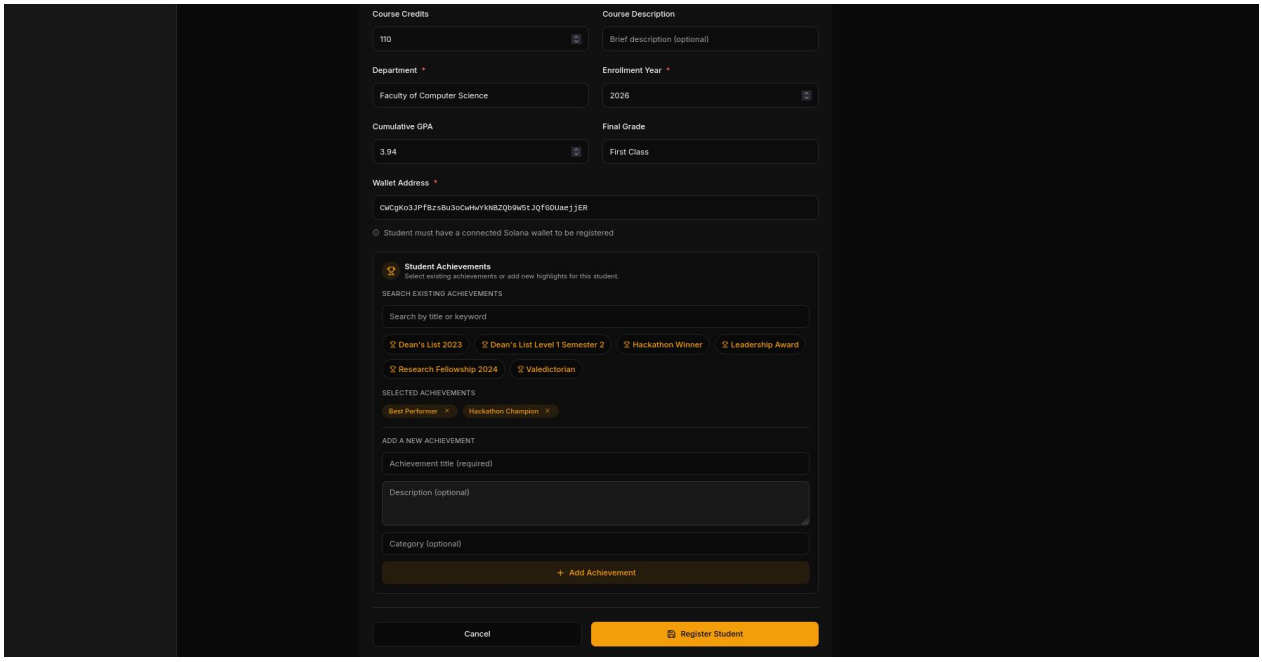


Figure 74: A23 - Student registration form with entered details 02

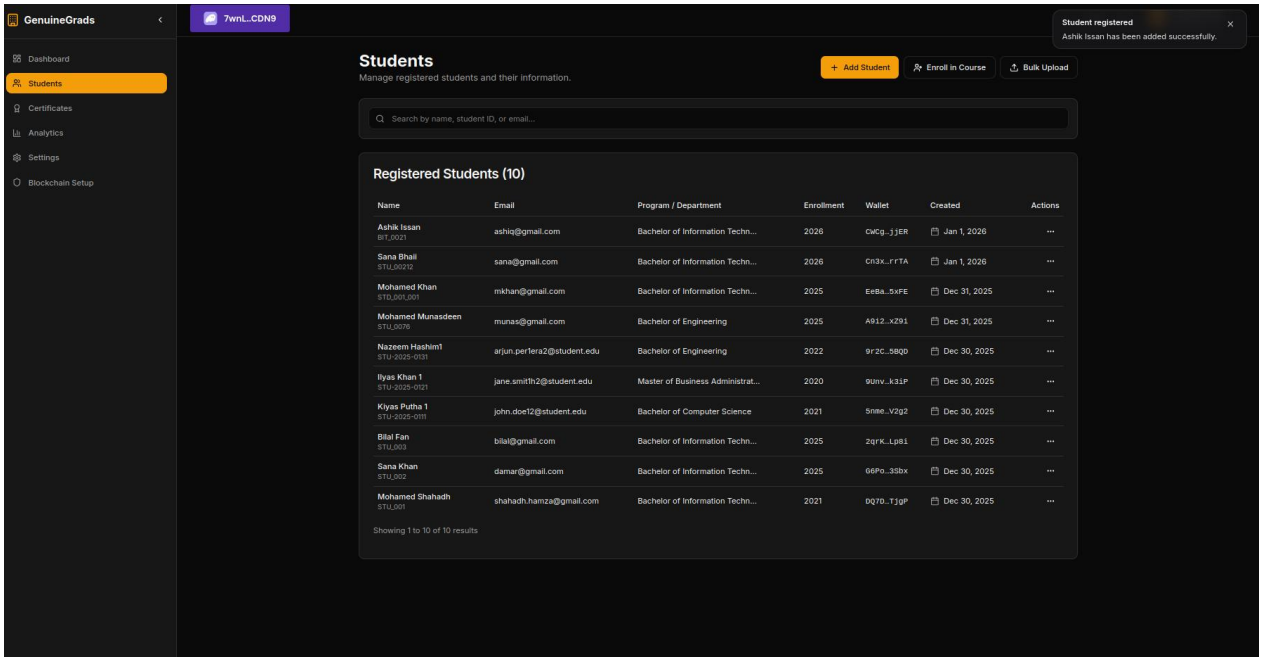


Figure 75: A24 - Students list showing the newly registered student

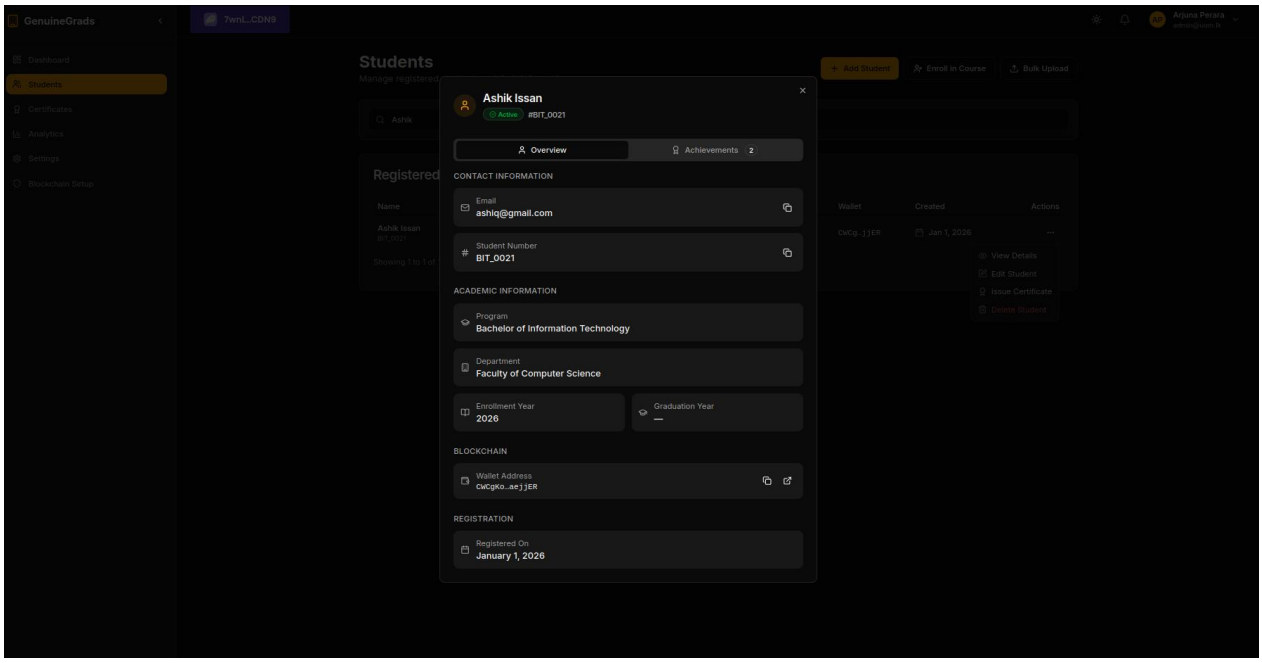
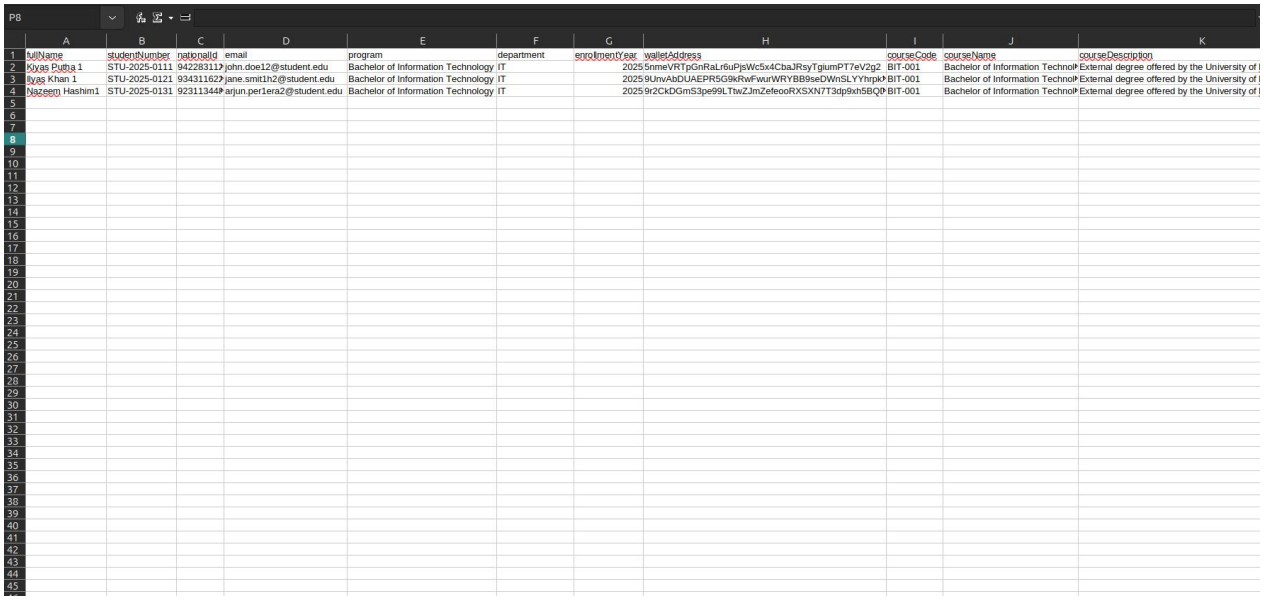


Figure 76: A25 - Student details view



	A	B	C	D	E	F	G	H	I	J	K
1	fullName	studentNumber	nationalId	email	program	department	enrollmentYear	walletAddress	courseCode	courseName	courseDescription
2	Kiyas Putna 1	STU-2025-0111	942283117	john.doe12@student.edu	Bachelor of Information Technology	IT	2025	5nmeVRTpGnRaLr6uPjsWc5x4CbaJRsyTgiumPT7eV2g2	BIT-001	Bachelor of Information Technol	External degree offered by the University of
3	Ilyas Khan 1	STU-2025-0121	934311627	jane.smit1h2@student.edu	Bachelor of Information Technology	IT	2025	9UnvAbDUAEPR5G9kRwFwurWRYBB9seDWnSLYYhrpkP	BIT-001	Bachelor of Information Technol	External degree offered by the University of
4	Nazeem Hashim1	STU-2025-0131	923113449	arjun.per1era2@student.edu	Bachelor of Information Technology	IT	2025	9r2CkDGmS3pe99LTtwZJmZefeooRXSXN7T3dp9xh5BQP	BIT-001	Bachelor of Information Technol	External degree offered by the University of

Figure 77: A26 - CSV file with student data

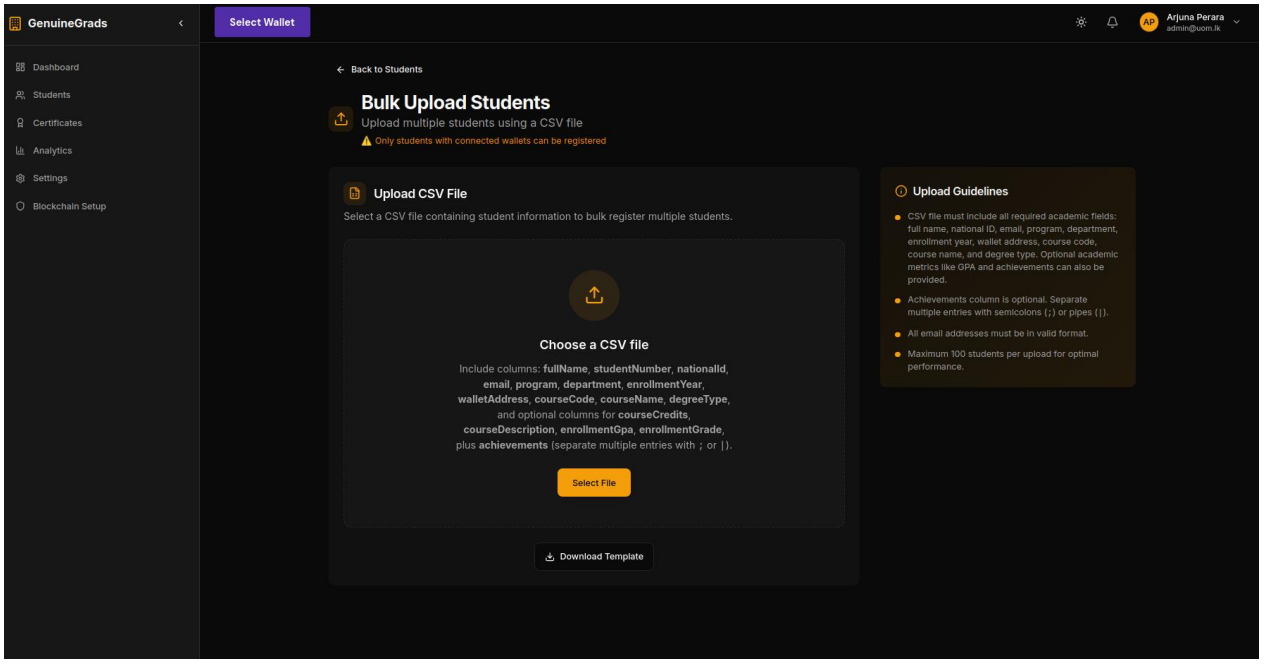


Figure 78: A27 - Bulk upload screen

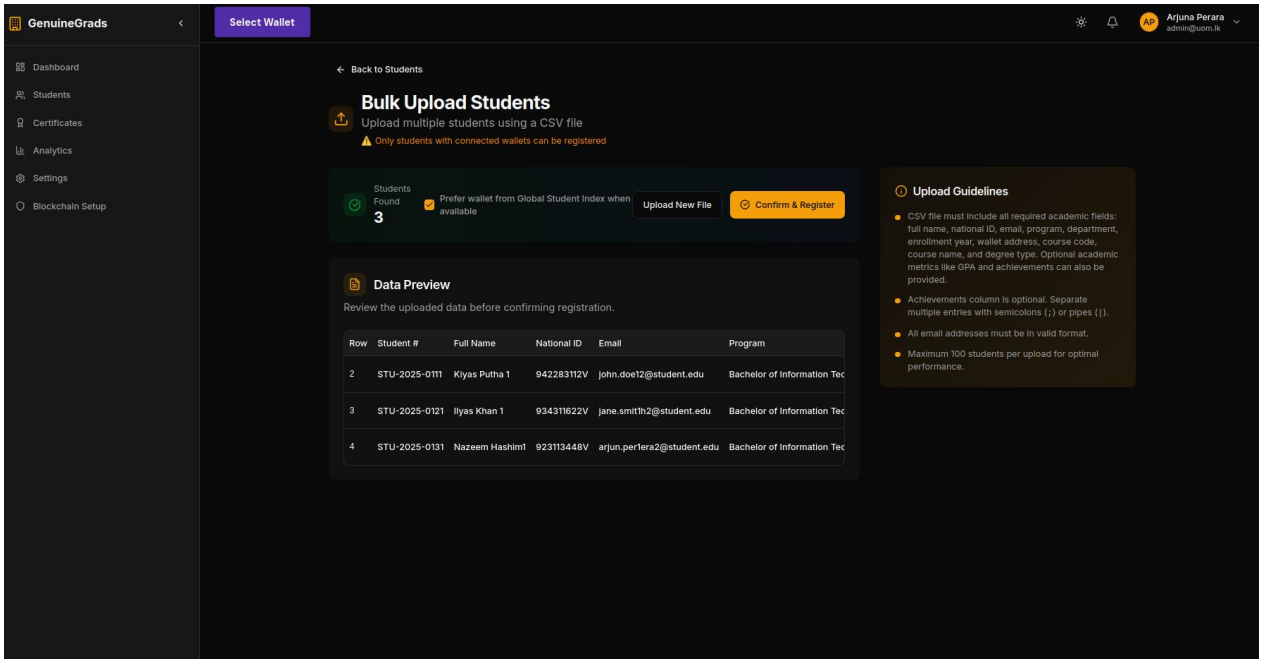


Figure 79: A28 - Data preview table

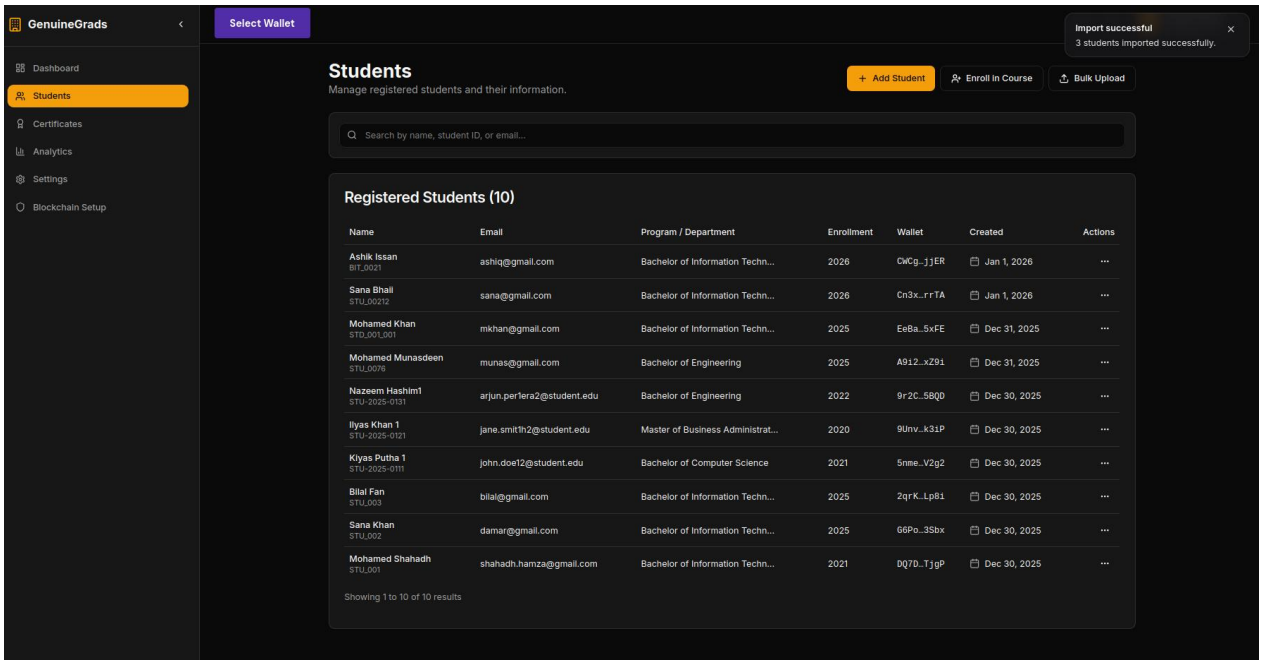
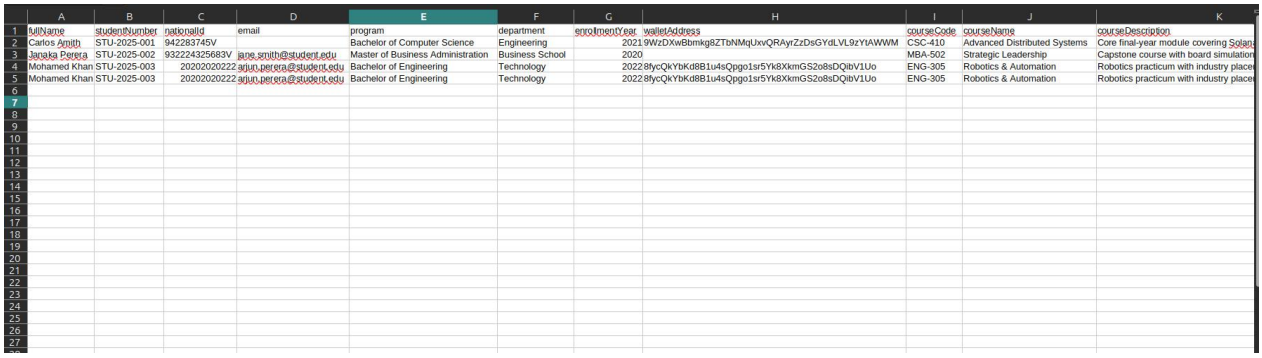


Figure 80: A29 - Import success message with updated registered students list



	A	B	C	D	E	F	G	H	I	J	K
1	fullName	studentNumber	nationalId	email	program	department	enrollmentYear	walletAddress	courseCode	courseName	courseDescription
2	Carlos Amith	STU-2025-001	942283745V		Bachelor of Computer Science	Engineering	2021	9WzDXwBbmkg8ZTbNMqUxvQRAyrZzDsGYdLVL9zYtAWWM	CSC-410	Advanced Distributed Systems	Core final-year module covering Solan
3	Janaka Perera	STU-2025-002	932224325683V	jane.smith@student.edu	Master of Business Administration	Business School	2020		MBA-502	Strategic Leadership	Capstone course with board simulation
4	Mohamed Khan	STU-2025-003	20202020222	arjun.perera@student.edu	Bachelor of Engineering	Technology	2022	8fycQkYbKd8B1u4sQpgo1sr5Yk8XkmGS2o8sDQibV1Uo	ENG-305	Robotics & Automation	Robotics practicum with industry place
5	Mohamed Khan	STU-2025-003	20202020222	arjun.perera@student.edu	Bachelor of Engineering	Technology	2022	8fycQkYbKd8B1u4sQpgo1sr5Yk8XkmGS2o8sDQibV1Uo	ENG-305	Robotics & Automation	Robotics practicum with industry place
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											
27											

Figure 81: A30 - Invalid CSV file

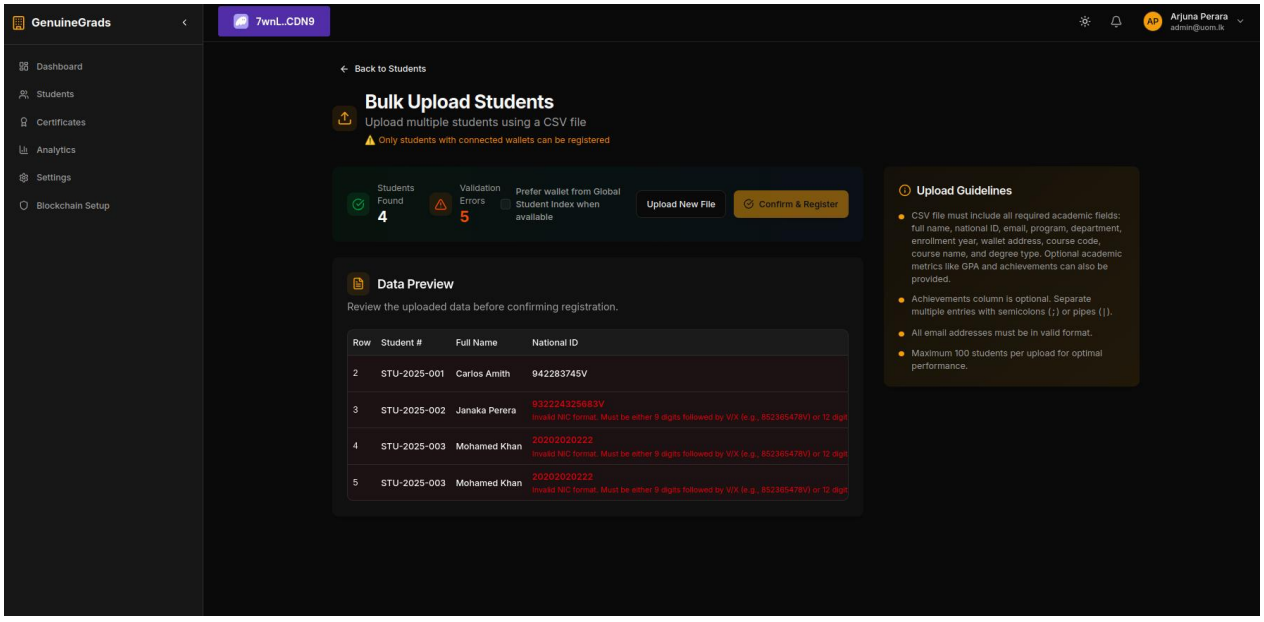


Figure 82: A31 - Data preview highlighting invalid and duplicate rows

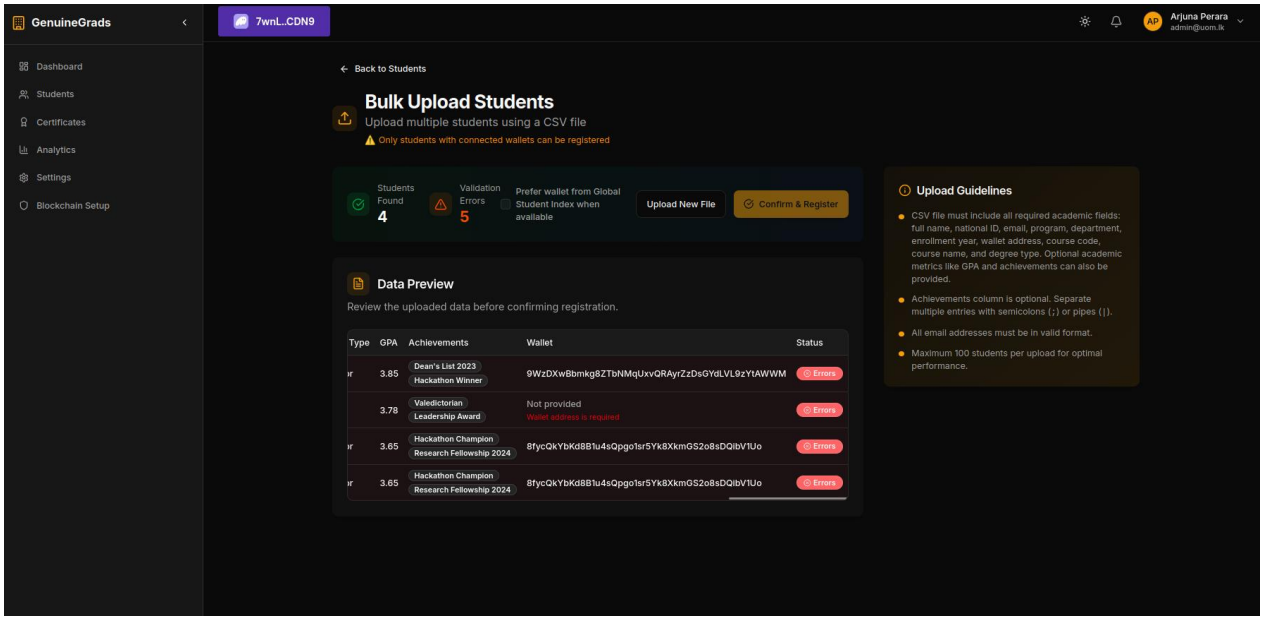


Figure 83: A32 - Error messages displayed for each invalid record

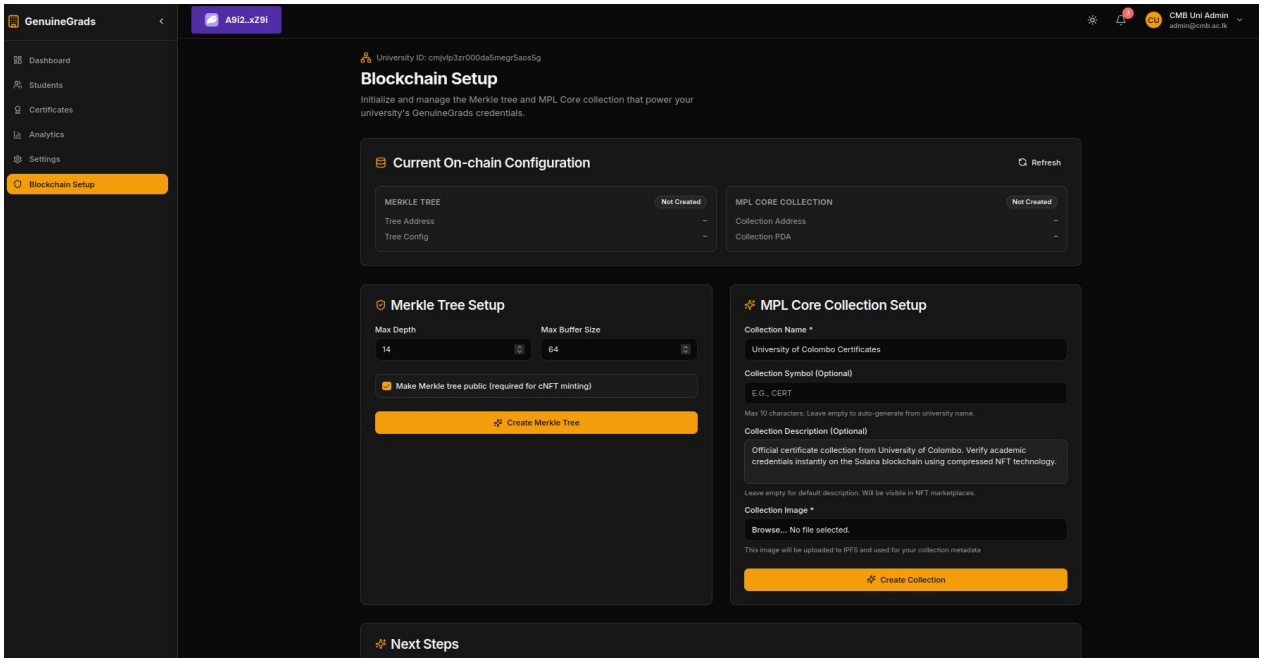


Figure 84: A33 - Blockchain setup screen showing missing Merkle Tree and Collection

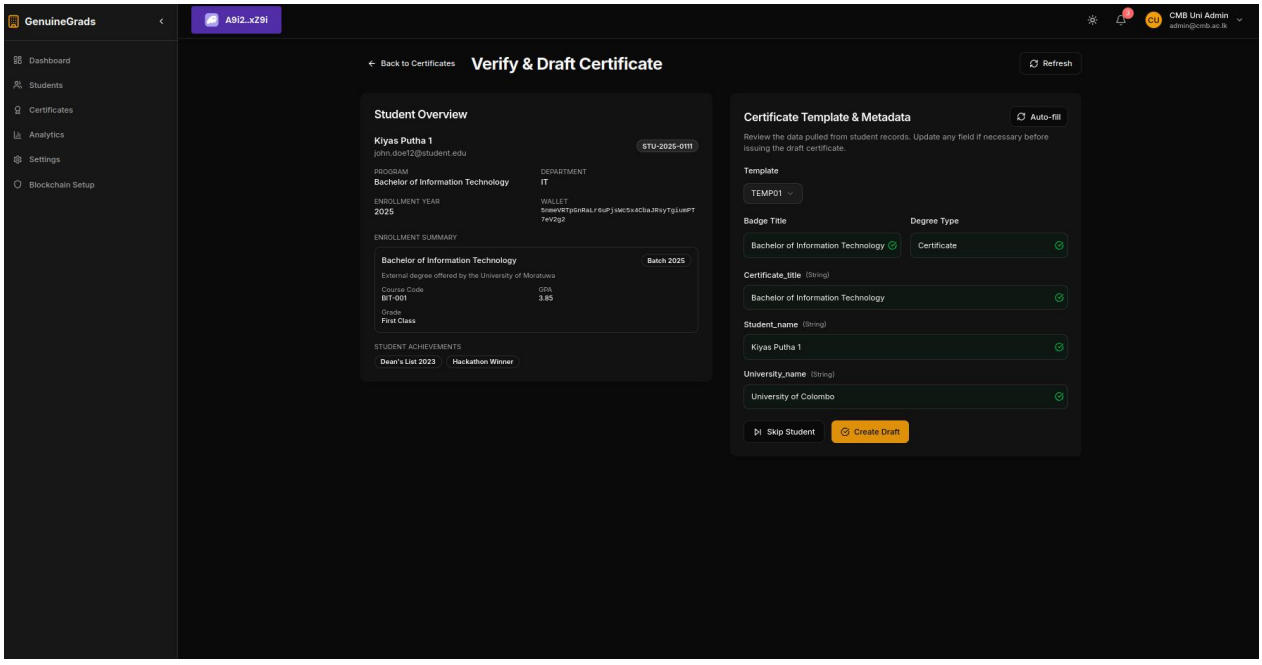


Figure 85: A34 - Verify & Draft Certificate screen

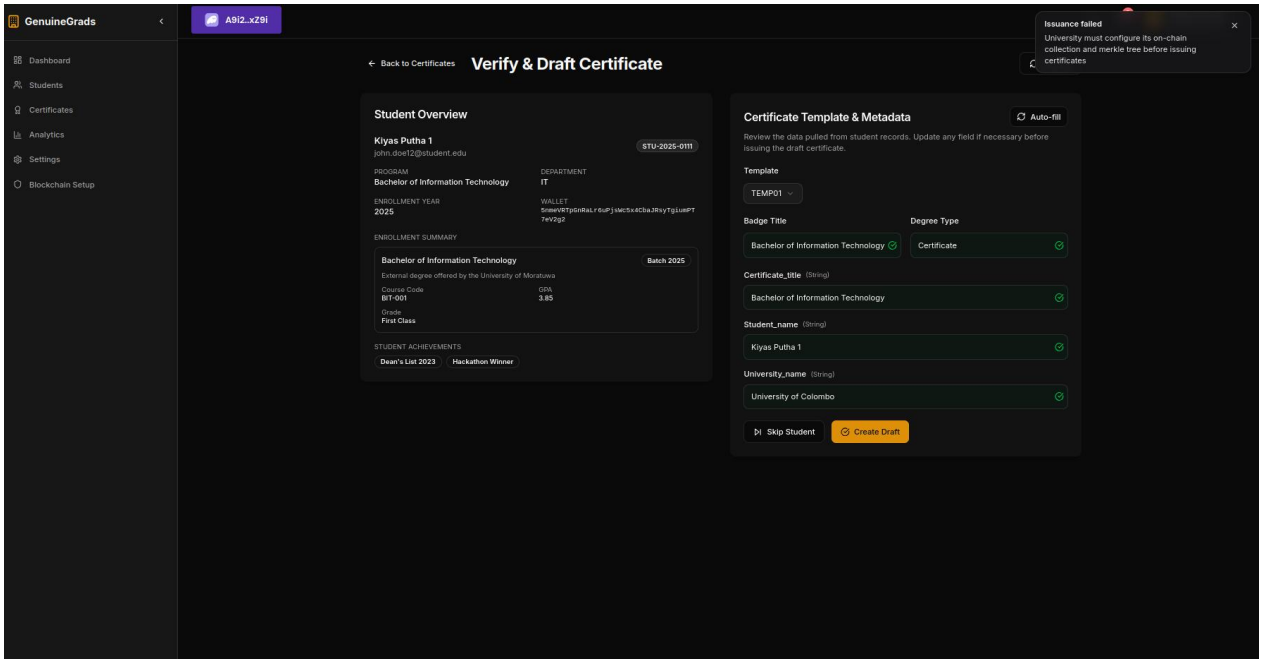


Figure 86: A35 - Error notification indicating issuance failure due to missing on-chain configuration

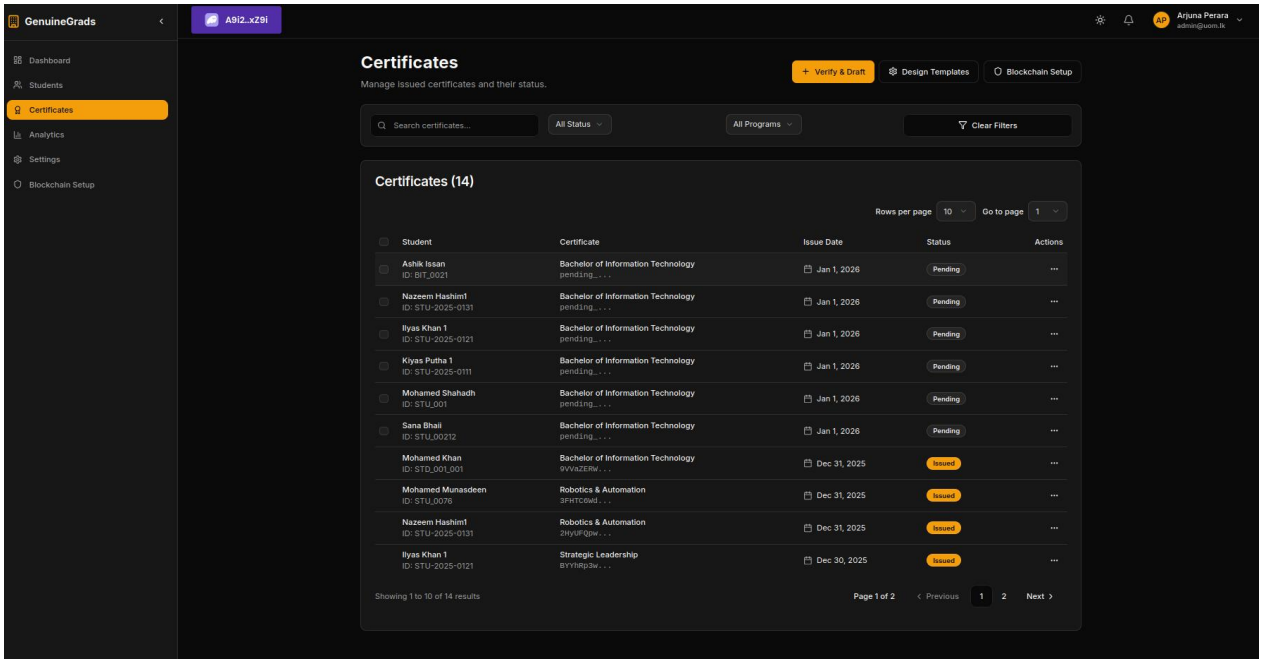


Figure 87: A36 - Pending certificate list

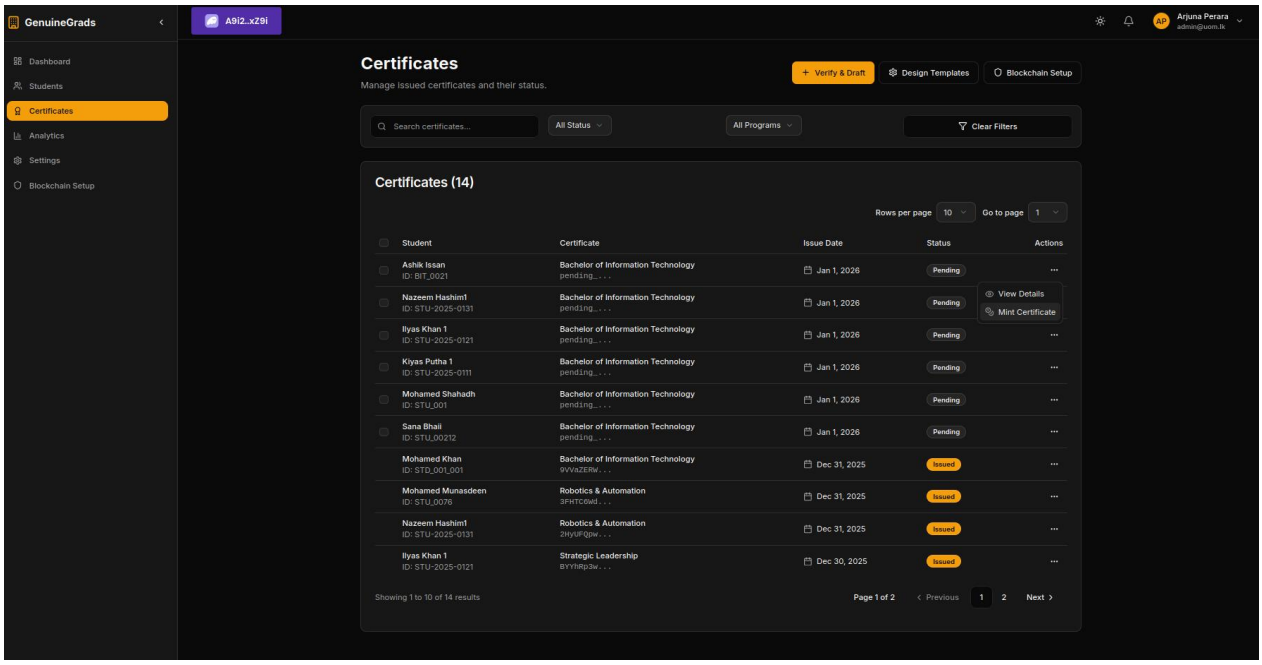


Figure 88: A37 - Mint certificate action

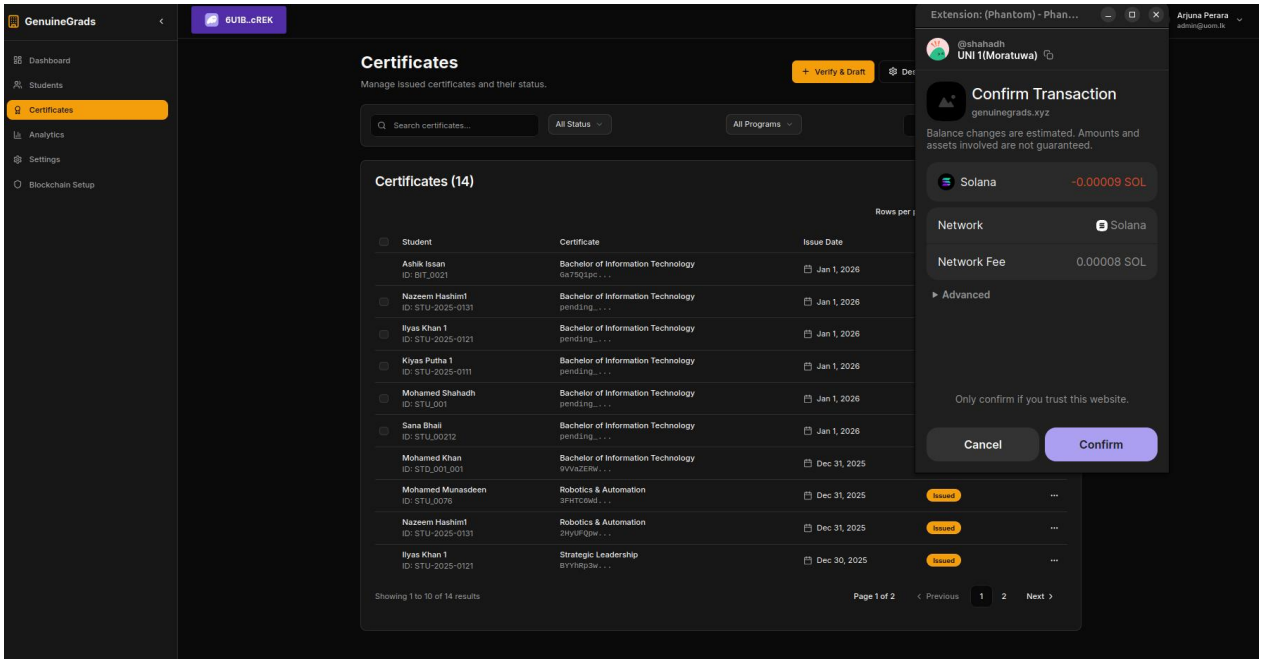


Figure 89: A38 - Wallet transaction confirmation

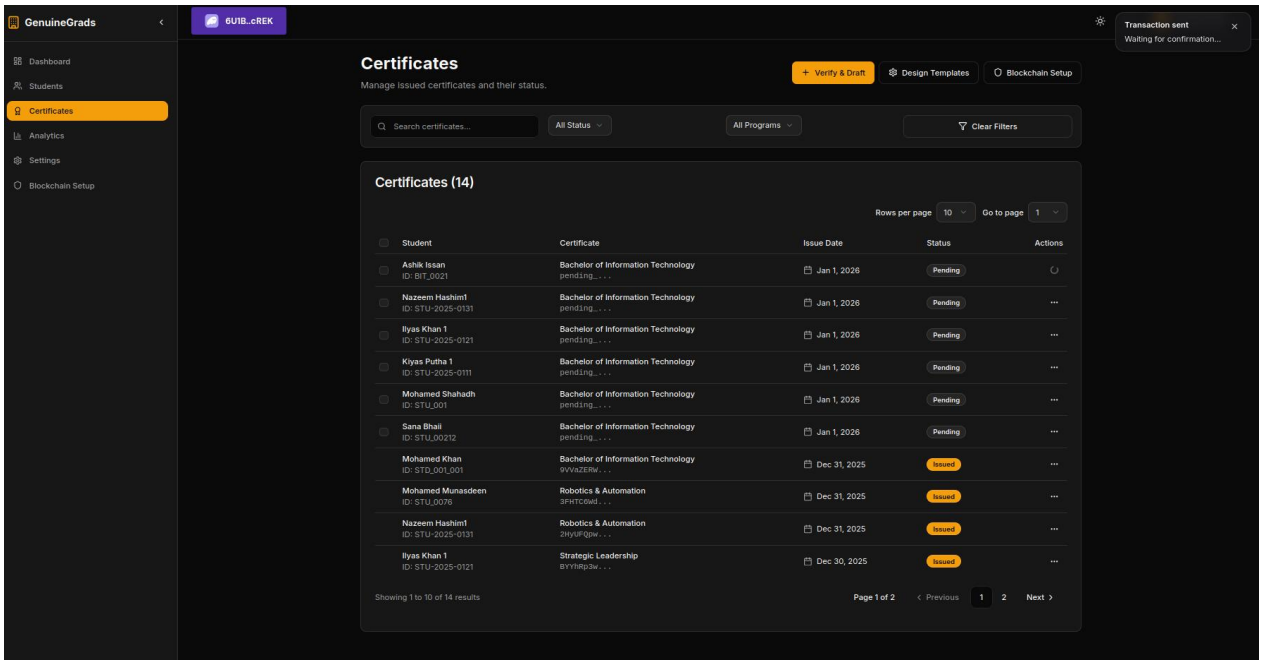


Figure 90: A39 - Transaction sent and waiting for confirmation

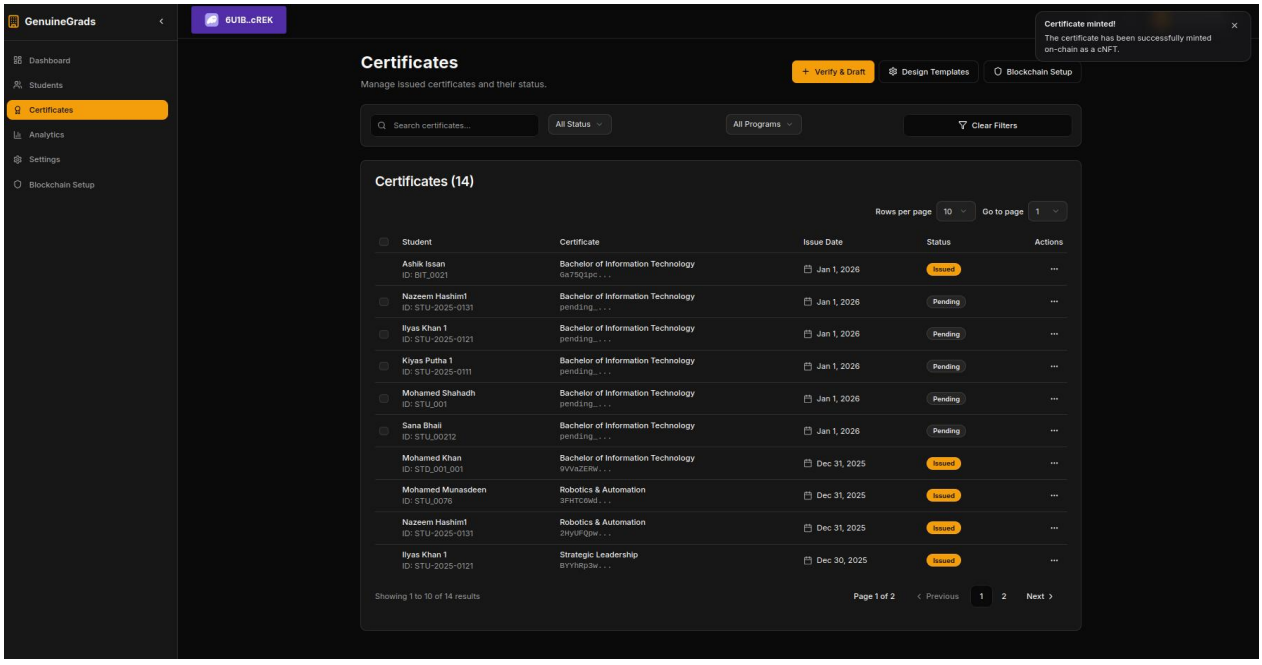


Figure 91: A40 - Certificate status updated to Issued & Transaction confirmation message

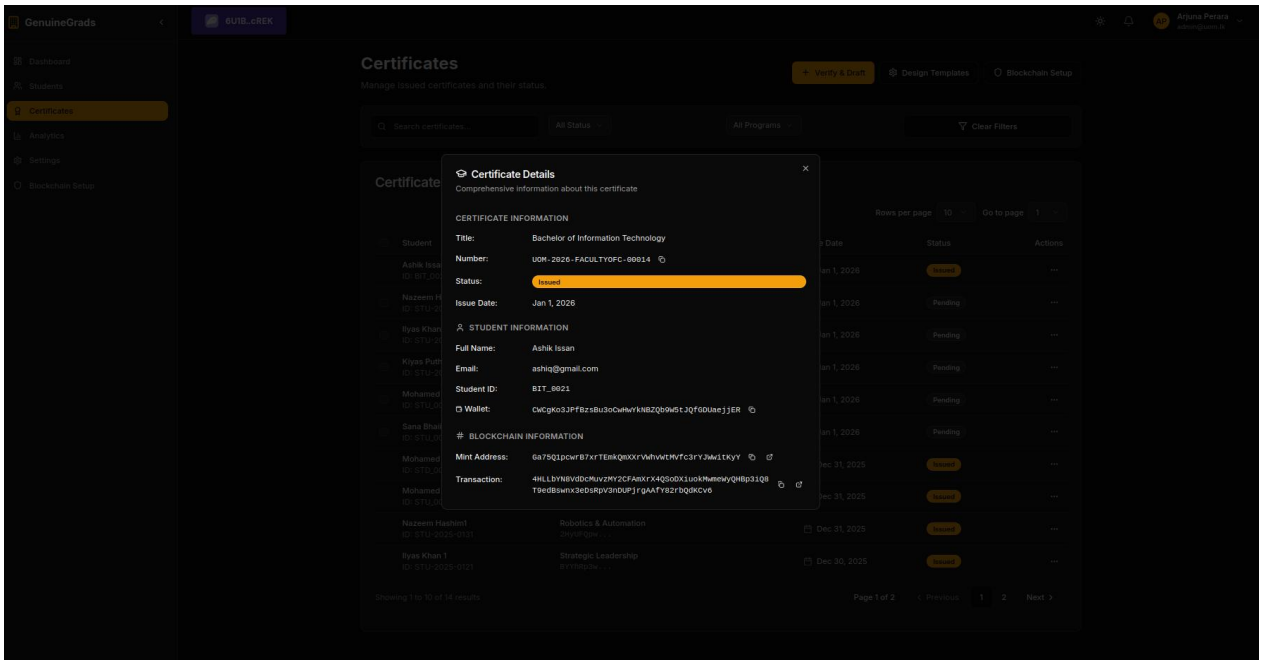


Figure 92: A41 - Certificate details modal

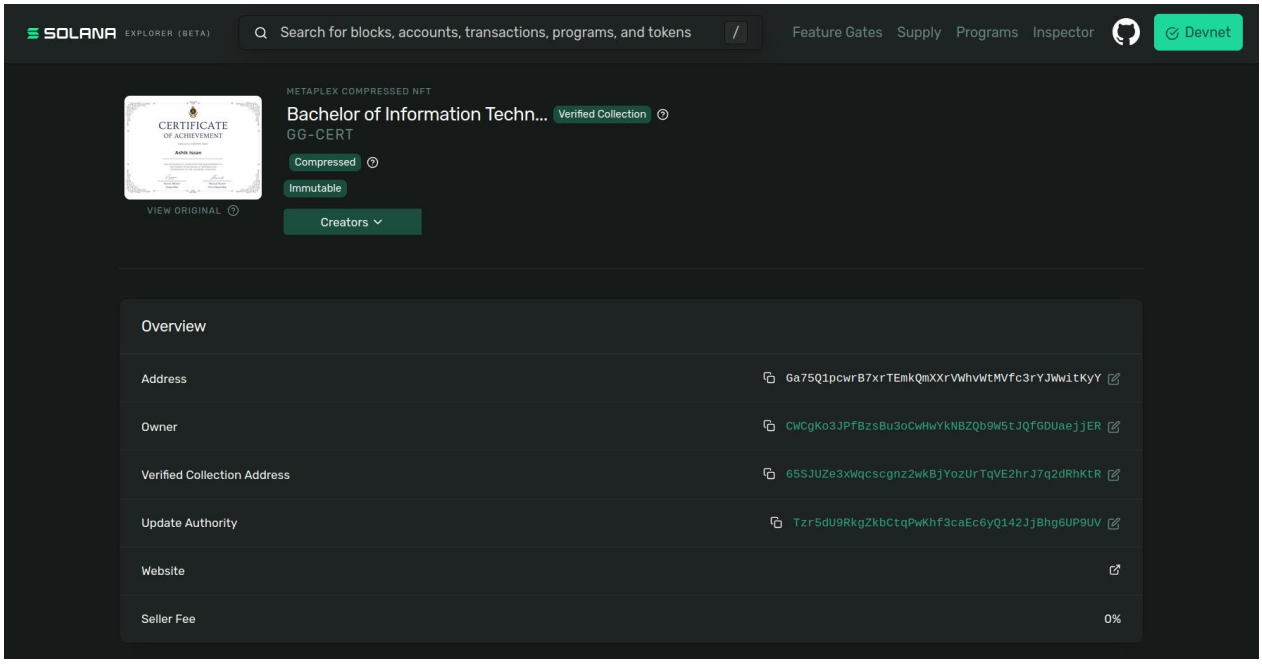


Figure 93: A42 - Solana Explorer view of minted cNFT

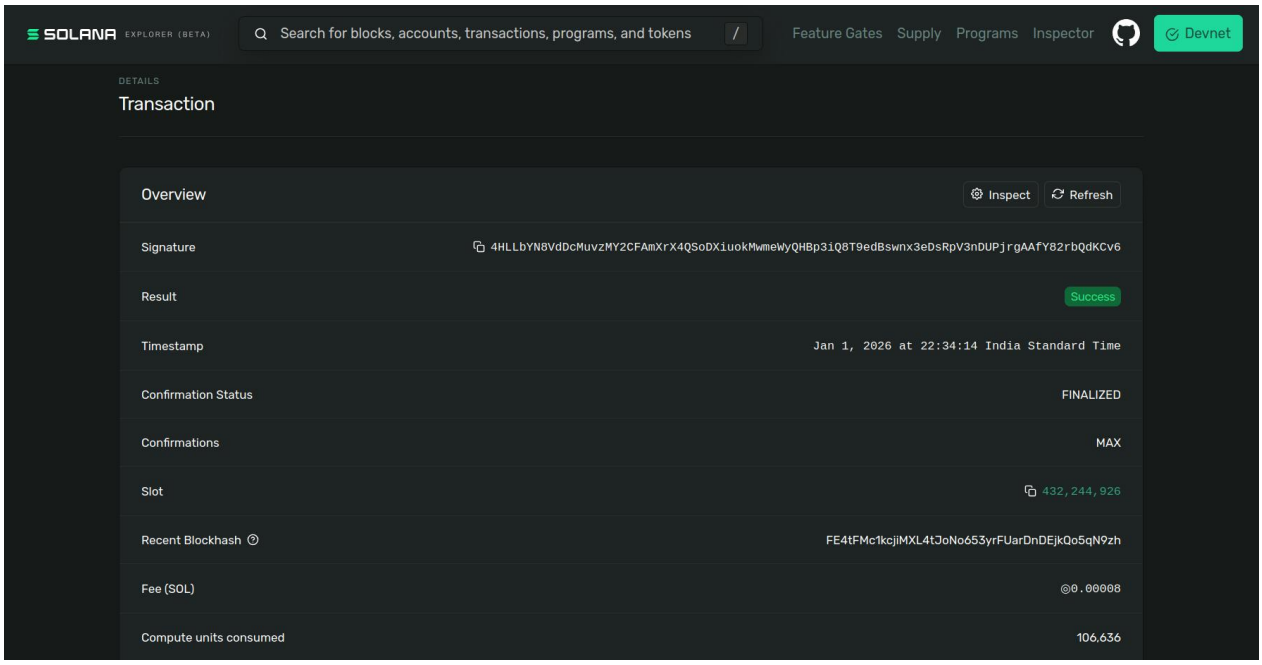


Figure 94: A43 - Solana Explorer view of transaction confirmation

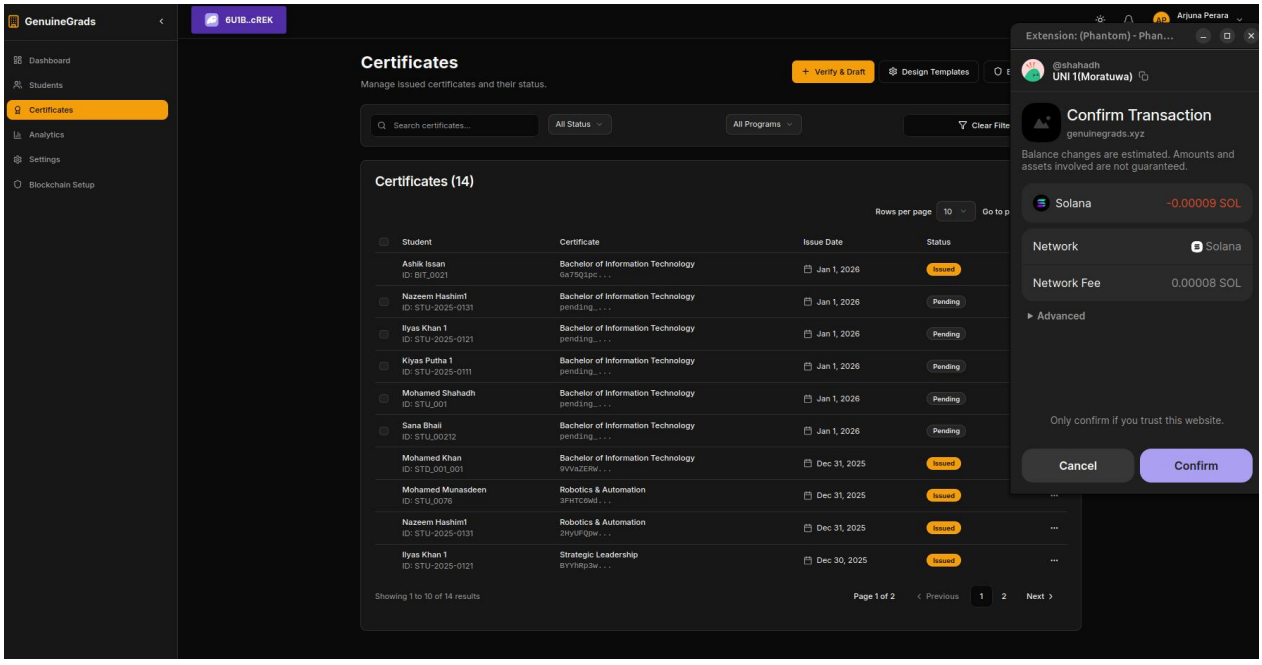


Figure 95: A44 - Screenshot showing wallet confirmation popup.

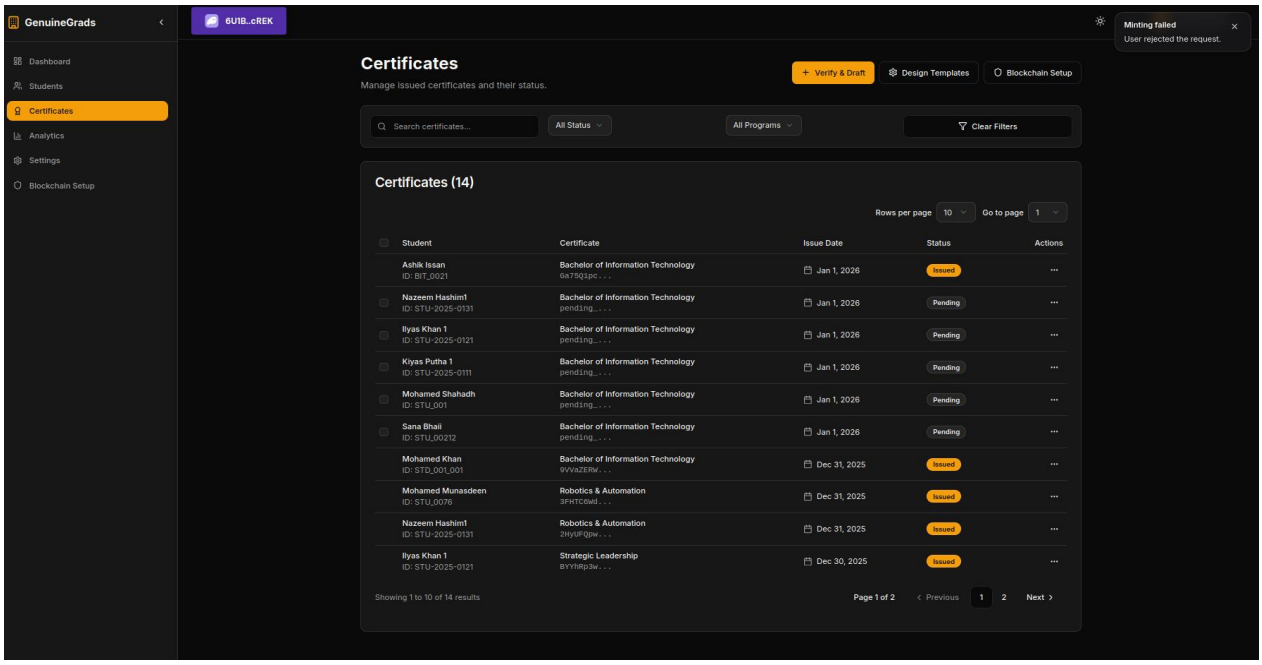


Figure 96: A45 - Screenshot displaying mint failure notification and unchanged certificate status.

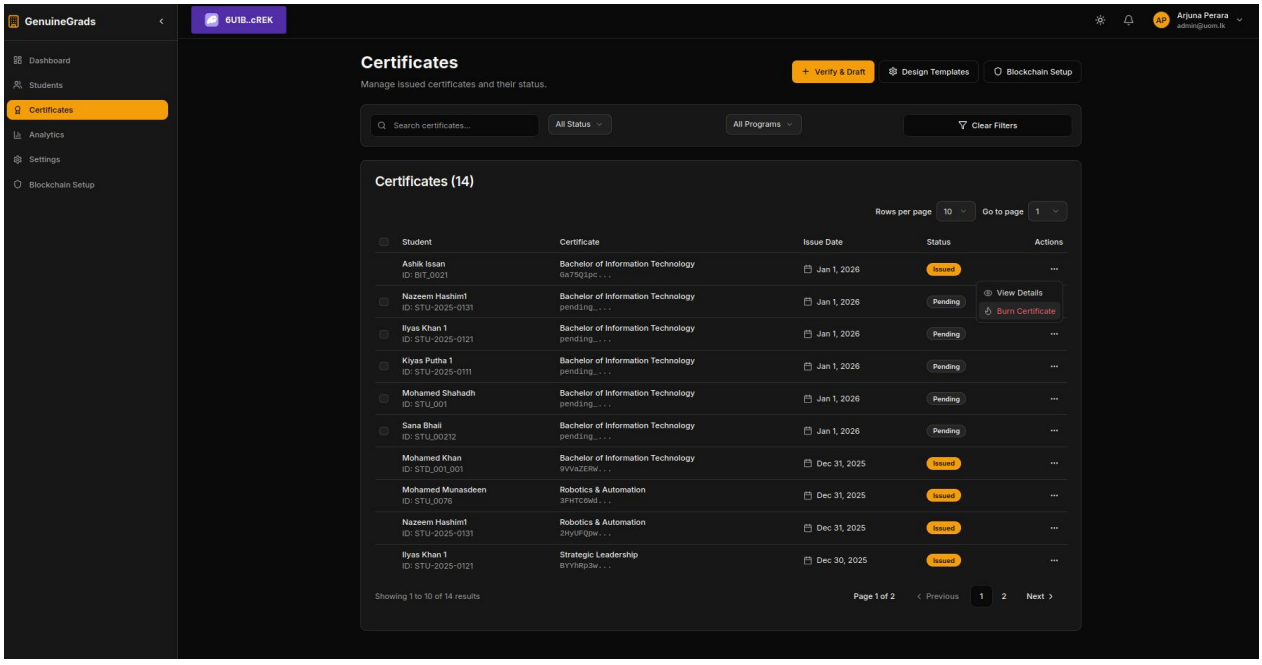


Figure 97: A46 - Burn Action

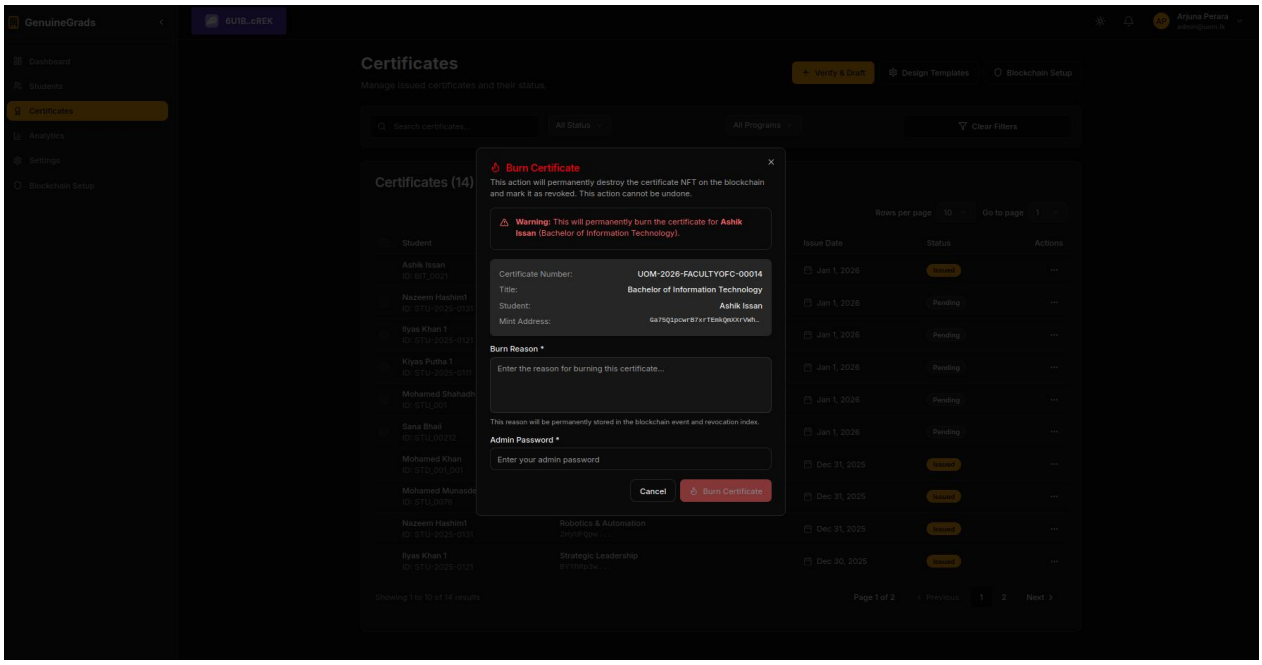


Figure 98: A47 - Burn confirmation dialog was displayed with certificate details.

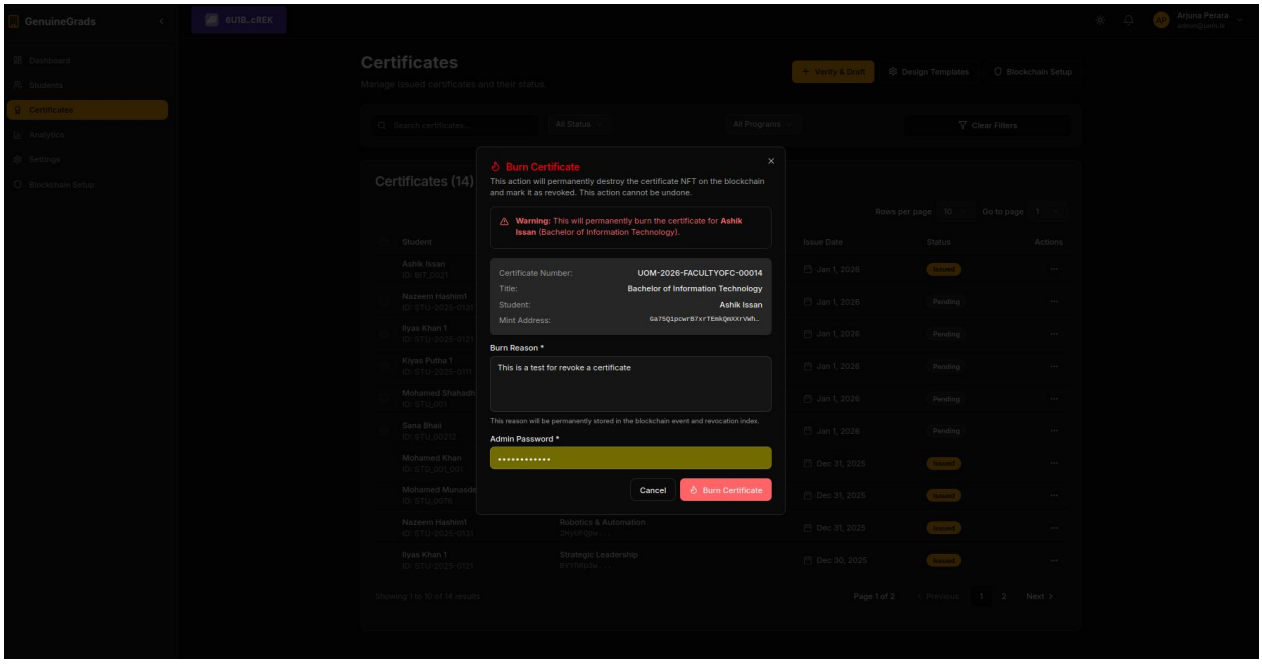


Figure 99: A48 - Admin provided a valid burn reason and credentials.

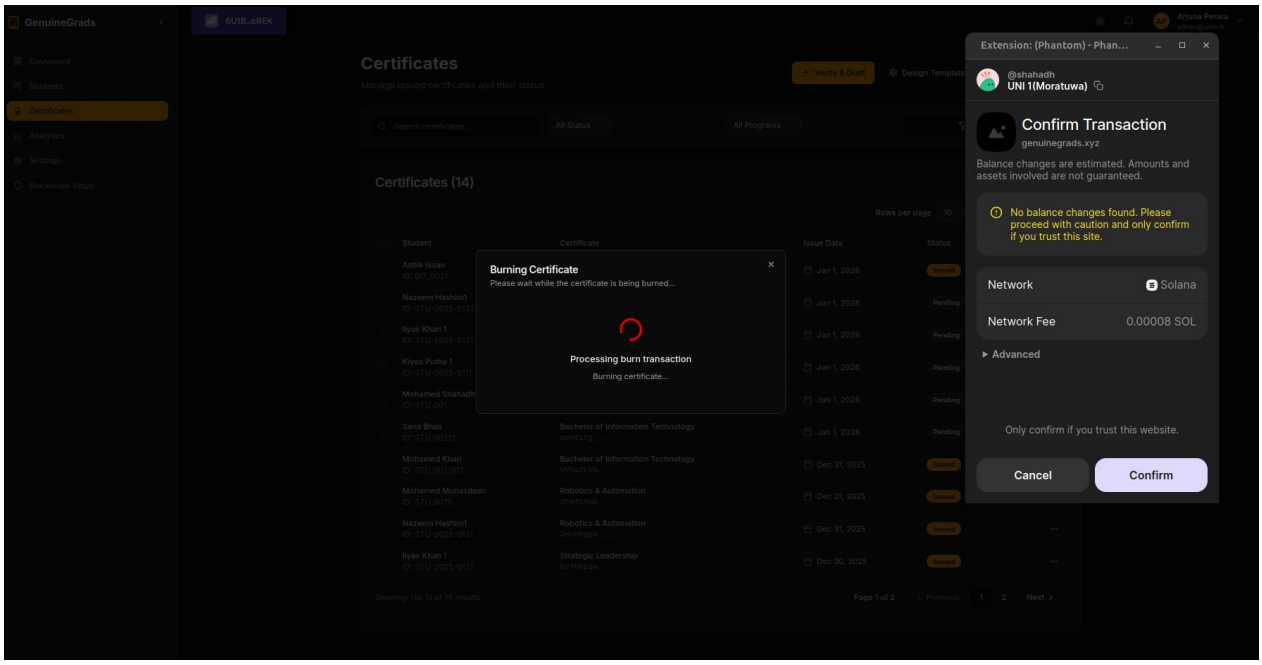


Figure 100: A49 - Wallet transaction was successfully shown for confimation.

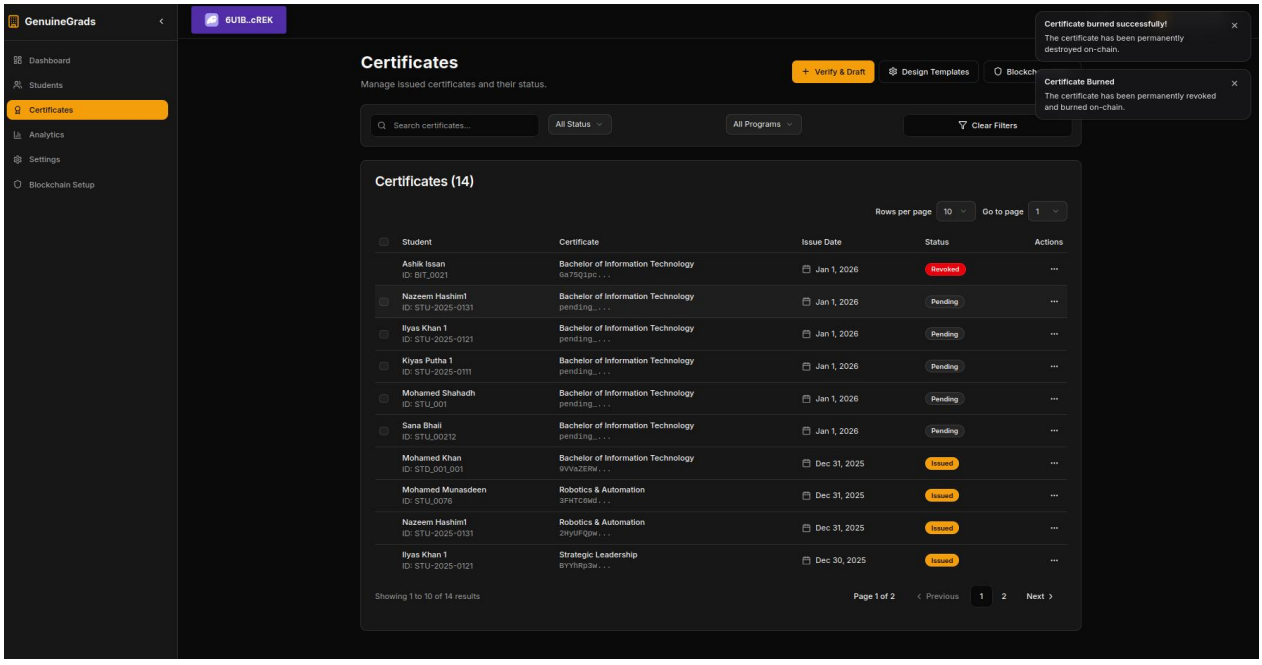


Figure 101: A50 - System displayed “Certificate burned successfully” notification.

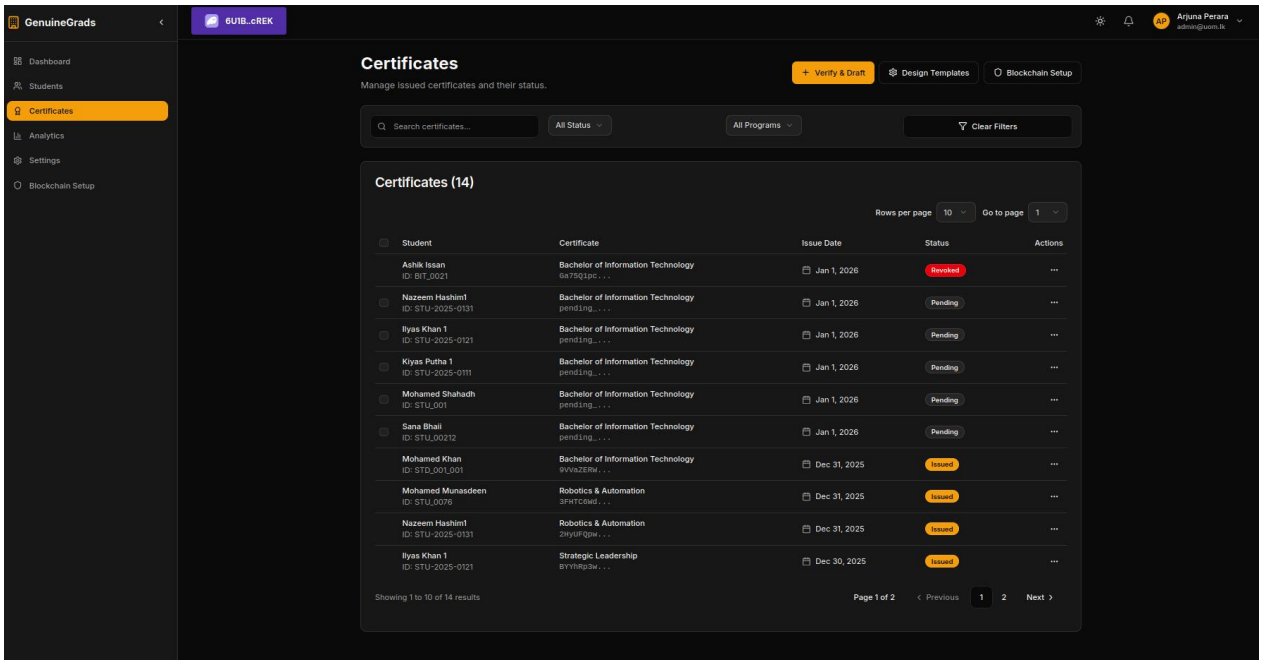


Figure 102: A51 - Certificate status changed to Revoked.

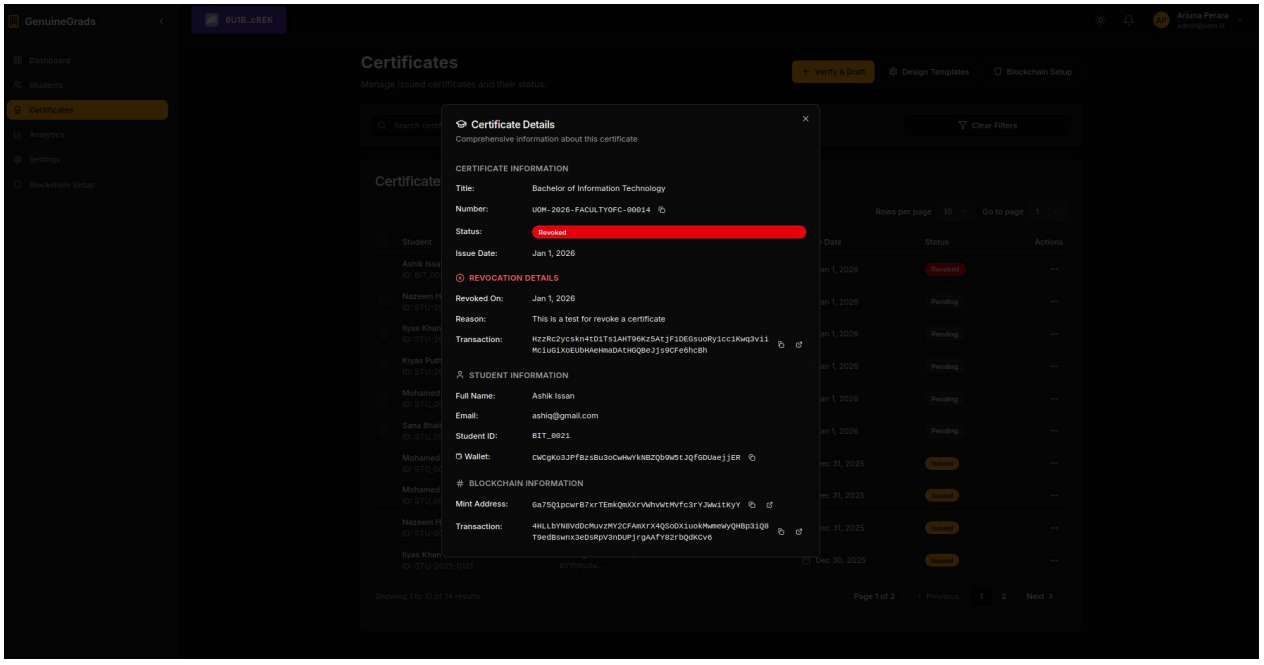


Figure 103: A52 - Revocation reason, date, and blockchain transaction hash were recorded and displayed in certificate details.

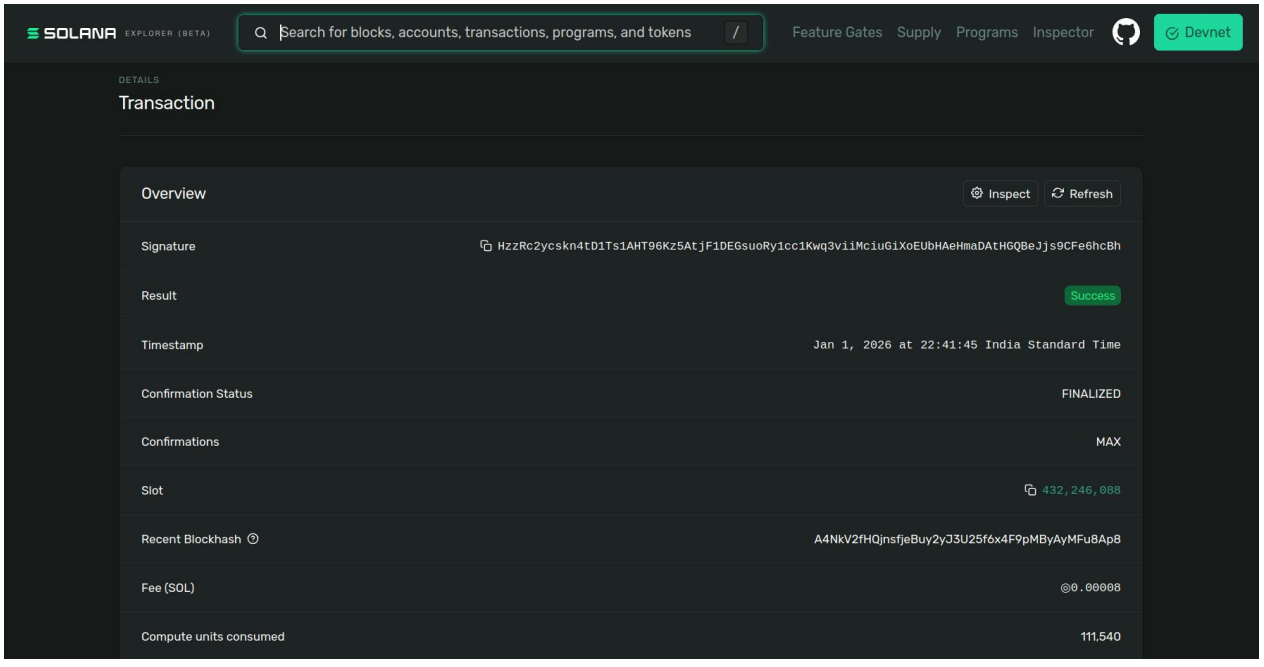


Figure 104: A53 - Solana Explorer view of burned transaction

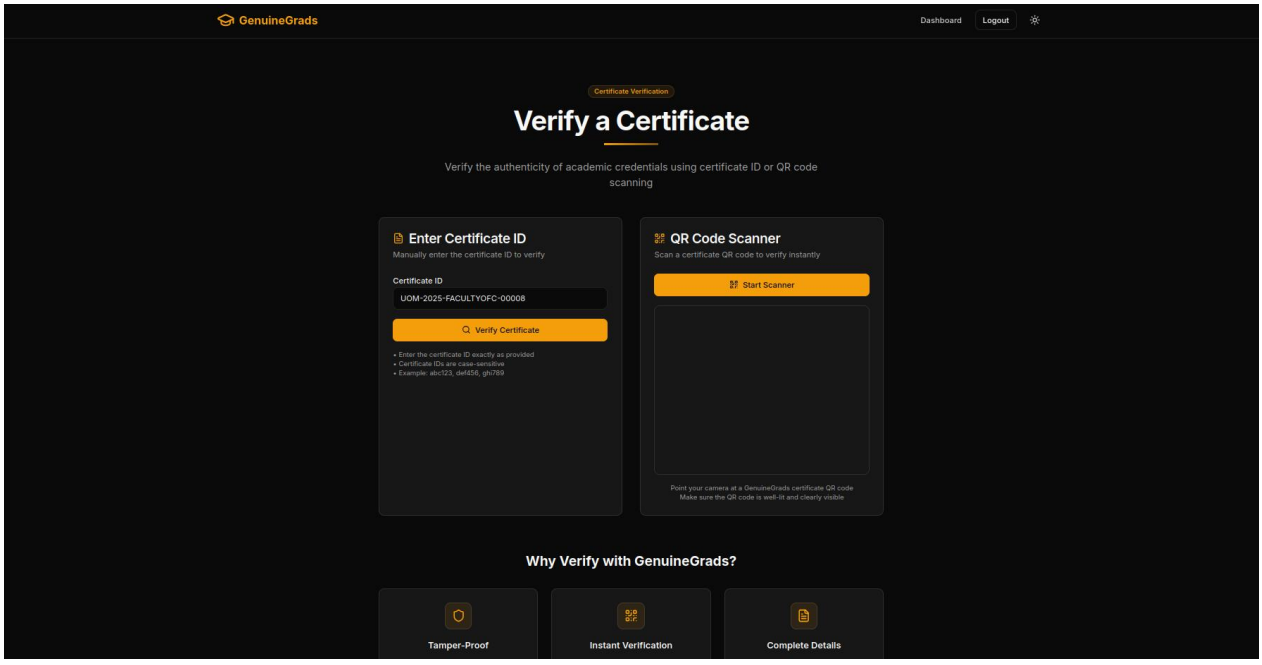


Figure 105: A54 - Verification Portal landing page with Certificate ID entry and QR Scanner option.

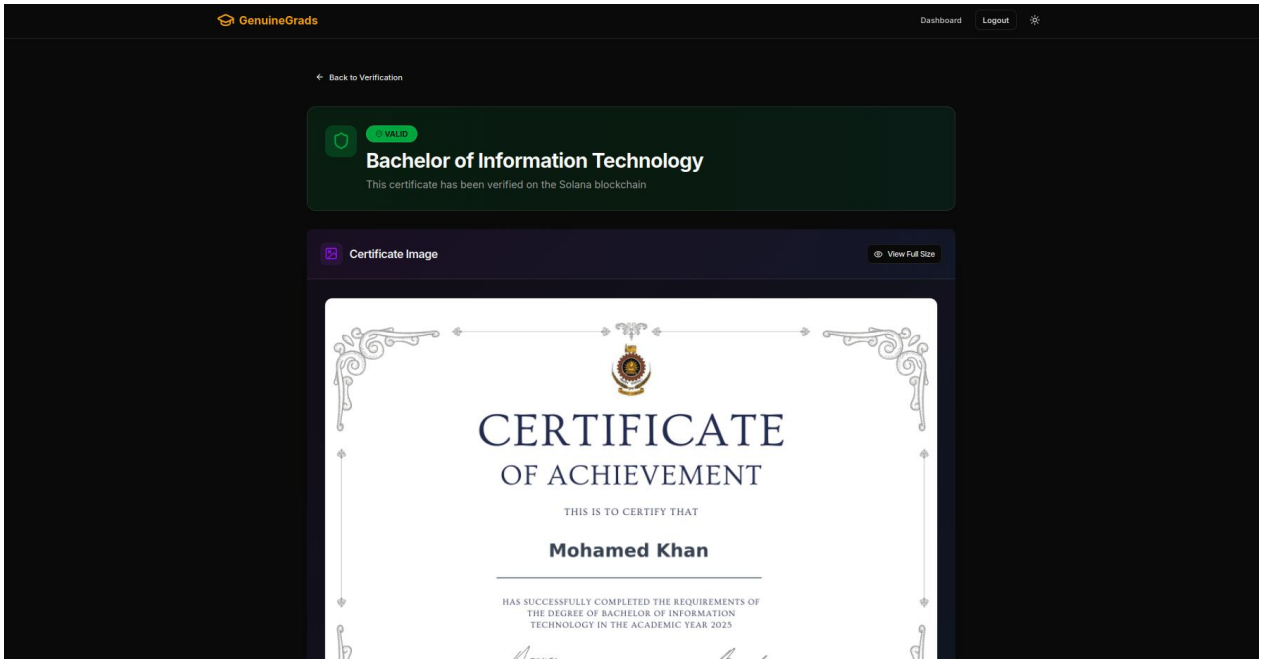


Figure 106: A55 - Valid Certificate view showing "VALID" status, Certificate Image, and Achievement list 01

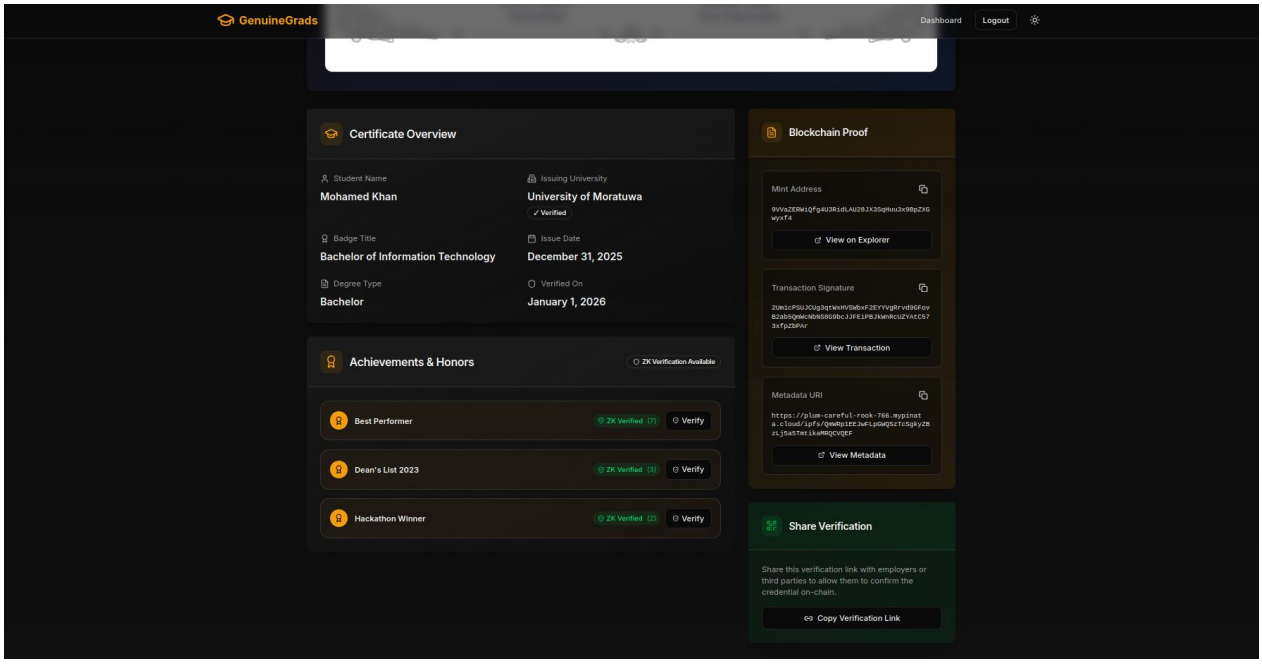


Figure 107: A56 - Valid Certificate view showing "VALID" status, Certificate Image, and Achievement list 02

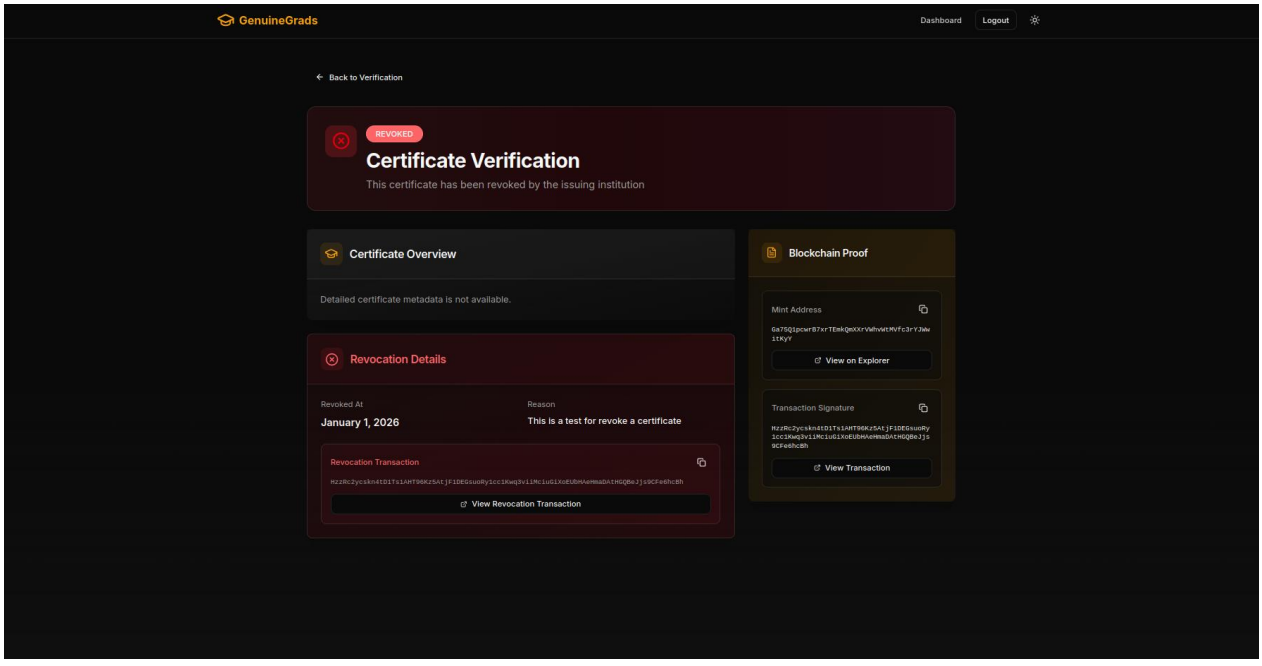


Figure 108: A57 - Revoked Certificate view showing "REVOKED" status, Revocation Date, and Reason.

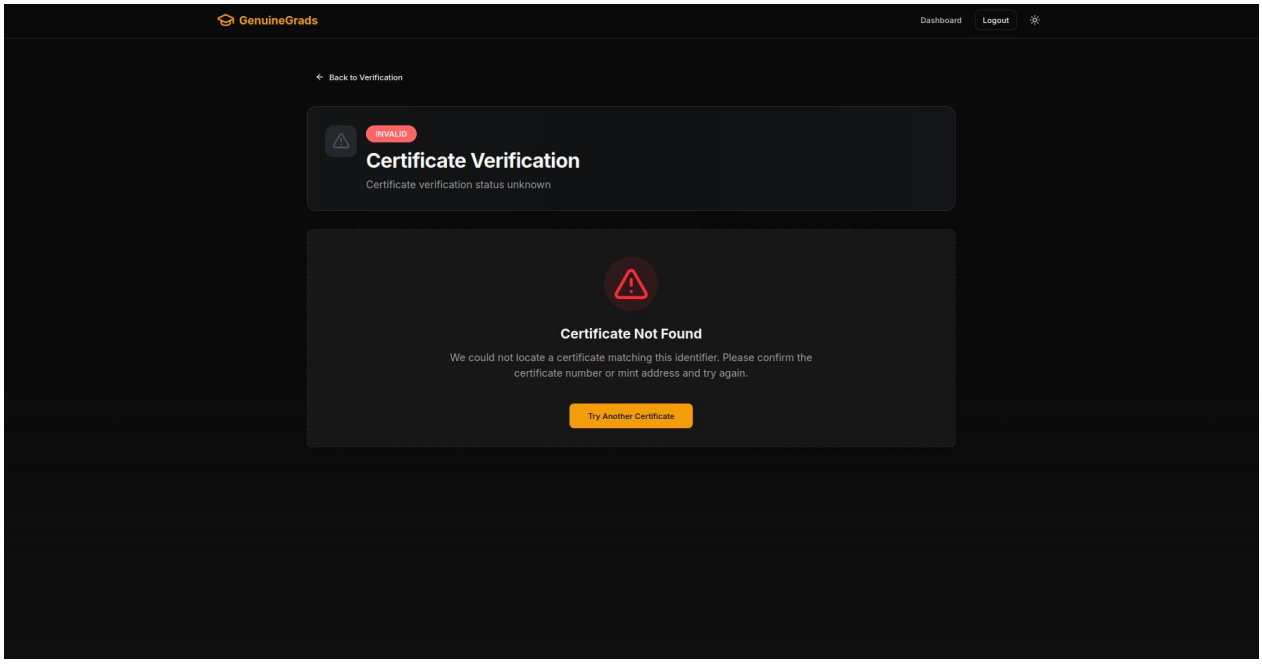


Figure 109: A58 - "Certificate Not Found" error page for invalid identifier input

APPENDIX B: USER INTERFACES

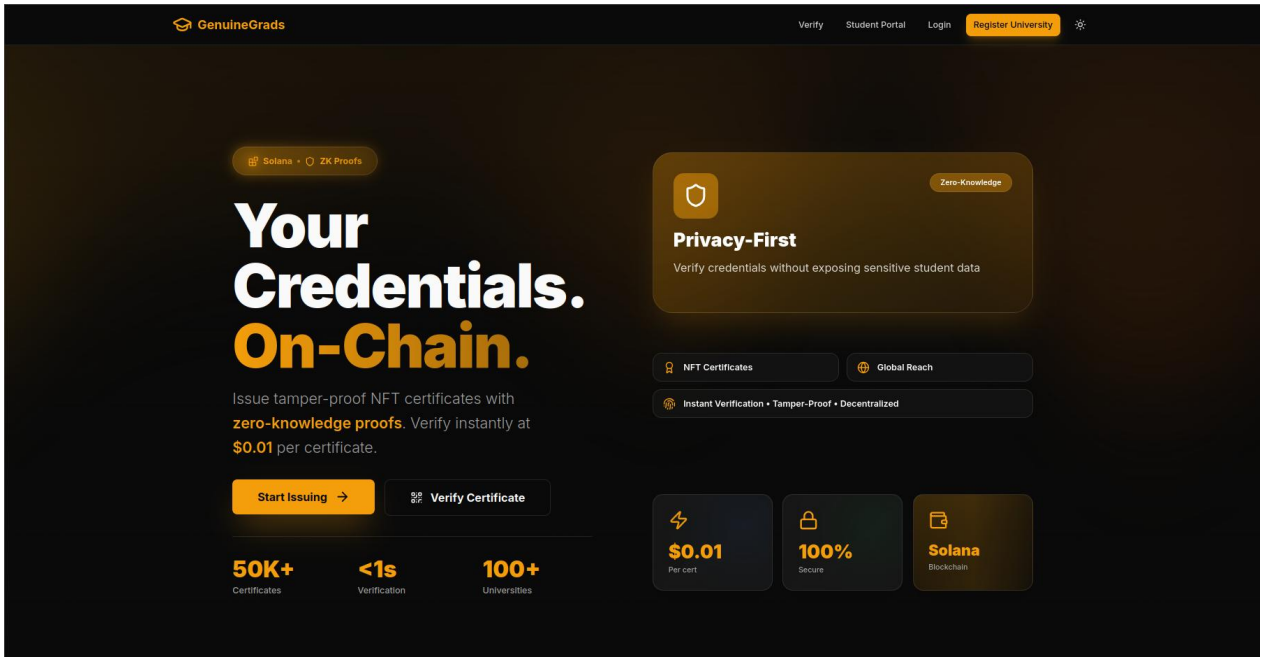


Figure 110: Landing Page

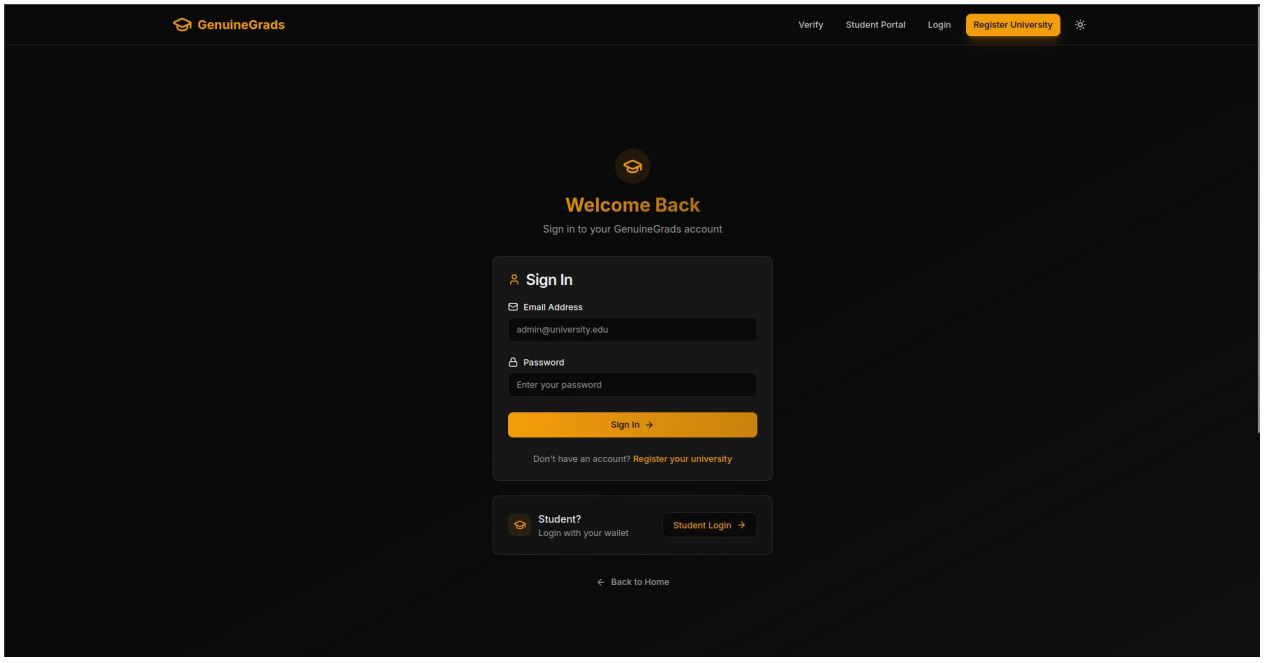


Figure 111: Admin/Super admin Login Page

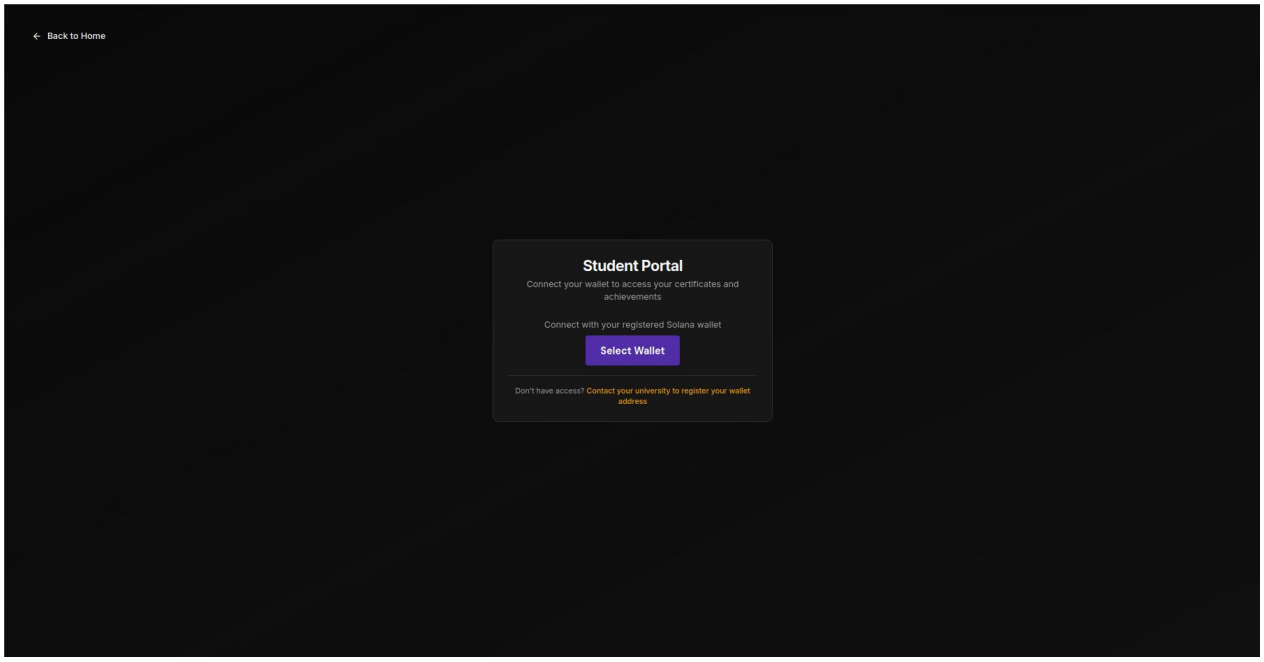


Figure 112: Student Portal Login Page

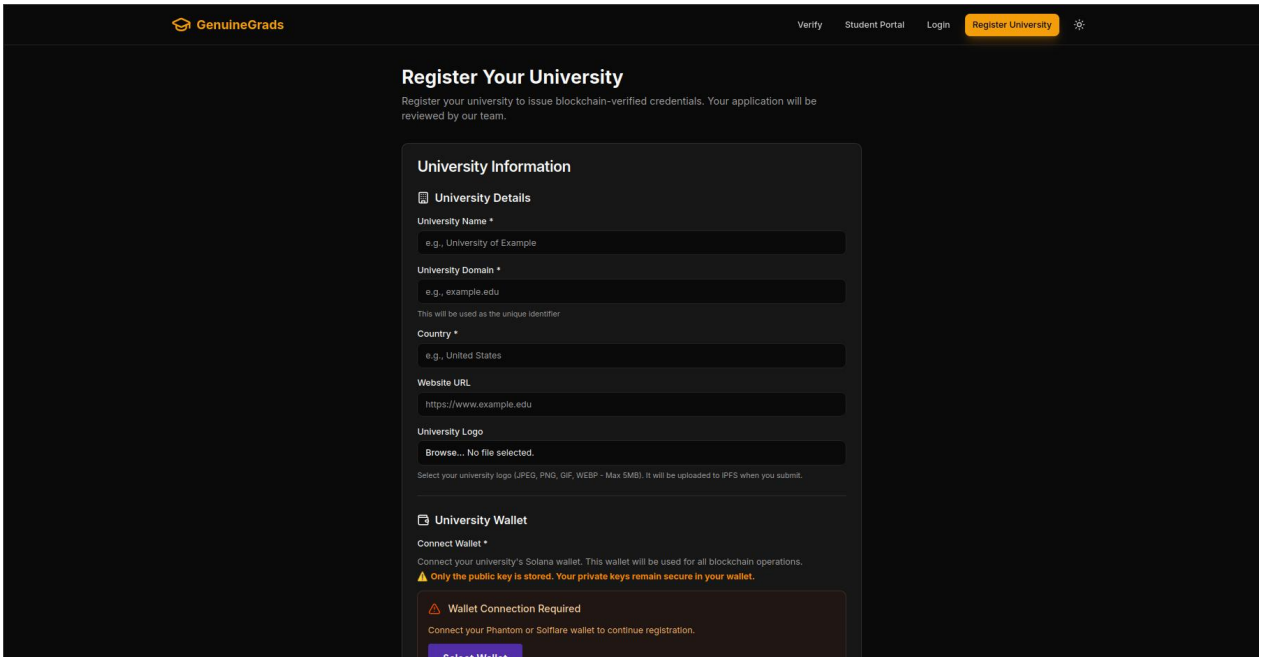


Figure 113: University Registration Page

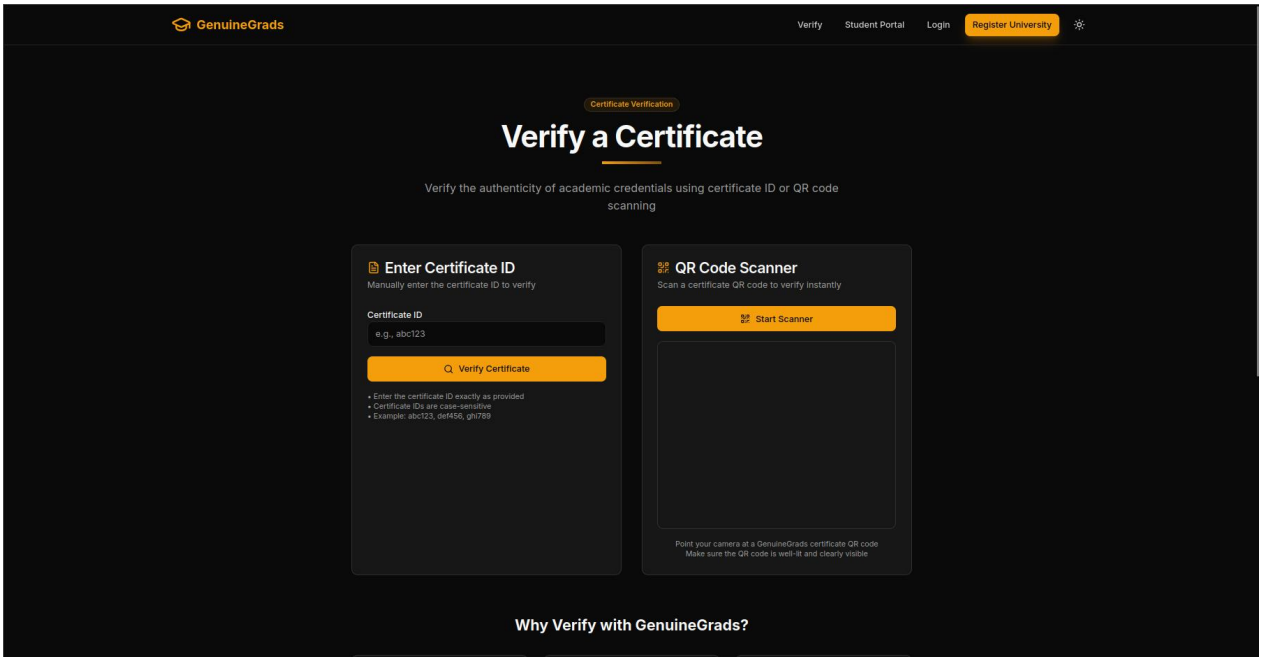


Figure 114: Verify a Certificate Page

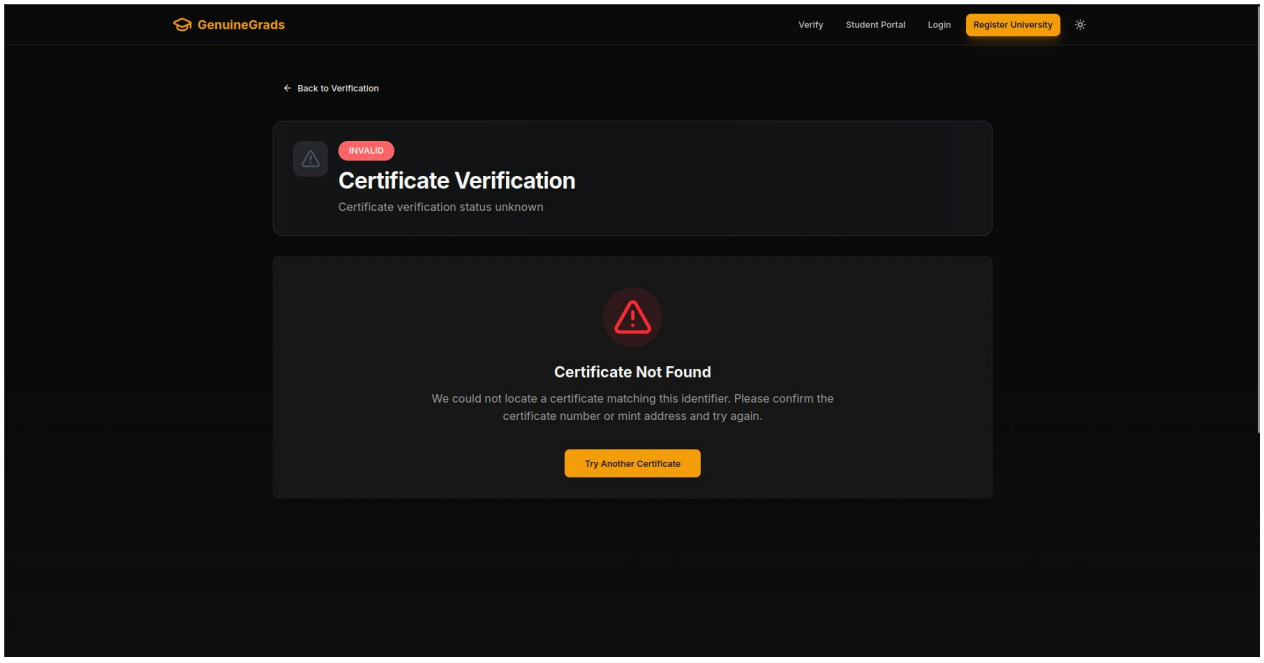


Figure 115: Certificate Not Found Page

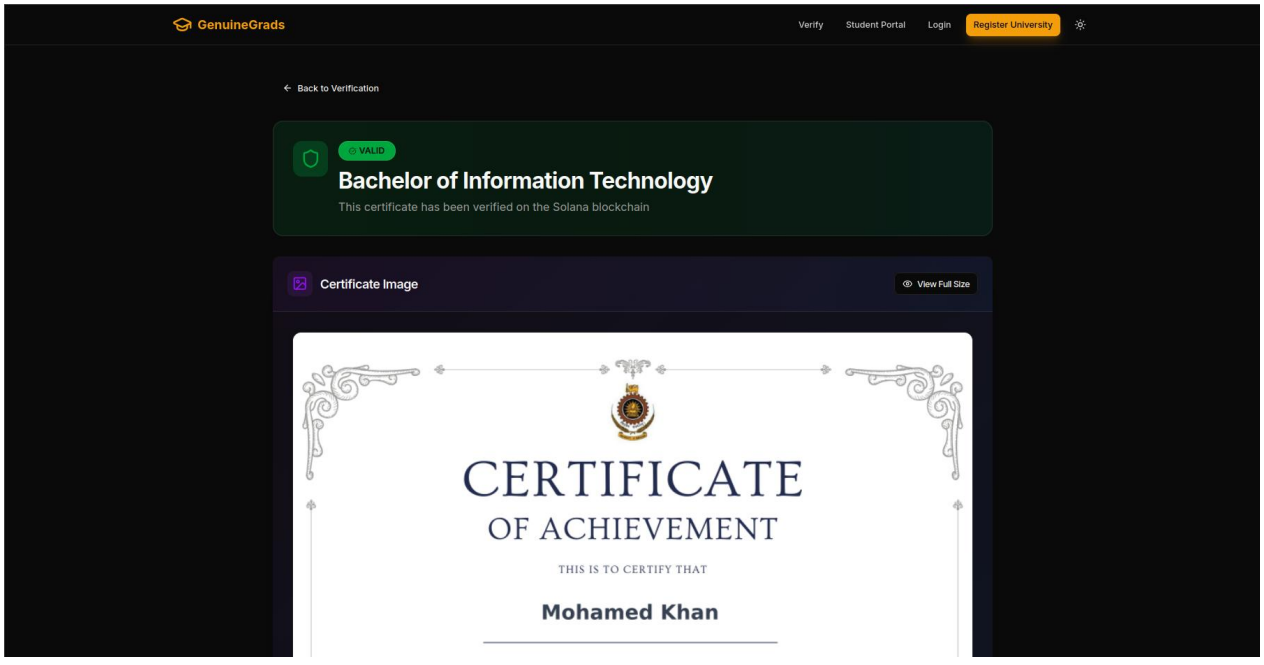


Figure 116: Valid Certificate Showing Page 1

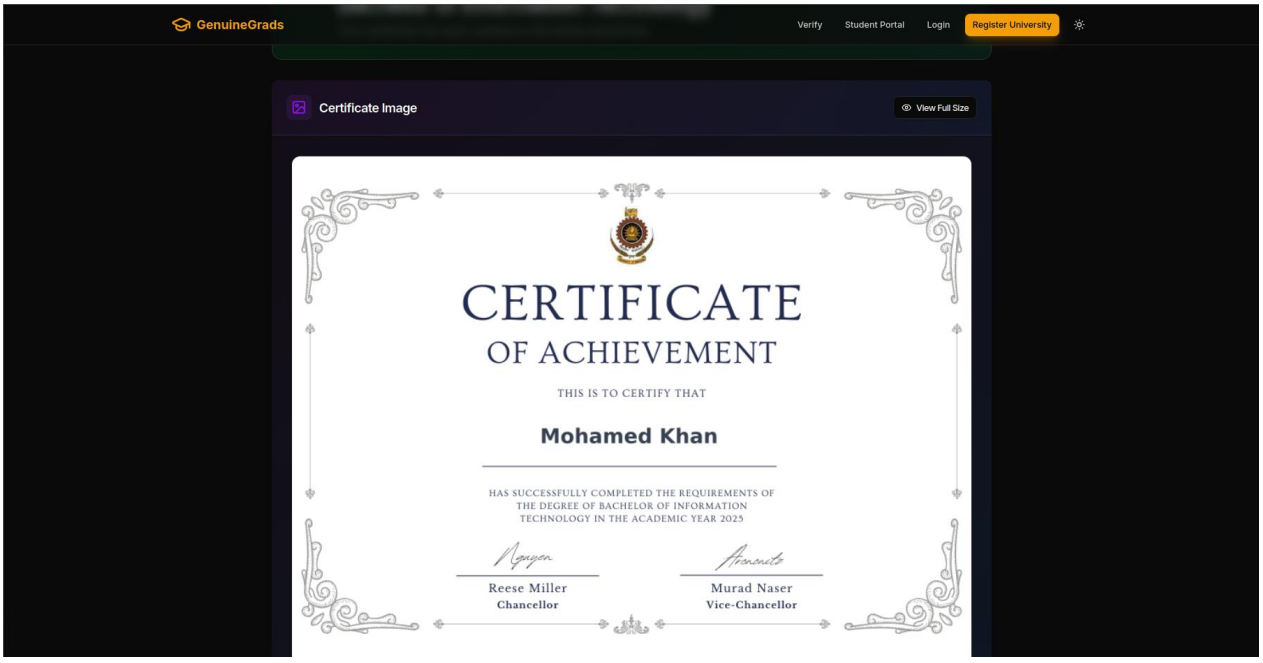


Figure 117: Valid Certificate Showing Page 1

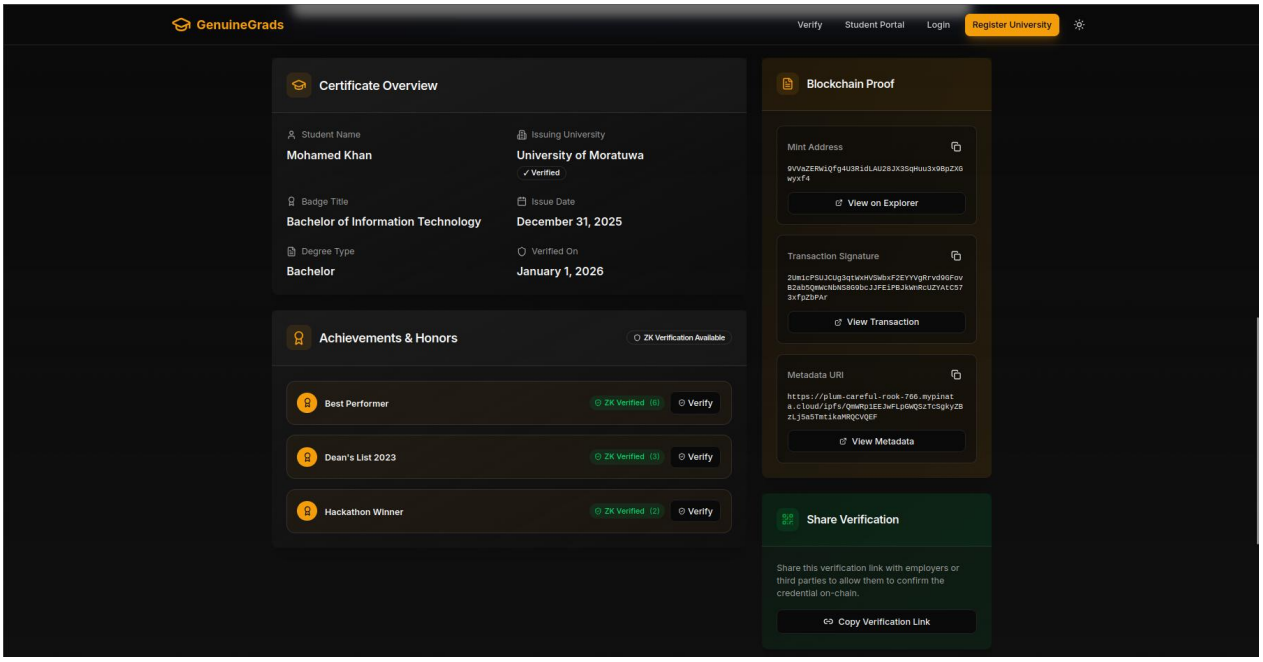


Figure 118: Valid Certificate Showing Page 1

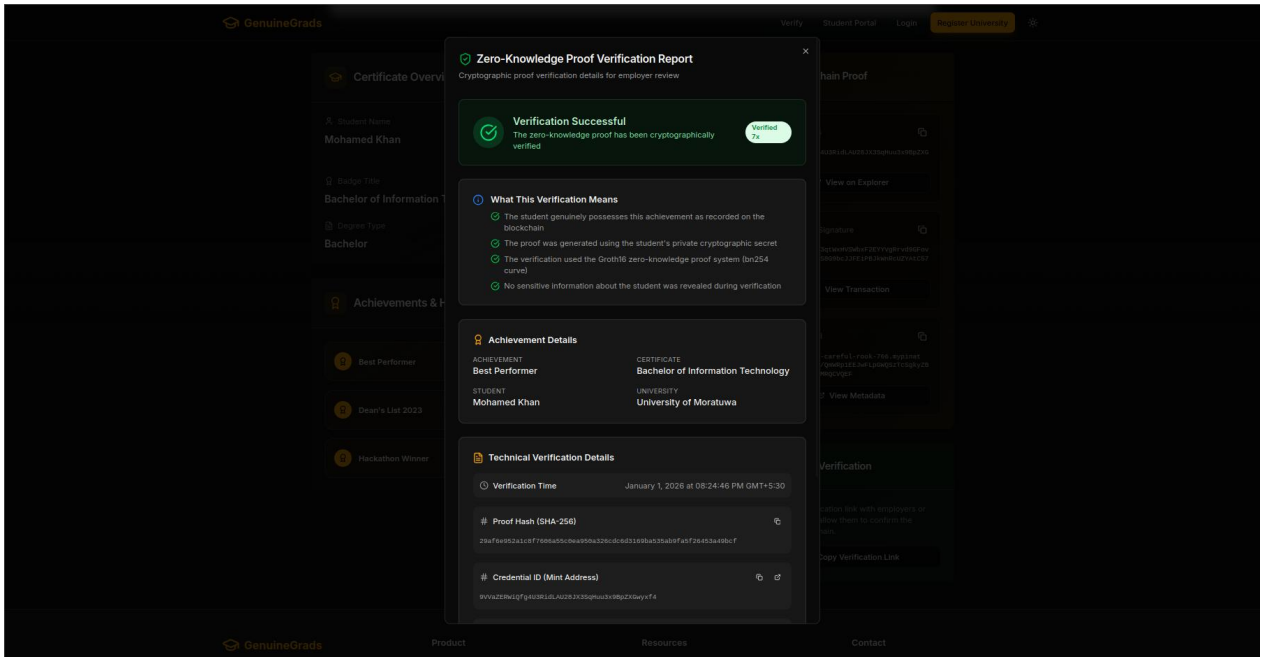


Figure 119: View ZKP Verfiication Results Modal

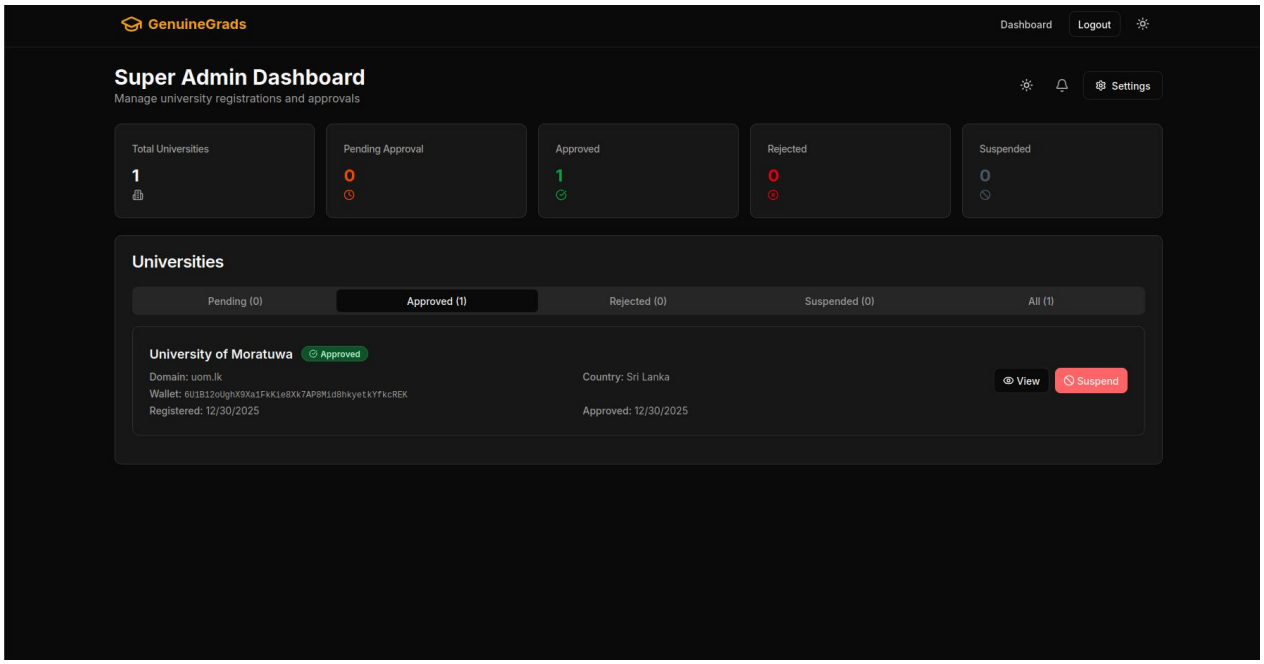


Figure 120: Super Admin Dashboard

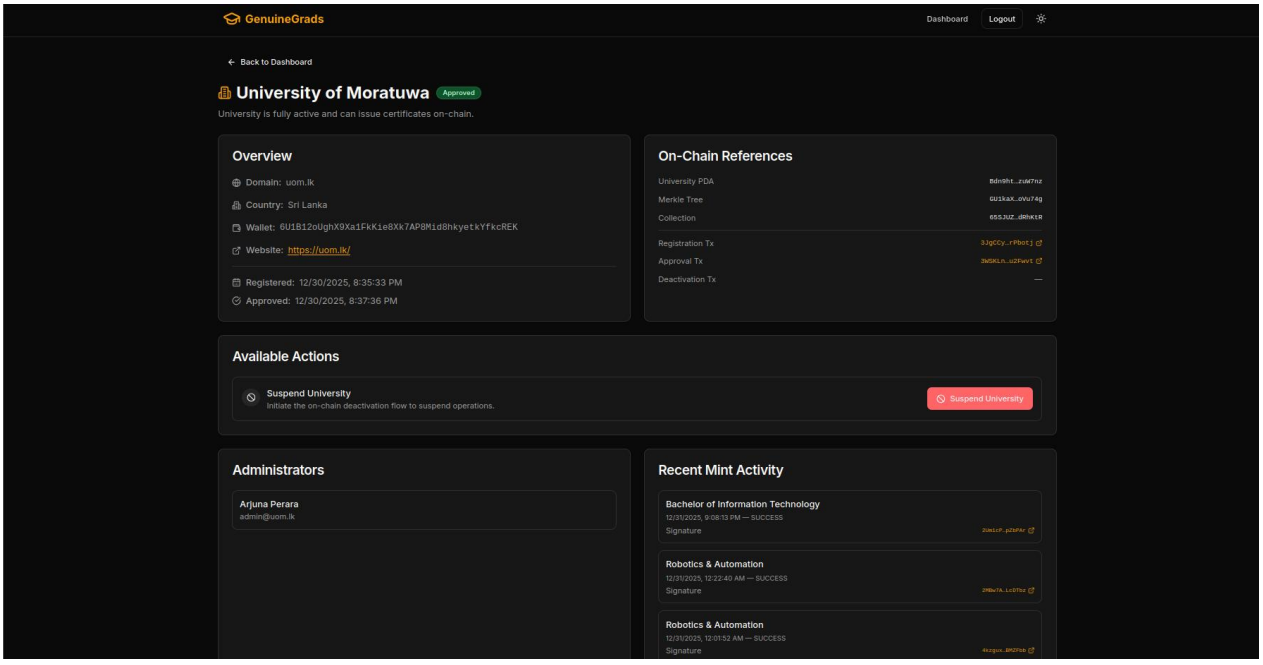


Figure 121: View Registered University Details Page

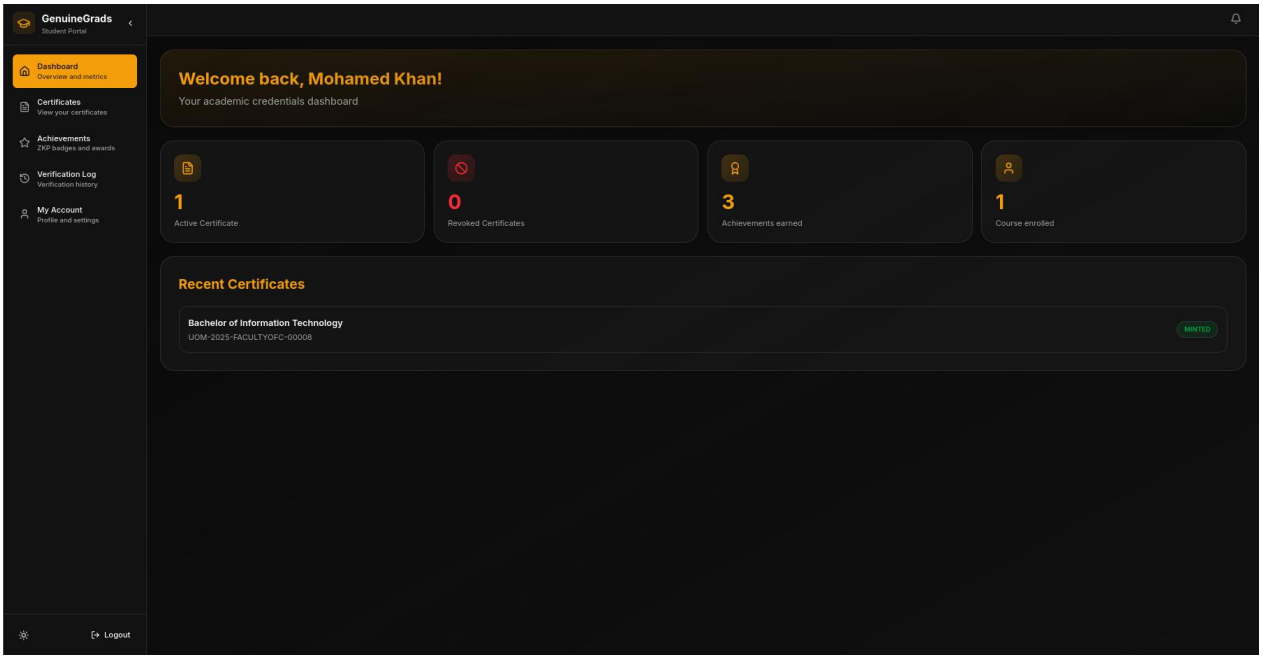


Figure 122: Student Dashboard

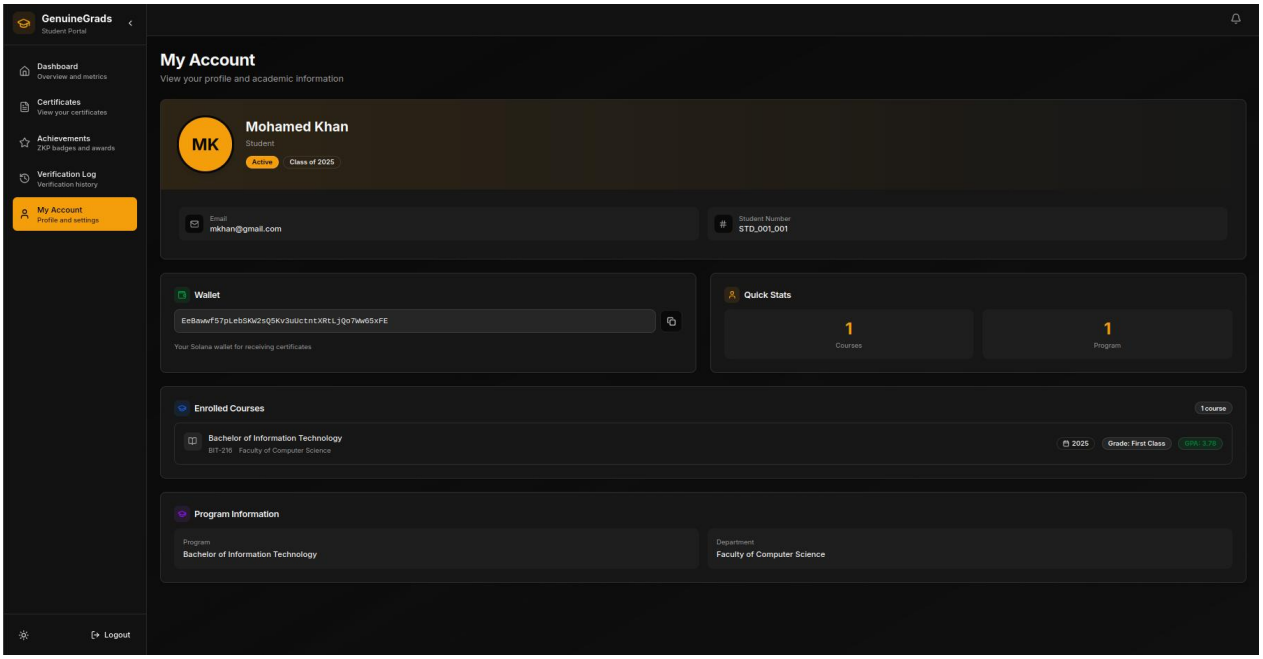


Figure 123: Student My Account Page

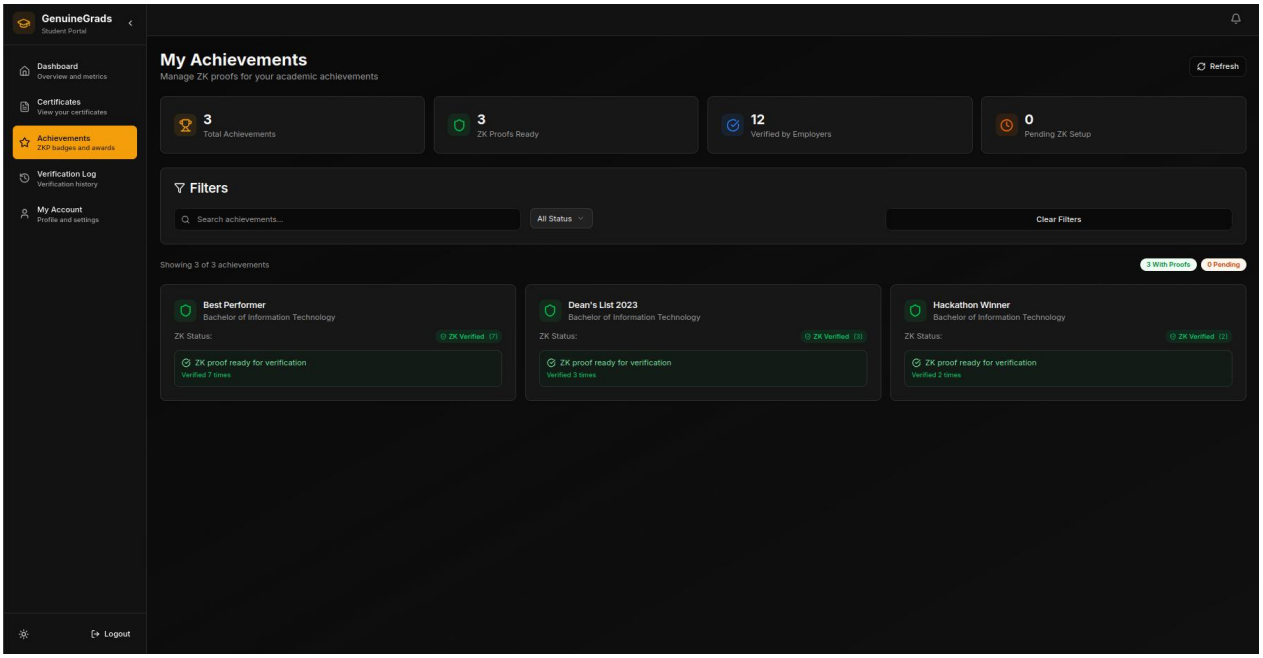


Figure 124: Student My Achievements Page

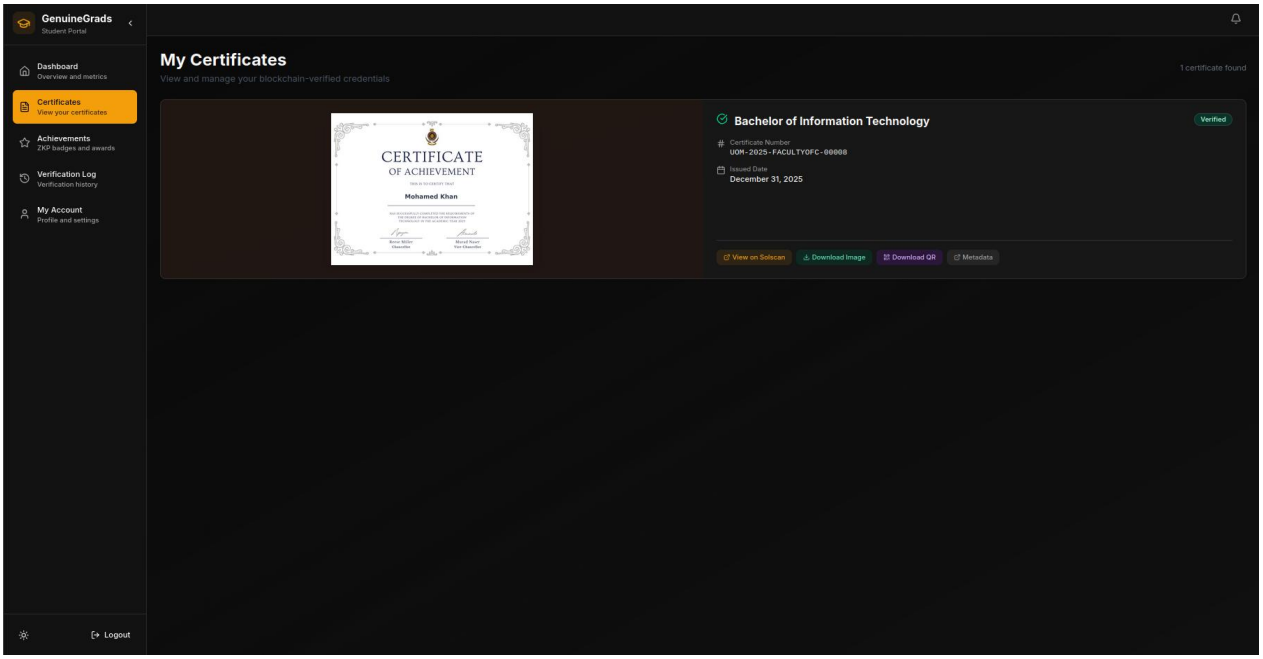


Figure 125: Student My Certificate Page

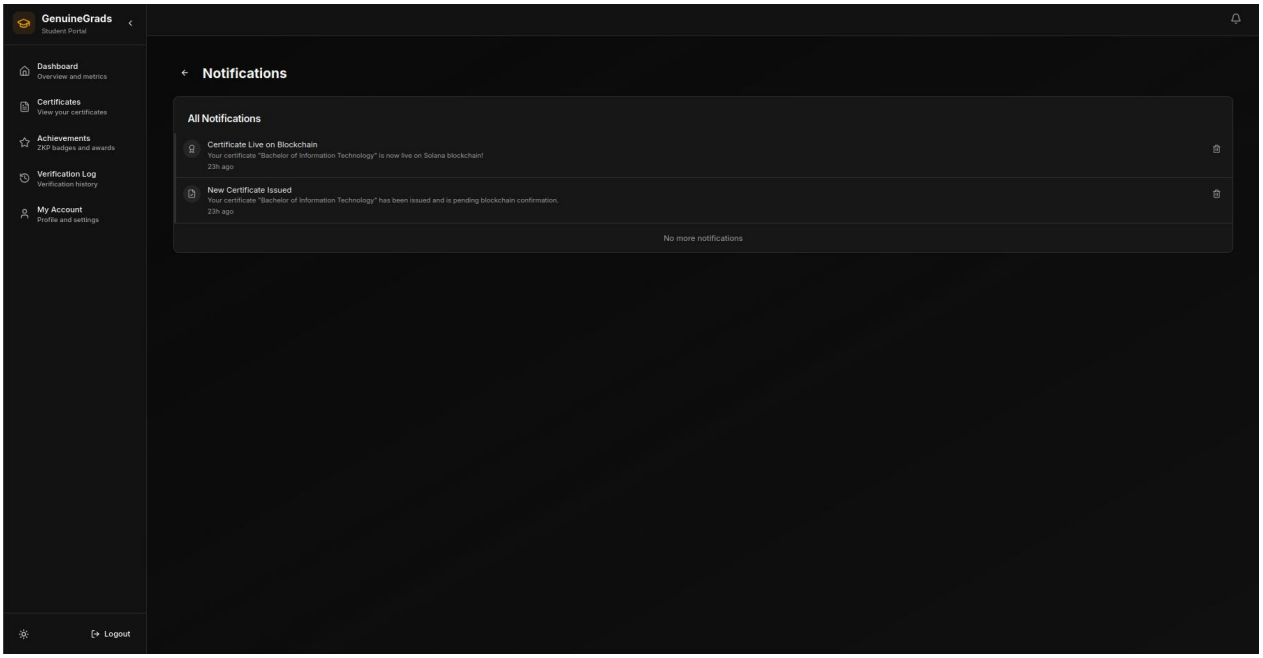


Figure 126: Student Notifications Page

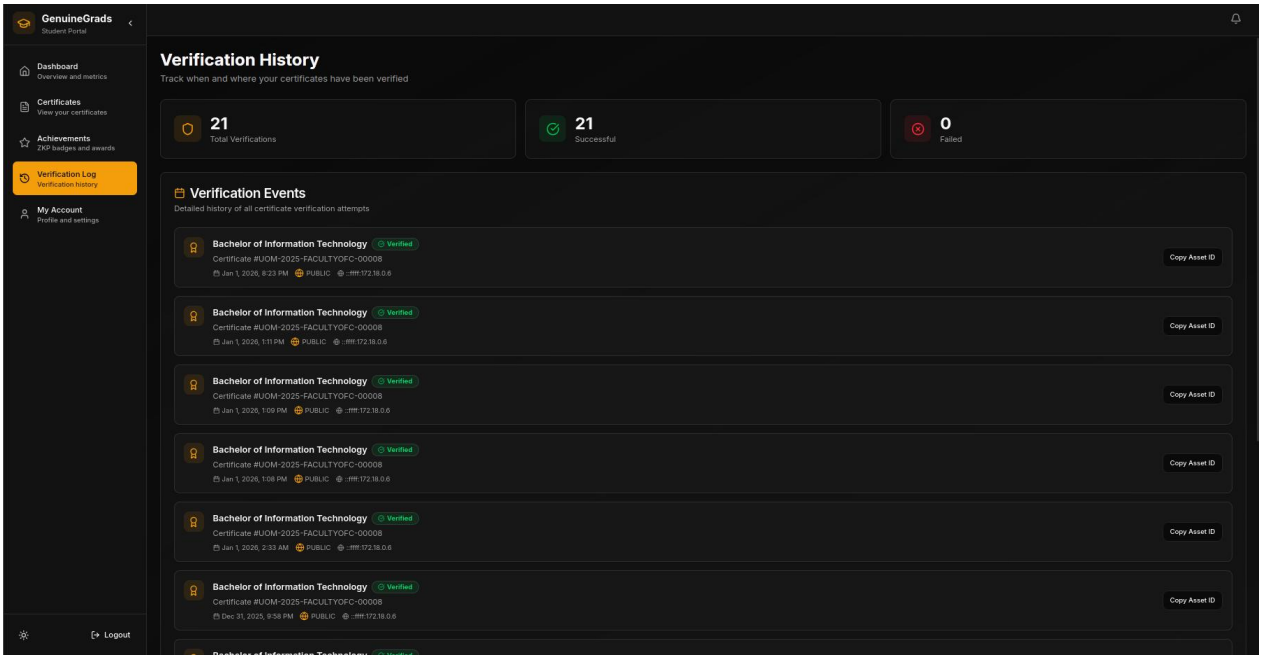


Figure 127: Student Verification Logs Page

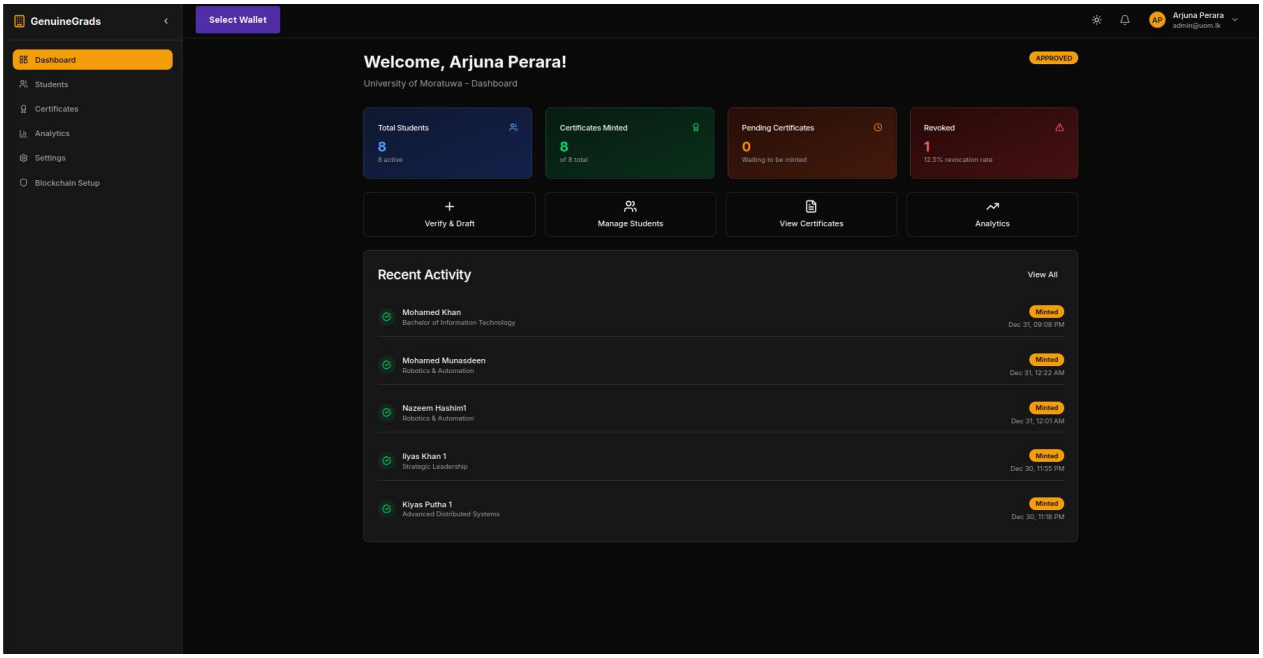


Figure 128: Admin Dashboard Page

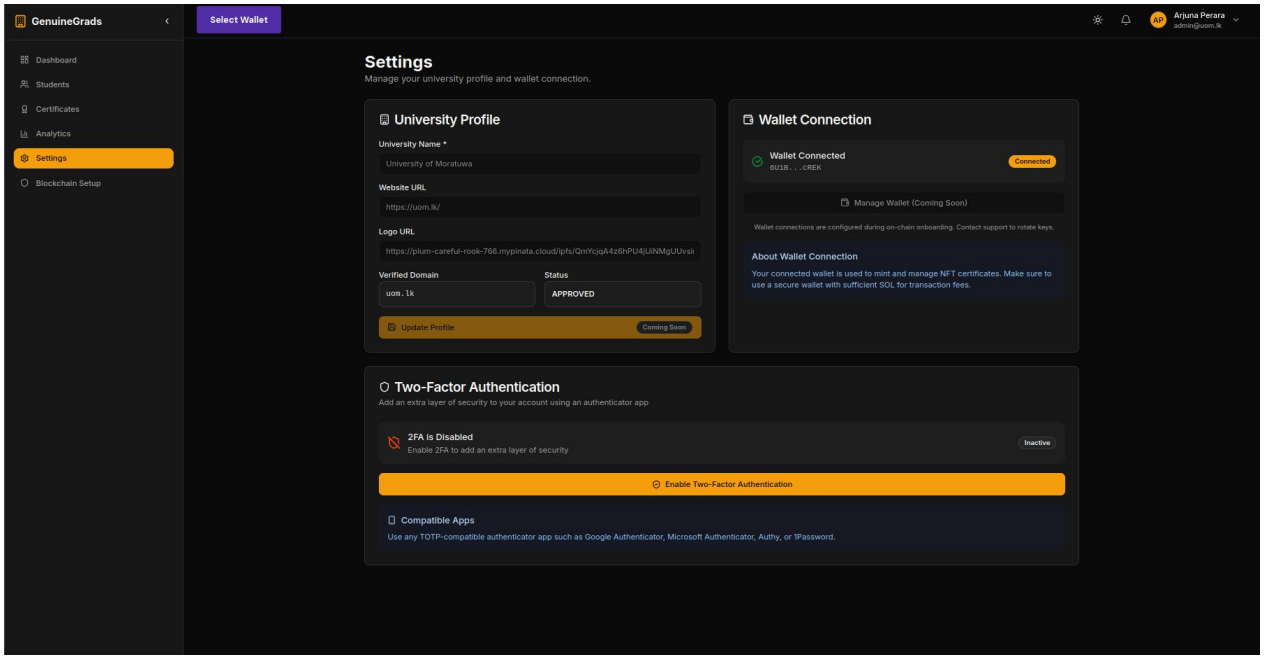


Figure 129: Admin Settings Page

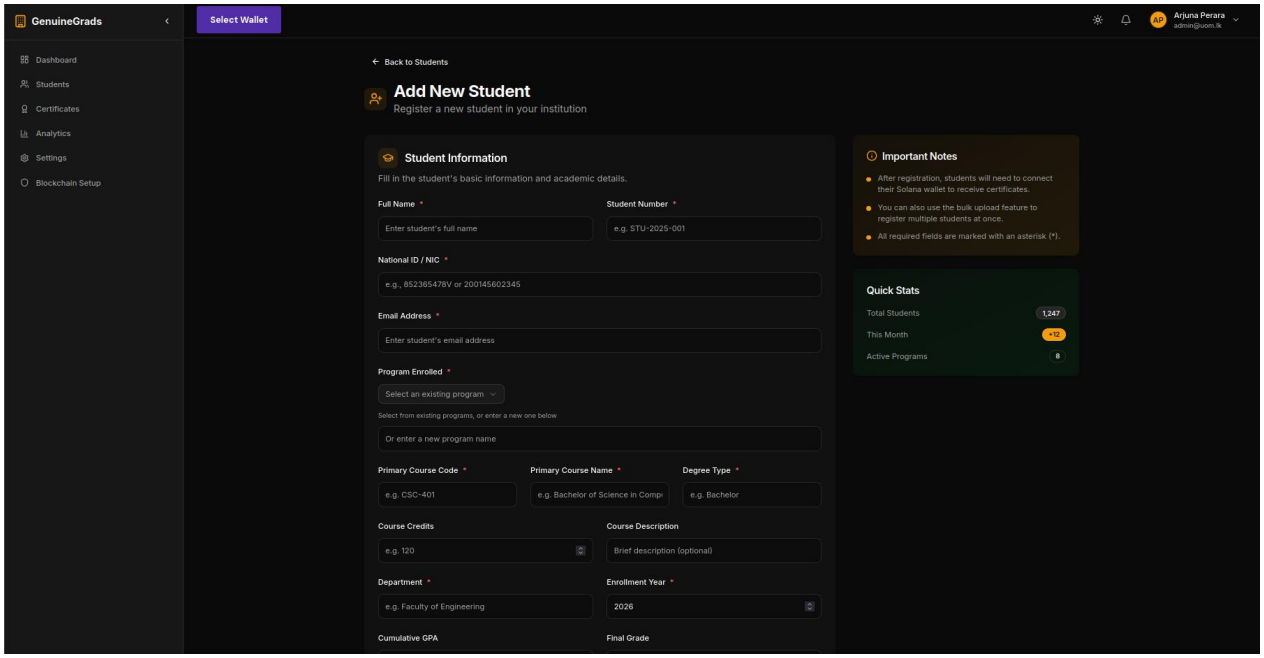


Figure 130: Admin Add Single Student Page

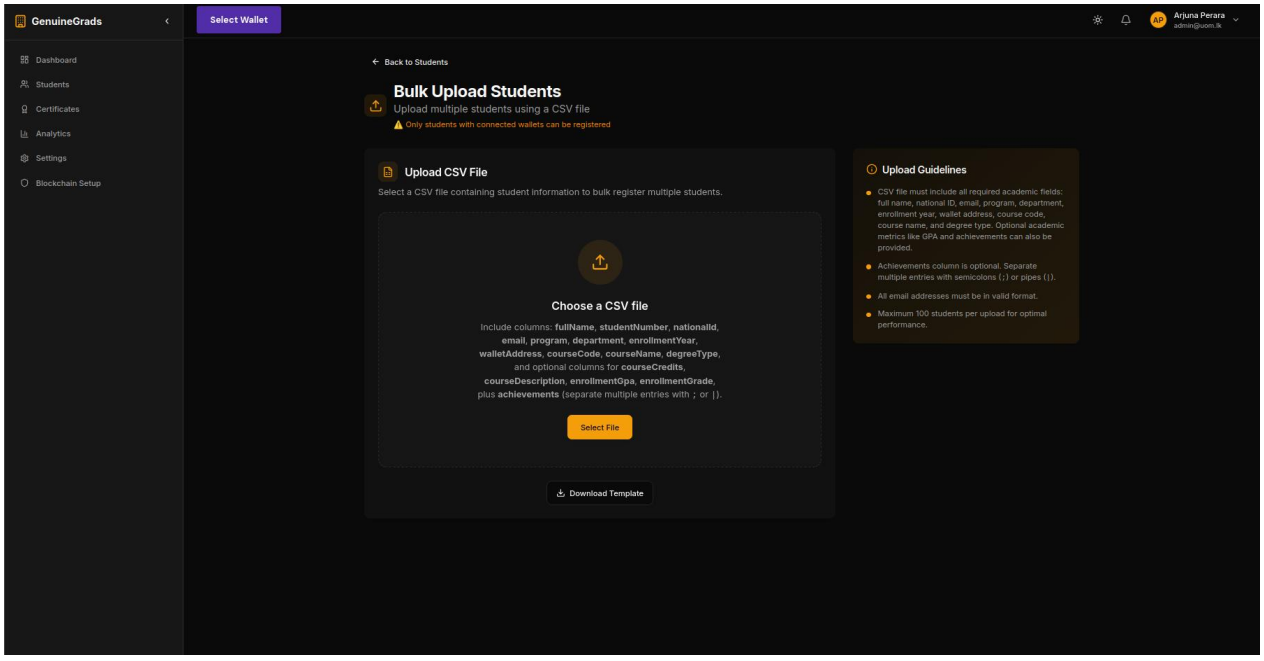


Figure 131: Student Bulk Upload Page

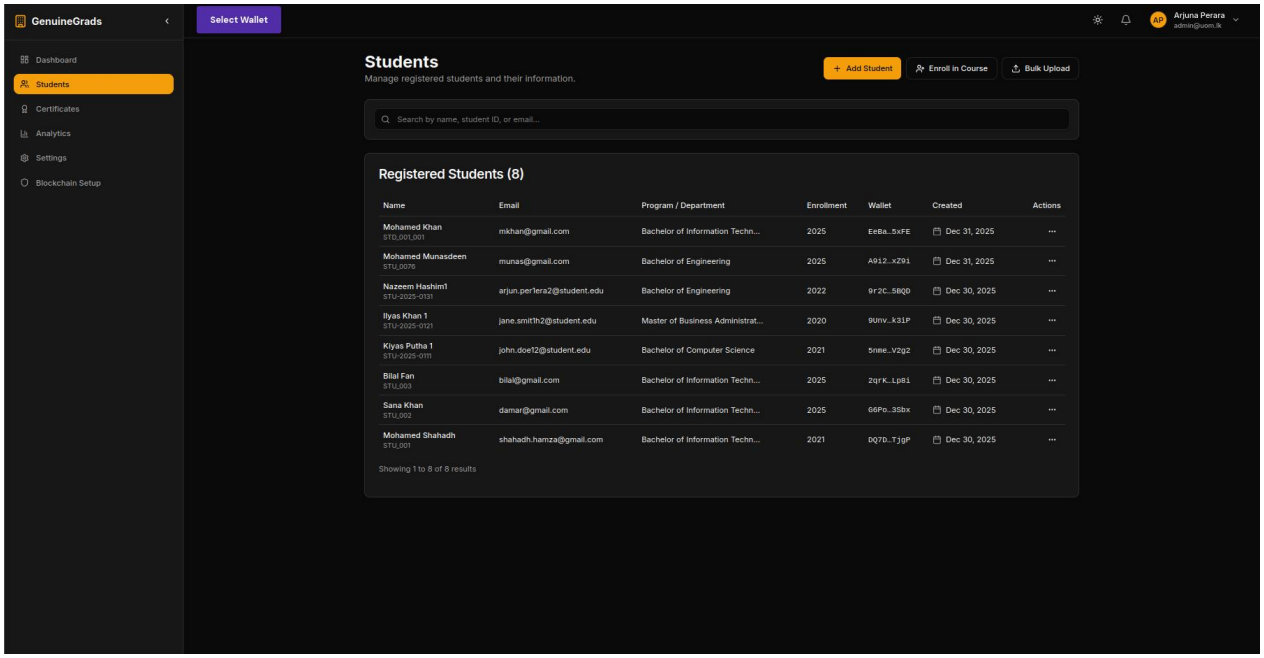


Figure 132: Admin Student Listing Page

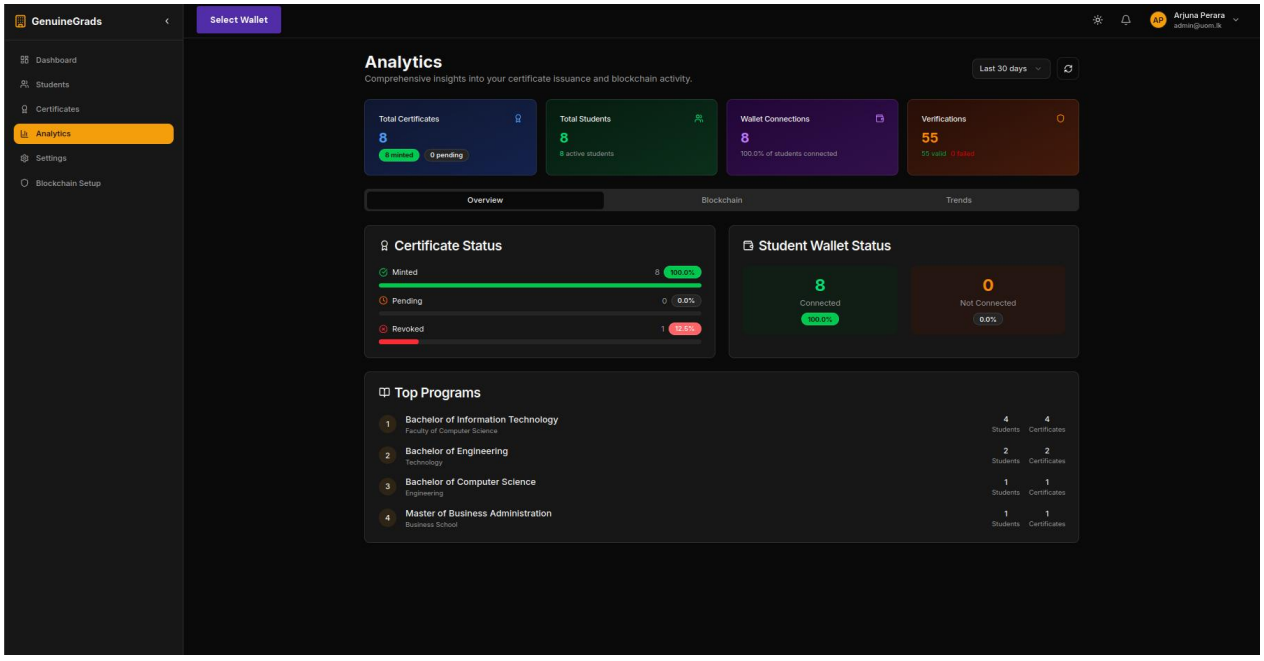


Figure 133: Admin Analytics Page

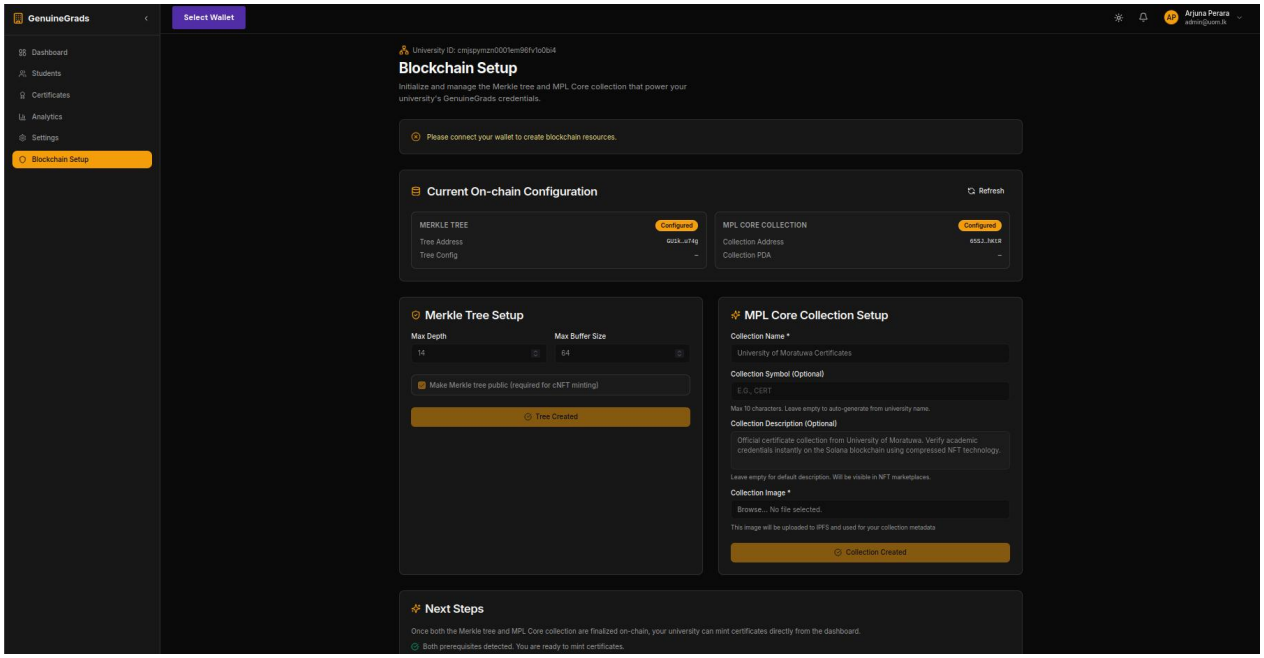


Figure 134: Admin Blockchain Setup Page

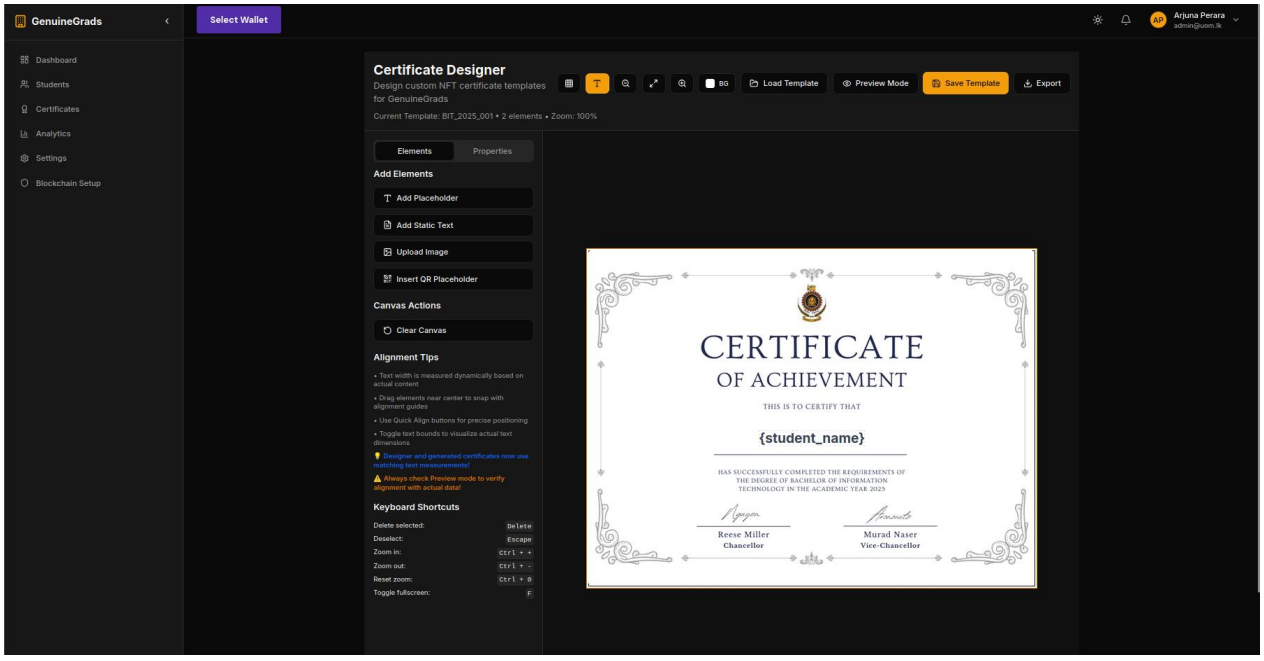


Figure 135: Admin Certificate Designer Page

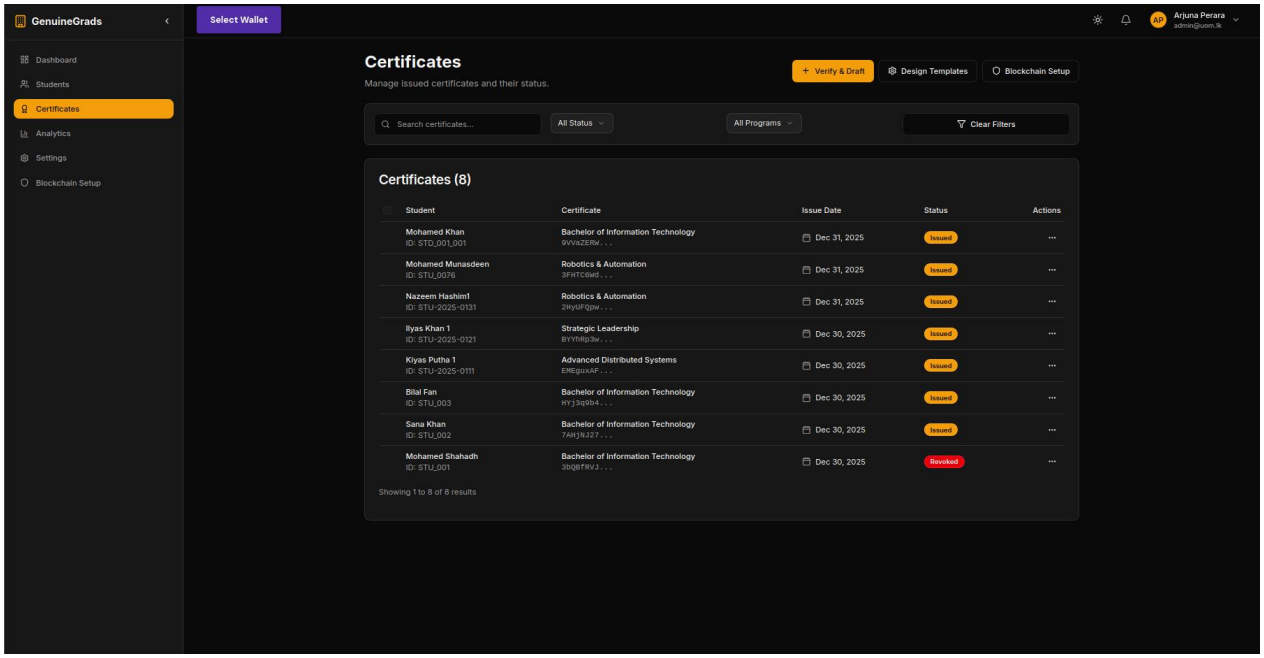


Figure 136: Admin Certficate Listing Page