Software Design Specification

AI and Blockchain-Based Certificate Verification System

Project Code:

BCV-AI-(25–26)

Internal Advisor:

Mr. Muhammad Fahad

Project Manager:

Dr. Muhammad Ilyas

Project Team:

M. Nouman Riaz BSCS51F21R003 (Team Lead)

Submission Date:

December 15, 2025

_____

Project Manager's Signature

Document Information

Customer	UOS – Department of Computer Science
Project	AI and Blockchain-Based Certificate Verification System
Document	Software Design Specification
Document Version	1.0
Identifier	BCV-AI-(25-26)
Status	Draft
Author(s)	Muhammad Nouman Riaz
Approver(s)	Dr. Muhammad Ilyas
Issue Date	December 15, 2025
Document Location	
Distribution	1. Project Advisor – Mr. Muhammad Fahad 2. Project Manager – Dr. Muhammad Ilyas

Definition of Terms, Acronyms and Abbreviations

This section should provide the definitions of all terms, acronyms, and abbreviations required to interpret the terms used in the document properly.

SDD	Software Design Specification
DD	Design Specification
AI	Artificial Intelligence
API	Application Programing Interface
SQL	Structured Query Language
OCR	Optical Character Recognition
SHA	Secure Hash Algorithm
JWT	JSON Web Token

Table of Contents

1. Introduction

1.1 Purpose of Document

This Software Design Specification document's goal is to give the AI and Blockchain-Based Certificate Verification System a thorough architectural and detailed design plan. It converts the specifications found in the Software Requirements Specification into a technical blueprint that directs the project's implementation, testing, and maintenance stages.

In order to guarantee that the finished product is safe, scalable, and satisfies all functional and non-functional requirements, this document outlines the system's software architecture, data structures, interfaces, and component design.

Intended Audience:

The following people are the target audience for this document

Developers: To understand the modules, coding standards, and system architecture needed for implementation.

Testers and QA engineers: They will create test cases and validation plans based on the data flow and system components.

Project Managers (Dr. Muhammad Ilyas): To monitor development and make sure the design complies with the project's schedule and scope.

Stakeholders: To obtain a high-level understanding of the technical solution in use.

Design Approach: The Object-Oriented Design (OOD) approach is used in this project. By encapsulating system entities (like Users, Certificates, and Blockchain Transactions) as objects, this method enables modular development . In order to integrate components like AI (OCR) and Blockchain (Smart Contracts), this encourages code reusability, simpler maintenance, and scalability [5].

1.2 Project Overview

By offering a tamper-proof verification method, the AI and Blockchain-Based Certificate Verification System is a safe, web-based platform created to detect fake or altered academic and professional degrees. Employers or individuals can instantly verify valid certificates registered by educational institutions through the system, which operates as a centralized digital solution.

Functionality relies on a two step verification process:

Registration:

Certificates are uploaded by authorized institutions. The system creates a distinct SHA-256 hash after extracting text data (such as student name, ID, and course) using optical character recognition (OCR). Smart contracts are used to permanently store this hash on the Ethereum blockchain. [4]
Verification:
The system recalculates the hash and verifies authenticity by comparing it to the unchangeable blockchain record when a verifier uploads a document.

Design Methodology:

The program uses a hybrid architecture that combines decentralized blockchain technology with web development frameworks

Frontend: To guarantee compatibility with desktop and mobile browsers, the user interface is built using HTML, CSS, and JavaScript.

Backend: The Python Flask framework, which controls API endpoints and routing, handles the server-side logic [7].

AI Component: Text extraction from certificate images is automated by integrating Tesseract OCR [6].

Blockchain Layer: To ensure data immutability, the system stores certificate hashes using Solidity smart contracts installed on the Ethereum Test Network [2][3].

Data Storage: Only the non-sensitive certificate hashes are kept on the blockchain, while sensitive user credentials and logs are kept in a local database (SQLite/MongoDB).

1.3 Scope

The project's boundaries are outlined in this section, along with the features that are and are not part of the development.
In-Scope (What the system will accomplish):
The following features are intended to be provided by the system:
Institution Registration: Allow verified Institutions to register and upload certificates.
Automated Extraction: Create a distinct SHA-256 digital fingerprint (hash) by automatically extracting text from certificate images using AI-based OCR.
Blockchain Storage: To avoid data manipulation, safely store the generated certificate hashes on the Ethereum blockchain.
Instant Verification: Give general users and employers the option to upload certificates and get instant feedback on their legitimacy ("Verified" or "Fake/Modified").
Web Interface: Offer a straightforward browser-based interface that can be accessed on Windows, Linux, and macOS using Chrome.

Out-of-Scope: (What the system is unable to perform)

The following features are specifically not included in the project's current phase:

National Database Integration: The system won't work with national government databases like NADRA or HEC (Higher Education Commission).

Real Cryptocurrency Transactions: No actual financial cryptocurrency transactions will take place; the system will only function on the Ethereum testnet.

Mobile Application: A dedicated native mobile app is not included; development is restricted to a web-based version.

Data Modification: The system will not permit the editing or removal of a particular blockchain entry once it has been stored on the blockchain.

2. Design Considerations

This section discusses particular technical problems and limitations that need to be fixed in order to produce a strong architectural solution.

2.1 Assumptions and Dependencies

The following technical presumptions and dependencies are essential for the system design and implementation, even though the operational assumptions have been recorded in the SRS.

Hosting Environment Compatibility:

The Tesseract OCR engine and Python-based blockchain libraries like web3.py are among the necessary backend dependencies that the system design assumes can be installed and run in the chosen hosting environment. The OCR component may need to undergo architectural changes, such as separation into a separate processing service, if the hosting environment places restrictions on binary execution or processing resources.

Blockchain Network Availability:

The design makes the assumption that the Ethereum test network (Goerli or Sepolia) and the testnet-funded accounts needed to carry out smart contract transactions will remain accessible. Real financial transactions are not involved because the system only runs on a testnet; however, transaction execution is dependent on testnet resources and network stability.

Data Standardization:

Before producing the SHA-256 hash, the system design presupposes that OCR-extracted text must go through stringent normalization, such as trimming whitespace, consistent casing, and structured formatting. Even small discrepancies in the extracted data could lead to verification failure because hash generation is totally dependent on the input. Reliable and repeatable certificate verification is directly supported by this assumption.

2.2 Risks and Volatile Areas

The system design needs to actively address the following potential sources of failure or change:

OCR Precision and Image Quality (High Risk):
Risk: Depending on the uploaded image's resolution and font clarity, Tesseract OCR accuracy is extremely erratic. Inaccurate hashes resulting from poor OCR extraction will mark legitimate certificates as "Fake/Modified."
Design Mitigation: Before hashing, extracted data will be cleaned and formatted using an AI-based pre-processing module. Furthermore, the system will use validation logic to promptly reject unclear images and request a better upload from the user.

Latency of the Blockchain Network (Medium Risk):

Risk: Congestion on Ethereum Testnets may result in delayed confirmation times. The web interface may "hang" or timeout while awaiting a blockchain response if the design is synchronous.

Design Mitigation: Asynchronous status updates will be used in the system design. While the blockchain confirmation status is polled in the background until the Transaction ID (TxID) is verified, users will receive an instant acknowledgment of the upload.

Error Immutability (High Risk):

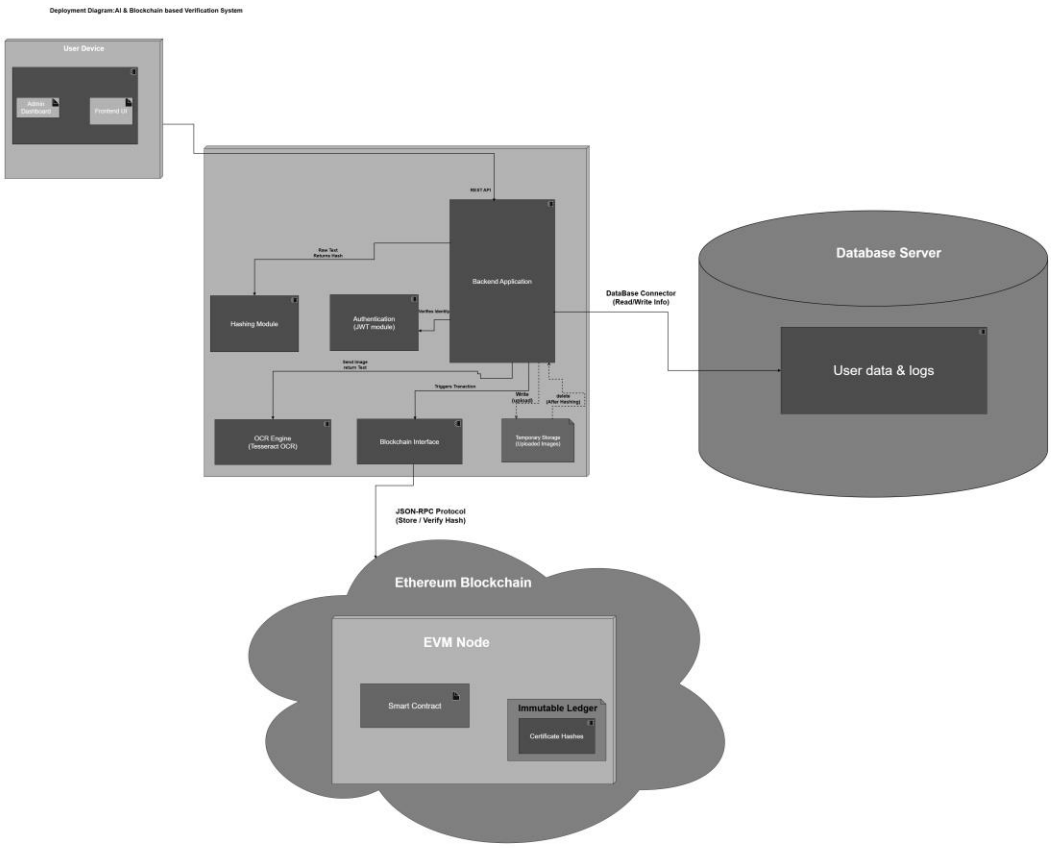
Risk: A certificate hash cannot be changed or removed once it has been recorded on the blockchain. Inaccurate data could be permanently recorded by an institution due to human error during the initial upload.

Design Mitigation: Even though the old record is still on the chain, the design incorporates a logic flow for Re-upload/Correction that creates a new, unique hash and record for the corrected certificate, thereby replacing the previous one in the application layer view.

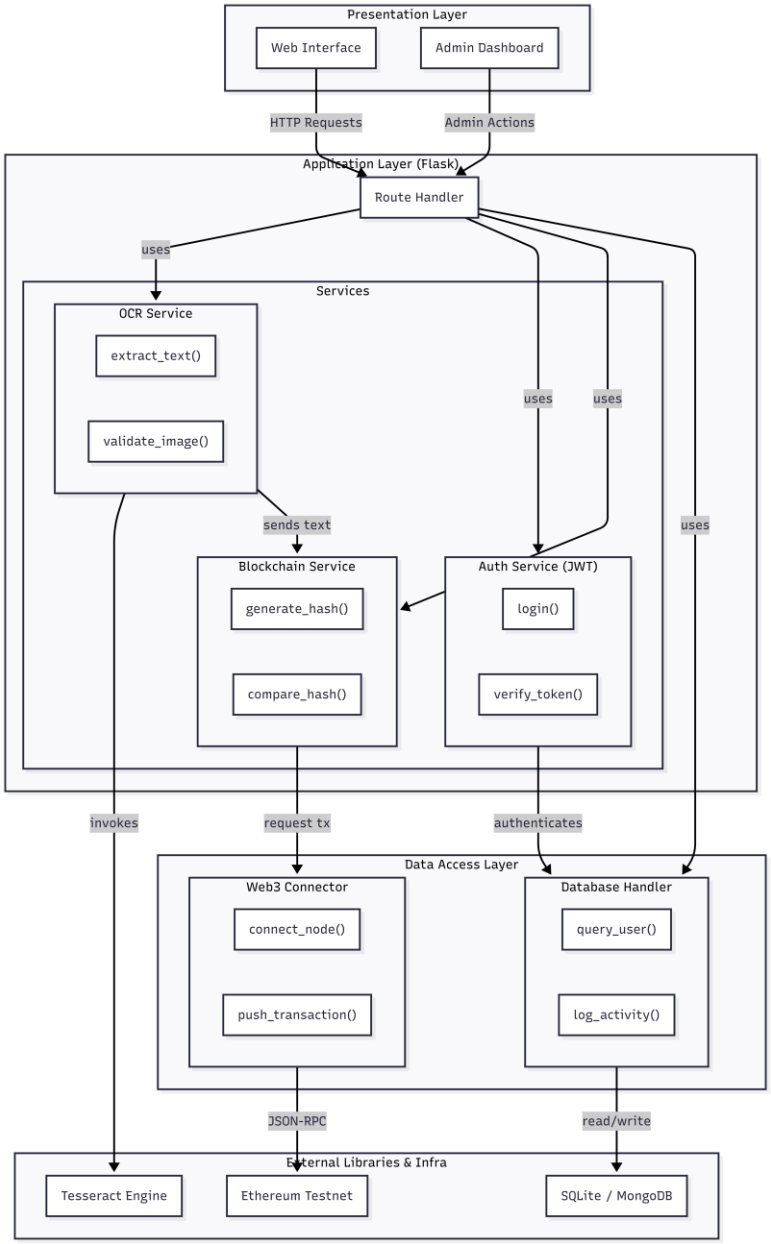
Testnet Stability Dependency (Medium Risk):
Risk: Periodically, testnets like Goerli or Sepolia are reset or deprecated. System availability may be impacted by a strict reliance on a particular network configuration.
Design Mitigation: A configurable blockchain integration layer is maintained in the system design. Updates or migration to different test networks are possible without requiring architectural changes because network identifiers and connection endpoints are abstracted from the main application logic.

3. System Architecture

3.1 System Level Architecture

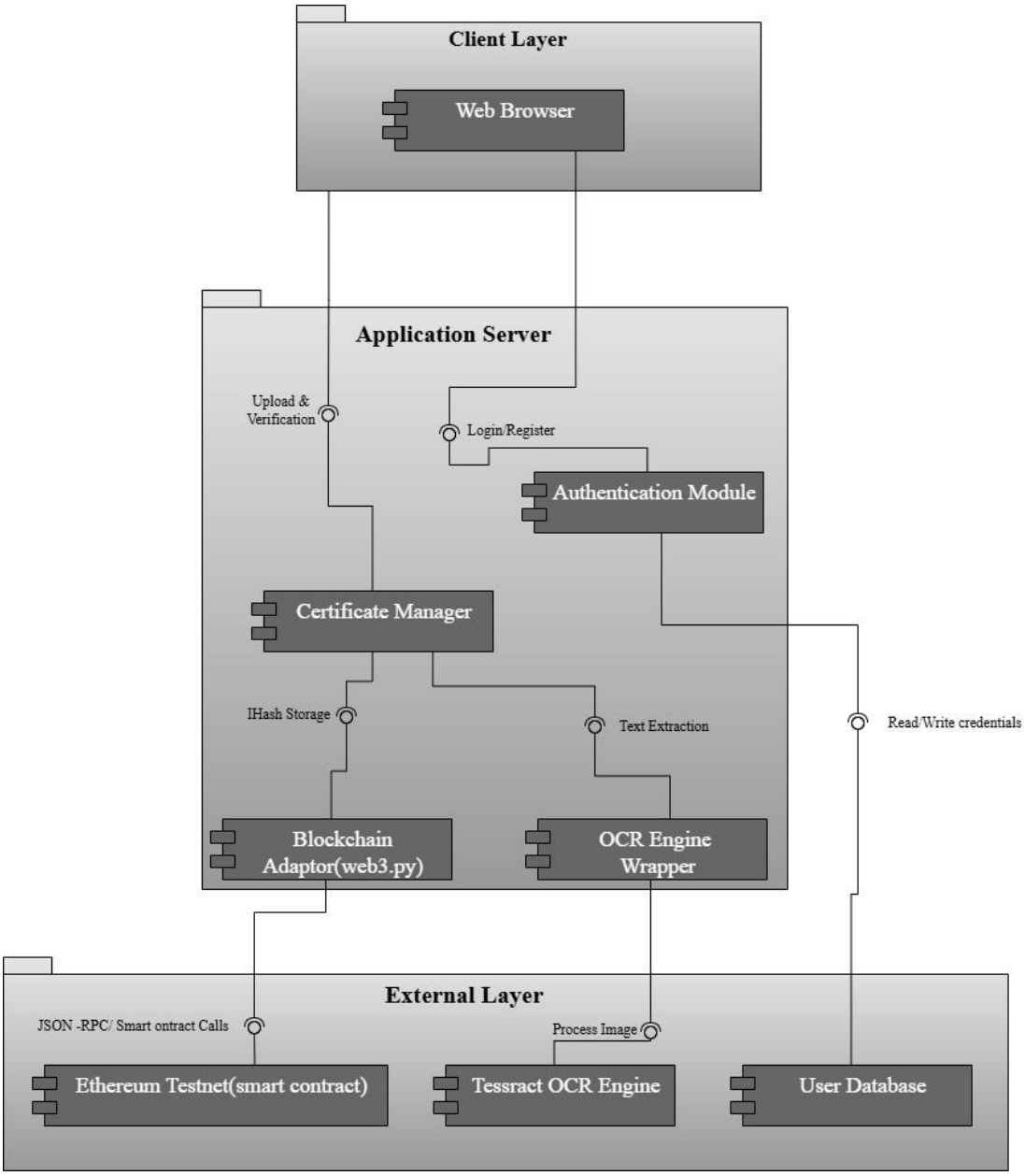


Deployement Diagram



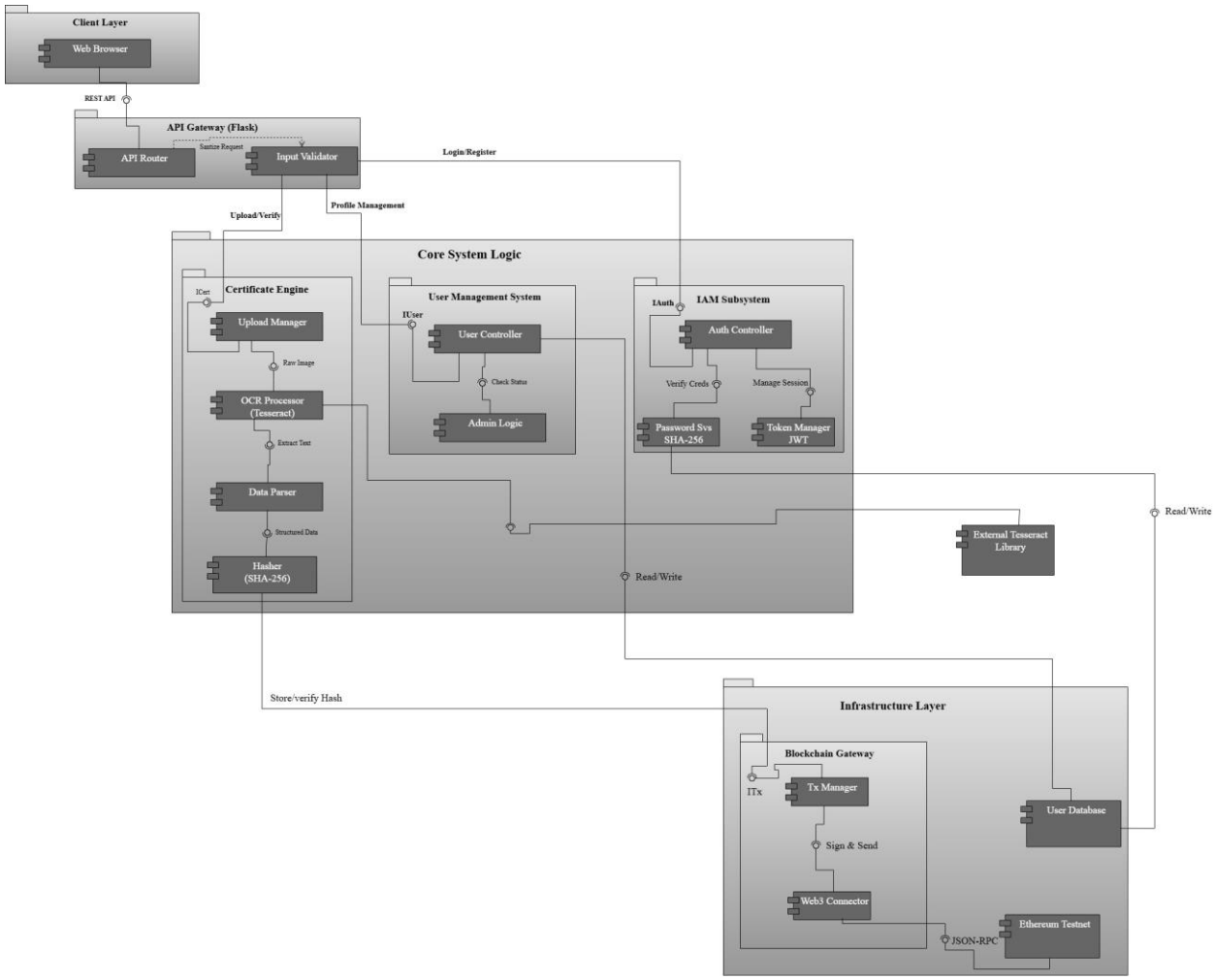
Package Diagram

3.2 Sub-System / Component / Module Level Architecture



Component Diagram (System Level)

3.3 Sub-Component / Sub-Module Level Architecture (1…n)



4. Design Strategies

The main architectural decisions that influenced the AI and Blockchain-Based Certificate Verification System are explained in this section. The system uses a hybrid decentralized application architecture, using blockchain technology only for immutable certificate verification records and traditional web technologies for user interaction, authentication, and data processing.

This method combines blockchain's immutability and trustworthiness with Web-based systems' usability and performance. The following tactics describe how the system respects the performance and privacy limitations specified in the SRS while achieving its objectives of security, automation, transparency, and scalability.

4.1 Strategy 1…n

Strategy 1: Hybrid Data Management (Dual-Storage Architecture)

The system divides data storage into two separate layers: a blockchain-based storage layer using the Ethereum test network and a conventional backend database (SQLite or MongoDB).

System logs, temporary certificate files needed for OCR processing, and user credentials (individuals, organizations, and employers) are all stored off-chain.

The SHA-256 hash of certificate data, which is a permanent digital fingerprint and is stored via smart contracts, is the only data that can be stored on-chain.

Reasoning:

The SRS's privacy and security requirements, which state that no sensitive or personal data should be kept on the blockchain, are directly supported by this tactic.

Because blockchain data is publicly available and unchangeable, storing complete certificates would be against privacy laws and prevent error correction. Sensitive data is kept private and manageable off-chain, and data integrity is guaranteed by storing only cryptographic hashes on-chain.

Because the blockchain verification layer can be independently reused in other document-verification systems, this design also enhances system reuse.

Trade-offs:

Increased complexity: To guarantee consistency, synchronization between the local database and blockchain records needs to be properly managed.

Persistence management: The backend database needs frequent backups and upkeep to avoid data loss, even though blockchain data is permanent.

Strategy 2: Automated Data Capture Powered by AI (OCR Processing Pipeline)

The system automatically extracts text from uploaded certificate images using an AI-based OCR pipeline driven by Tesseract OCR. Before creating a SHA-256 hash, extracted data like student name, ID, course title, and issue date is cleaned and standardized.

Reasoning:

This approach helps achieve the project's objectives of minimizing human error and doing away with manual data entry. Inconsistencies that could result in hash mismatches during verification are more likely when input is done by hand. Automating data extraction increases overall verification reliability and guarantees consistent hashing.

By enabling institutions to upload certificate images directly instead of manually entering data, this method streamlines the process from a usability standpoint.

Trade-offs:

Limitations on accuracy: Inadequately scanned documents may result in inaccurate extraction because OCR accuracy is dependent on image quality and formatting.

Computational overhead: Compared to handling plain text, image processing and text extraction demand more server resources.

Strategy 3: Smart Contracts for Trustless Verification

The system assigns Solidity-based smart contracts installed on the Ethereum test network (Goerli or Sepolia) the responsibility of registering and verifying certificates. These smart contracts serve as an independent, impenetrable registry for hashes of certificates.

Reasoning:

This approach satisfies the need for a transparent and impenetrable verification system. The system does away with the need for a central authority by storing verification logic and records on the blockchain. Long-term confidence in verification results is ensured by the fact that once a certificate hash is recorded, it cannot be changed or deleted, not even by system administrators [1].

Because blockchain transaction records can be used to independently validate verification results, this design also promotes system transparency.

Trade-offs:

Transaction latency: Verification outcomes may be delayed due to blockchain confirmation times.

External dependency: The system is reliant on third-party node providers and the Ethereum test network's availability and stability.

Strategy 4: Future Extension through Component-Based Modularity

The Python Flask framework is used to implement the system using a component-based structure that adheres to the Model-View-Controller (MVC) architectural pattern.

View: HTML, CSS, and JavaScript were used to create this web-based frontend.

Controller: The Flask backend in charge of business logic, API endpoints, and request processing.

Model: The blockchain integration and backend database make up the dual-storage data layer.

Reasoning:

Future system expansion and maintainability are supported by this modular design. Without impacting backend logic, frontend components can be changed or localized (for example, by adding Urdu language support). In a similar vein, future mobile or third-party apps can utilize the backend APIs without having to completely redesign the system.

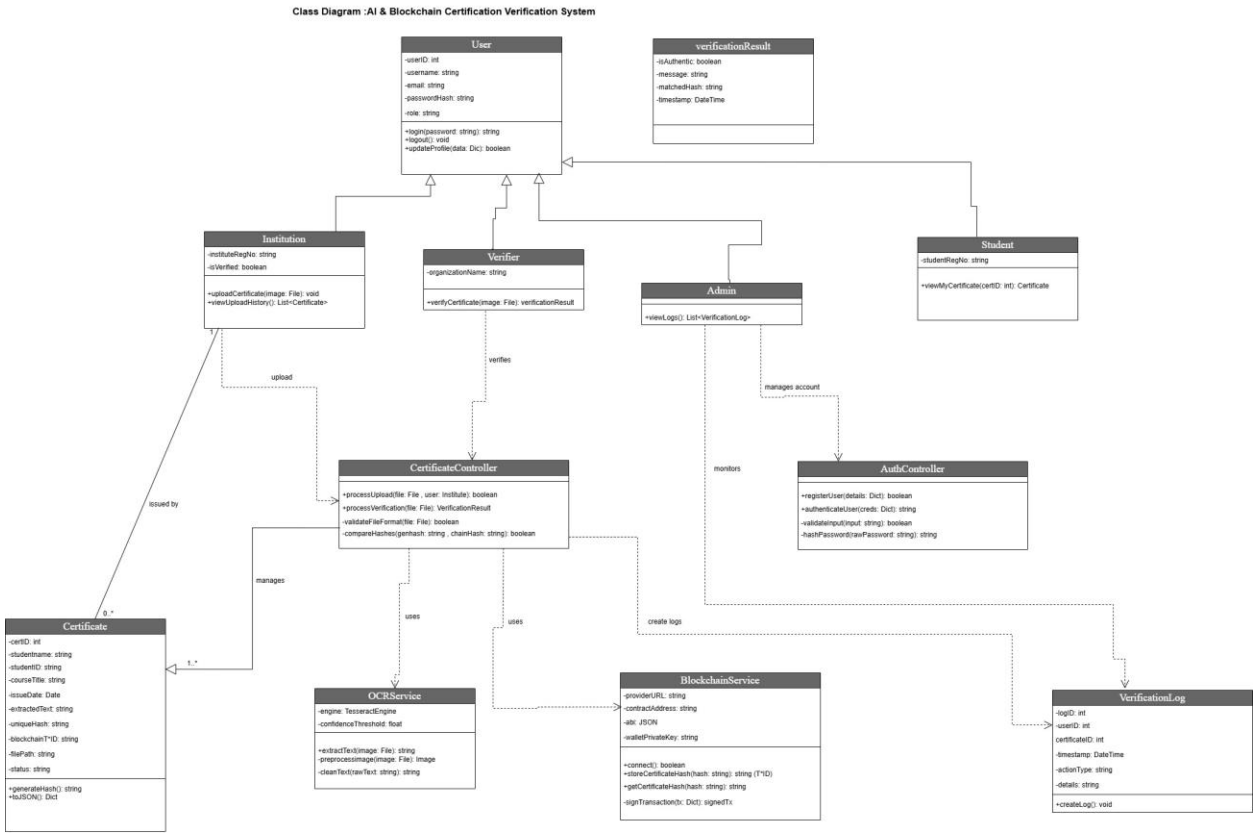
Additionally, this method enhances code organization and enables independent testing and maintenance of individual components.

Trade-offs:

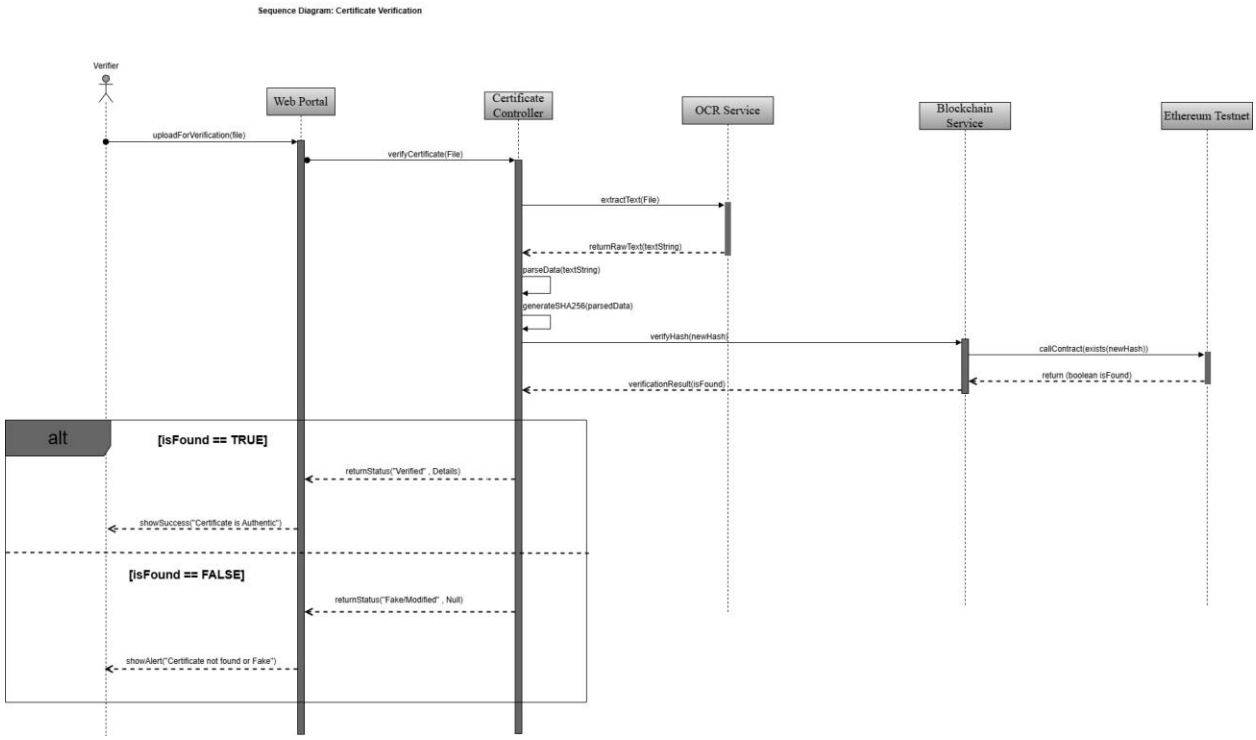
Initial setup effort: More upfront development work is needed to create a clear API structure and enforce separation of concerns.

Learning curve: Instead of managing a single, monolithic application, developers must oversee several system layers.

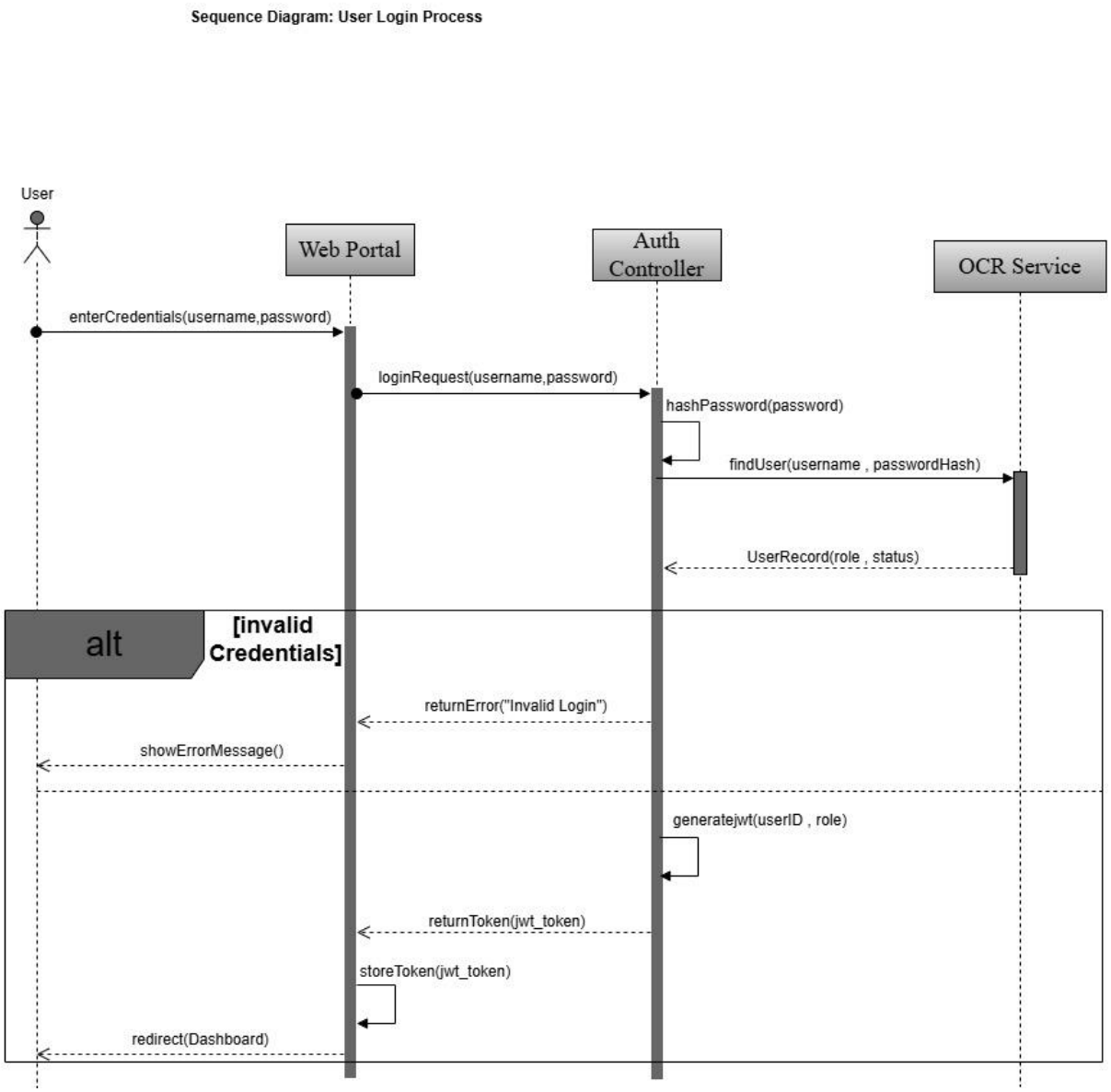
5. Detailed System Design



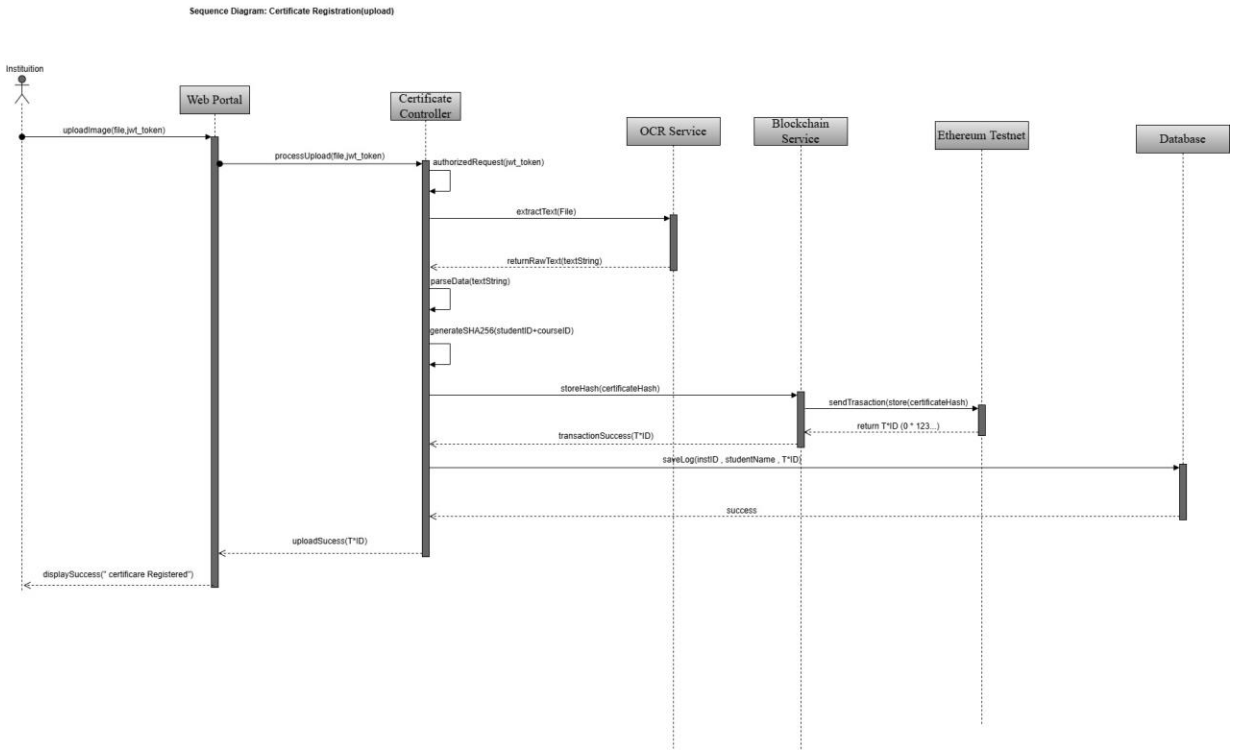
Class Diagram



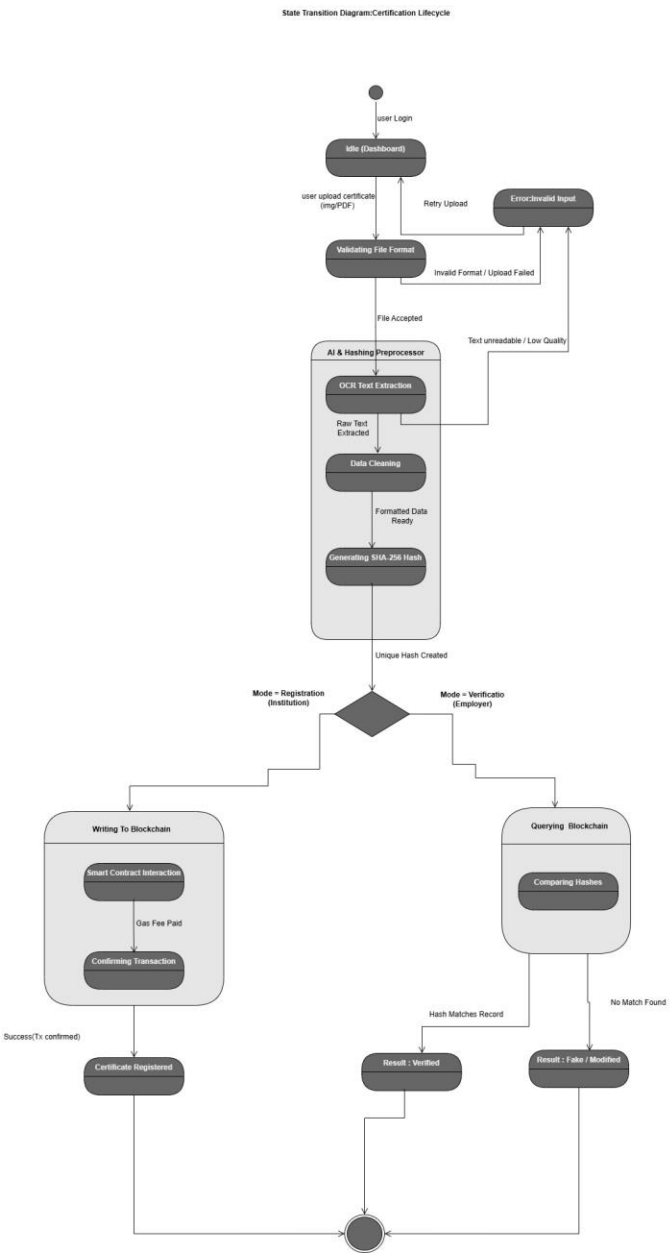
Sequence Diagram (Verification)



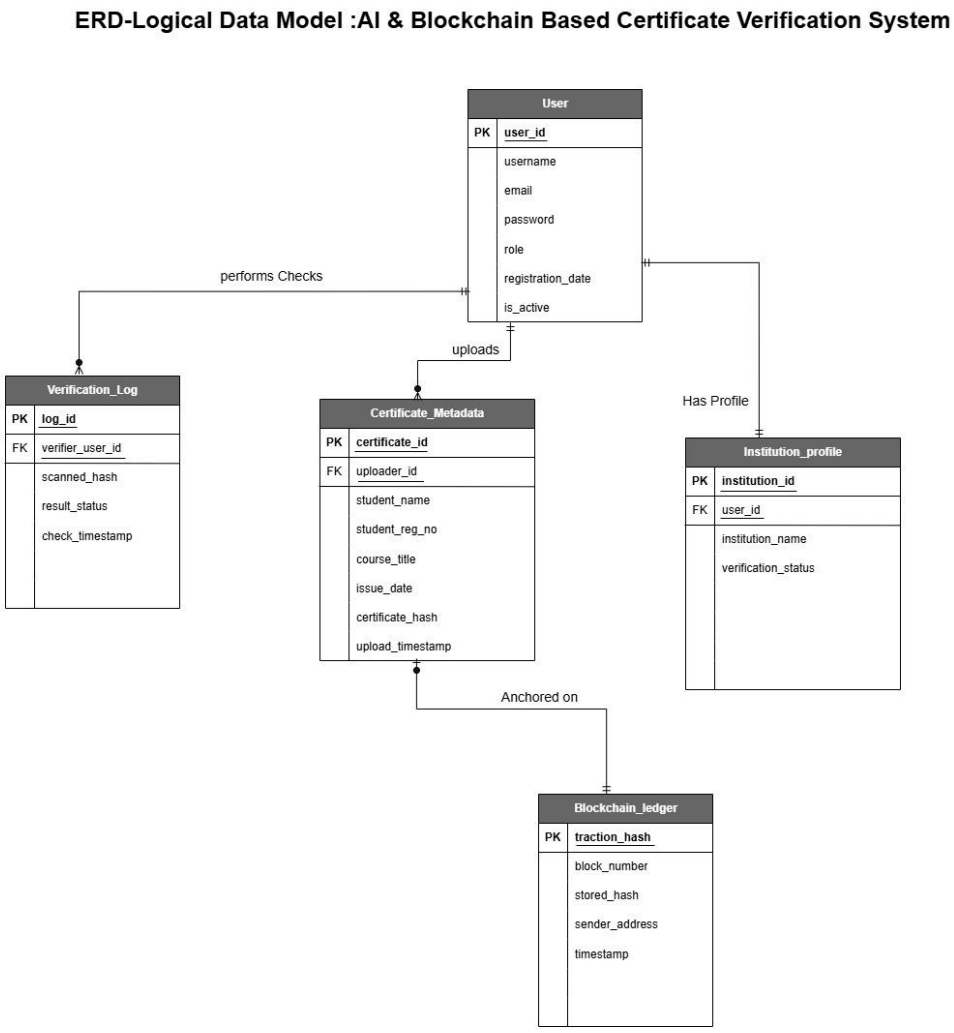
Sequence Diagram (User Login)



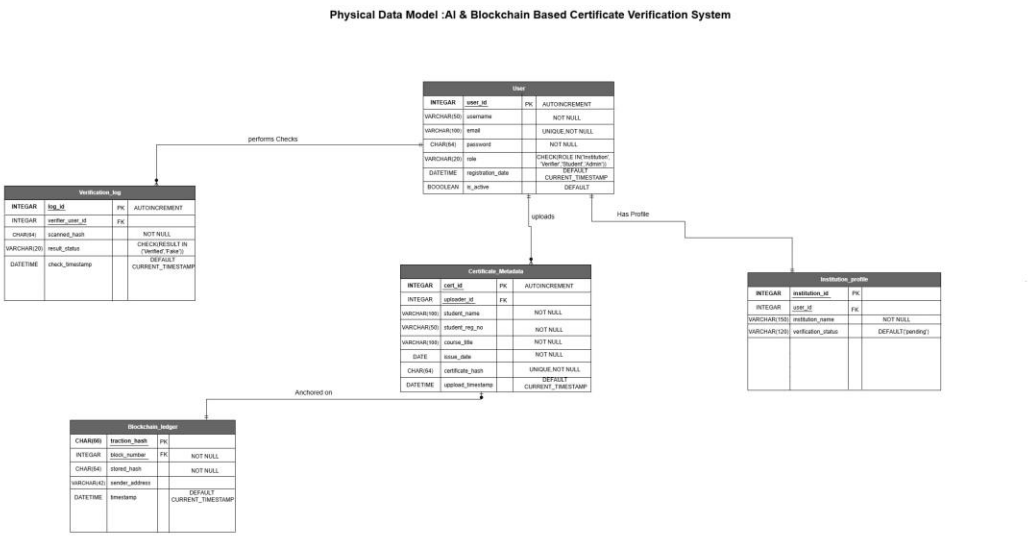
Sequence Diagram (Registration)



State Transition Diagram

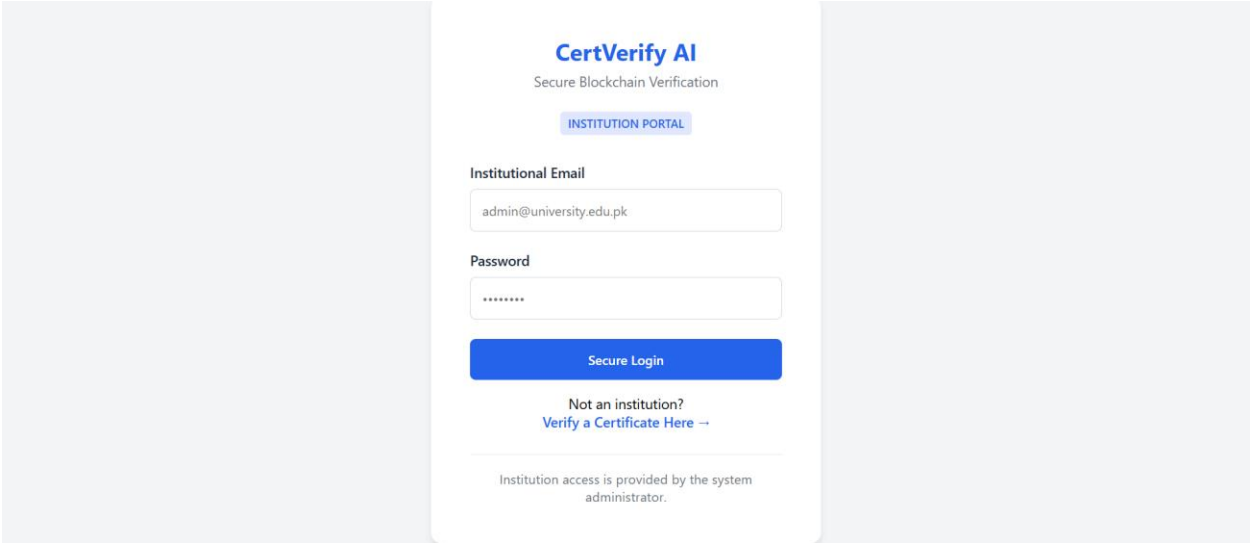


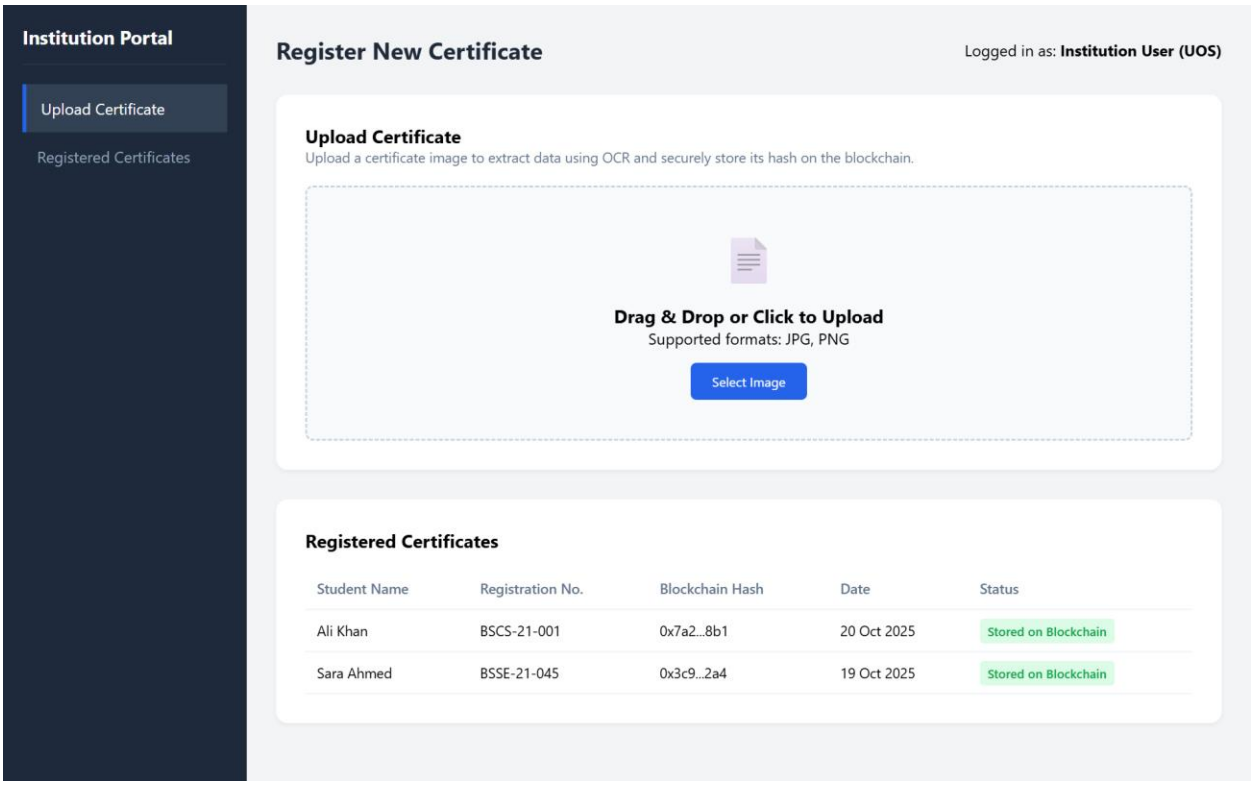
Logical Data Model Diagram

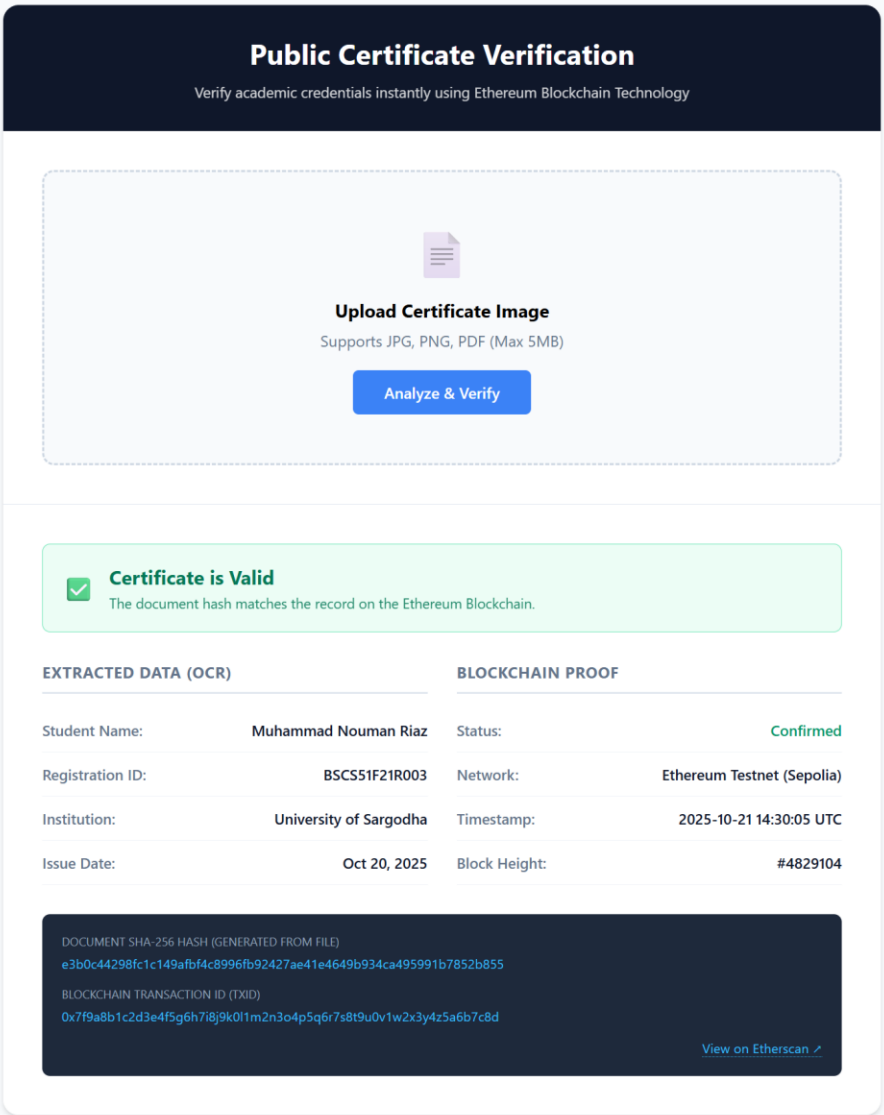


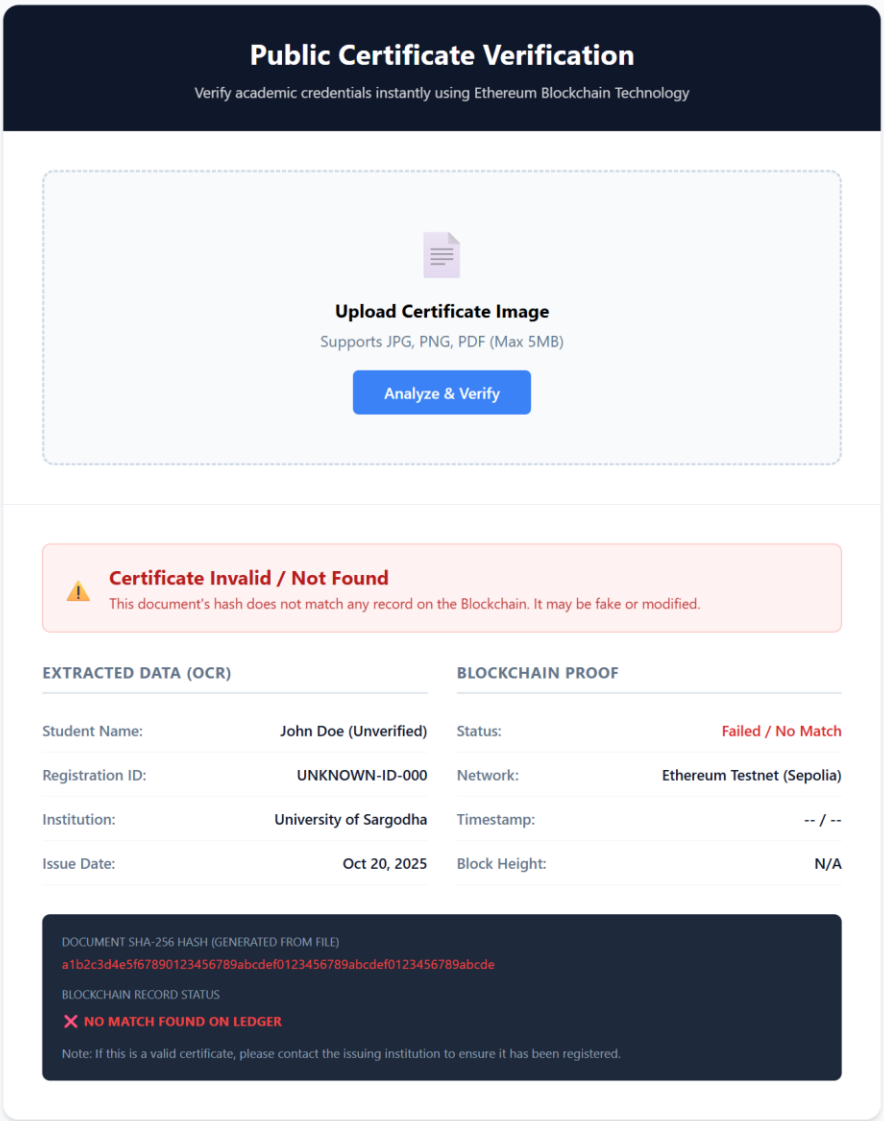
Physical Data Model

GUIs




Institution Portal
Upload Certificate
Registered Certificates
Register New Certificate
Logged in as: Institution User (UOS)
Upload Certificate
Upload a certificate image to extract data using OCR and securely store its hash on the blockchain.
Drag & Drop or Click to Upload
Supported formats: JPG, PNG
Select Image
Registered Certificates
Student Name
Registration No.
Blockchain Hash
Date
Status
Ali Khan
BSCS-21-001
0x7a2...8b1
20 Oct 2025
Stored on Blockchain
Sara Ahmed
BSSE-21-045
0x3c9...2a4
19 Oct 2025
Stored on Blockchain


Public Certificate Verification
Verify academic credentials instantly using Ethereum Blockchain Technology
Upload Certificate Image
Supports JPG, PNG, PDF (Max 5MB)
Analyze & Verify
Certificate is Valid
The document hash matches the record on the Ethereum Blockchain.
EXTRACTED DATA (OCR)
Student Name: Muhammad Nouman Riaz
Registration ID: BSCS51F21R003
Institution: University of Sargodha
Issue Date: Oct 20, 2025
BLOCKCHAIN PROOF
Status: Confirmed
Network: Ethereum Testnet (Sepolia)
Timestamp: 2025-10-21 14:30:05 UTC
Block Height: #4829104
DOCUMENT SHA-256 HASH (GENERATED FROM FILE)
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
BLOCKCHAIN TRANSACTION ID (TXID)
0x7f9a8b1c2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u0v1w2x3y4z5a6b7c8d
View on Etherscan ↗



6. References

S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*	Blockchain principles, immutability, trustless verification	2008	https://bitcoin.org/bitcoin.pdf
Ethereum Foundation, Ethereum Documentation – Smart Contracts	Solidity smart contracts, blockchain implementation	2025	https://ethereum.org/en/developers/docs/smart-contracts/
G. Wood, *Ethereum: A Secure Decentralised Generalised Transaction Ledger (Yellow Paper)*	Ethereum architecture, decentralized ledger	2014	https://ethereum.github.io/yellowpaper/paper.pdf
NIST, *Secure Hash Standard (FIPS 180-4)*	SHA-256 cryptographic hashing for certificates	2015	https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
I. Sommerville, *Software Engineering*, 10th Edition	Object-Oriented Design (OOD)	2016	
Tesseract OCR Project, Tesseract OCR Documentation	AI component for OCR text extraction	2025	https://tesseract-ocr.github.io/tessdoc/
Pallets Projects, *Flask Web Framework Documentation*	Backend development using Flask	2025	https://flask.palletsprojects.com/en/stable/