FORGERY RESISTANT DECENTRALIZED DIGITAL STUDENT ID

By

Chen Jin Shen

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfilment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)

COMMUNICATIONS AND NETWORKING

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

COPYRIGHT STATEMENT

© 2026 Chen Jin Shen. All rights reserved.

This Final Year Project proposal is submitted in partial fulfilment of the requirements for the degree of Bachelor of Information Technology (Honours) Communications and Networking at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors,
Ts Dr Gan Ming Lee and Dr Nadeem Muhammad Waqas who has given me this bright opportunity to engage in a Blockchain design project. It is my first step to establish a career in Blockchain design field. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support and continuous encouragement throughout the course.

ABSTRACT

Student identity management in universities is critical for accessing resources but traditionally relies on easily forgeable physical cards and centralized databases. These legacy frameworks face escalating vulnerabilities regarding security and student data privacy. This project addresses these critical deficiencies by establishing the Forgery Resistant Decentralized Digital Student Identity (FRDDSID) system as an architectural prototype, fully realized in the finalized UNICESS mobile application. Synergizing blockchain technology with Android development, the system utilizes a custom Solidity smart contract on the Ethereum Sepolia Testnet to create a decentralized backend. UNICESS integrates Web3j and the MetaMask SDK to ensure user-controlled transaction execution without exposing private keys. The platform's core innovation is a multi-layered cryptographic architecture. Off-chain personal data is secured using AES-GCM encryption prior to on-chain storage, maintaining confidentiality and immutable transparency. For authentication, hardware-backed biometrics are paired with on-chain data to generate dynamic QR codes. Upon scanning, validators verify the signature and permanently burn the QR hash on-chain, effectively preventing screenshot replay attacks. A localized Retrieval-Augmented Generation (RAG) AI chatbot is also integrated for instant institutional assistance. Ultimately, UNICESS demonstrates a highly secure, transparent, and user-centric model. The dynamic verification mechanism neutralizes forgery vectors, confirming the viability of Web3 architectures in replacing traditional identification frameworks.

Areas of Study: Blockchain-Based Identity Authentication and Verification Systems, Mobile Application Security via Biometric Verification Mechanisms

Keywords: Decentralized Identity, Blockchain Security, Smart Contracts, Cryptographic QR Codes, Replay Attack Prevention, Hardware-Backed Biometrics, AES-GCM Encryption, Role-Based Access Control (RBAC), Identity Forgery, Web3 Architecture

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABI	Application Binary Interface
AES	Advanced Encryption Standard
AI	Artificial Intelligence
AML	Anti-Money Laundering
API	Application Programming Interface
DAO	Decentralized Autonomous Organisation
dApp	Decentralized Application
DID	Decentralized Identity
EBSI	European Blockchain Services Infrastructure
ECDSA	Elliptic Curve Digital Signature Infrastructure
ECIES	Elliptic Curve Integrated Encryption Scheme
FRDDSID	Forgery Resistant Decentralized Digital Student ID
GCM	Galois Counter Mode
IDE	Integrated Development Environment
IPFS	InterPlanetary File System
KMS	Key Management Service
KYC	Know Your Customer
LAN	Local Area Network
LLM	Large Language Model
NFT	Non-Fungible Token
OPA	Open Policy Agent
PII	Personal Identifiable Information
QR	Quick Response
RAG	Retrieval-Augmented Generation
RBAC	Role-Based Access Control
RPC	Remote Procedure Call
SaaS	Software-as-a-Service
SDK	Software Development Kit
SIWE	Sign-In with Ethereum
SSI	Self-Sovereign Identity

TTL	Time-to-Live
UI	User Interface
UTAR	Universiti Tunku Abdul Rahman
W3C	World Wide Web Consortium
ZKP	Zero-Knowledge Proof

CHAPTER 1

Introduction

Identity serves as a fundamental pillar in establishing trust, particularly within structured environments like universities [1]. Student identification is crucial for verifying access to essential resources such as libraries, examination halls and for a wide range of administrative processes. The traditional model of physical identification cards, supplemented by centralized digital records, has long been the standard [2]. However, this model possesses inherent limitations, including susceptibility to damage, loss and critically, forgery. The manual verification processes associated with these cards are often inefficient and prone to human error, leading to operational bottlenecks.

In response to these challenges, many educational institutions have transitioned towards centralized digital identity solutions, typically in the form of mobile applications. Duke University, The University of Alabama and The University of Oklahoma were among those early adopters who explored integrating student IDs into mobile applications [3], offering increased convenience and portability compared to physical cards. While these systems offer greater convenience and portability, they often replicate the underlying centralized architecture, introducing new vulnerabilities related to data security, user privacy and single points of failure.

This project, titled "Forgery Resistant Decentralized Digital Student Identity (FRDDSID)," presents a paradigm shift from these conventional models. It proposes and develops a robust digital identity system for students at Universiti Tunku Abdul Rahman (UTAR) by leveraging the unique properties of blockchain technology [4]. The core of this project is the development of a fully functional prototype, encompassing an application for students, a corresponding verifier application, dynamic Quick Response (QR) codes logic, and the UNICESS, backed by a secure Solidity smart contract deployed on the Ethereum blockchain [15]. This FRDDSID system not only modernizes student identification but also significantly improves its resistance to forgery and enhances the overall integrity and efficiency of identity verification processes within a university environment [6].

1.1 Problem Statement and Motivation

In the modern university environment, the shift from physical to centralized digital student identification was adopted by various universities. Unfortunately, this transition has fundamentally failed to address the most critical challenge of security, even while showing improvement on administrative overhead [7]. According to El Haddouti and El Kettani, the core problem lies in the inherent vulnerabilities of these centralized digital identity systems [8].

These systems typically rely on a single central database to store and manage sensitive student identity information. A breach of this database can compromise the personal data of thousands of students, leading to severe privacy violations and identity theft. The centralized nature of these systems also raises concerns about potential unauthorized surveillance or misuse of information without explicit student consent or transparency [9][10]. Furthermore, a tangible security flaw in many basic digital IDs is their reliance on static barcodes or QR codes for verification. These are highly susceptible to simple forgery methods, such as taking a screenshot or photograph of a legitimate ID. A truly secure digital identity system requires a verification method that resists such elementary attacks. Therefore, the problem is not merely the lack of a digital ID, but the significant security deficiencies of the prevailing centralized model used in universities [11]. These weaknesses undermine the trust placed in the identification system by students, administrators and external parties who rely on a verifiable student status.

The primary motivation for this project is to address these critical security and efficiency gaps. The increasing sophistication of forgery techniques necessitates a more robust and modern solution. A key driving force is the desire to leverage the transformative potential of blockchain technology to create a fundamentally more secure and privacy-respecting system, moving beyond simply digitising the student ID [12]. This is motivated by blockchain's inherent characteristics of immutability, transparency and resistance to single points of failure.

Furthermore, the project is motivated by the specific goal of implementing a sophisticated forgery resistant mechanism. Recognising that a static digital ID can be easily copied, the integration of a dynamic QR code, linked to a secure blockchain transaction, is crucial to significantly enhance the security of the verification process

and ensure that only the legitimate, current identity can be validated [13]. Ultimately, the core motivation is to explore and demonstrate the application of decentralized identity principles in the higher education context [14], showcasing a solution that provides enhanced security and improved efficiency for verification, thereby setting a potential standard for future digital identity solutions in the sector.

1.2 Objectives

The primary objectives of this project are strategically designed to address the multifaceted challenges of modern university identification. The first objective is to digitalise and modernise the student identity experience by delivering a user-friendly Android mobile application, named UNICESS. This application empowers students to securely manage their digital identity, leveraging device-level biometrics for secure key storage and ensuring the displayed information is a comprehensive and real-time representation of their official status.

Secondly, the project aims to implement and validate a robust, dynamic QR code mechanism to combat forgery. This is achieved by developing a time-sensitive, cryptographically signed QR code alongside the UNICESS featuring Role-Based Access Control (RBAC) allowing authorized staff to scan and authenticate the credential, thereby demonstrating a superior resistance to forgery compared to static codes.

The third and final objective is to demonstrate a decentralized and transparent identity system founded on blockchain technology. This is accomplished through the design and deployment of a Solidity smart contract on the Ethereum Sepolia Testnet, with full integration into the UNICESS via the Web3j library, showcasing how blockchain's immutability and transparency create a more secure and trustworthy verification process.

1.3 Project Scope and Direction

The scope of this project encompasses the development and delivery of the finalized UNICESS application, which is built upon the foundational proof-of-concept prototype of the FRDDSID system established in previous phases. This project demonstrates the viability and enhanced security of a decentralized approach over traditional, centralized identity systems within a university context.

The primary deliverable is the UNICESS application, a unified, native Android platform featuring OpenZeppelin Role-Based Access Control (RBAC). This single application allows students to manage their encrypted identity and generate dynamic QR codes, while simultaneously enabling authorized university staff to scan and validate these credentials, manage student identity statuses, and monitor an in-app blockchain transaction audit log in real-time. The backbone of the system is a Blockchain Smart Contract, developed in Solidity and deployed on the Ethereum Sepolia Testnet, which handles all core identity processes. Finally, the scope encompasses the full Integration Layer, utilising the Web3j library and the Etherscan V2 Application Programming Interface (API) to connect the UNICESS with the blockchain network, leveraging the MetaMask Android SDK for secure transaction signing, and integrating a local Retrieval-Augmented Generation (RAG) Artificial Intelligence (AI) backend for institutional assistance.

The project is therefore directed towards establishing a comprehensive proof-of-concept and a foundational architecture, rather than a production-ready system for immediate, large-scale deployment.

1.4 Impact, Significance and Contributions

The primary significance of this project lies in its potential to enhance the security and trustworthiness of the student identification system at UTAR. The project reduces the risks of identity fraud, which has implications for campus security, examination integrity and access to university resources. The impact extends to student empowerment by adopting a model based on Self-Sovereign Identity (SSI). Moreover, the project provides students with direct control over their personal data, aligning the university with modern data privacy standards and enhancing student trust in the institution's digital infrastructure.

Furthermore, the successful implementation of this prototype serves as a significant case study for other higher education institutions in the region. It demonstrates a possible path away from legacy systems and towards a more secure, efficient and modern digital campus environment.

Key Contributions:

- A Functional End-to-End Prototype: The project delivers a working prototype encapsulated within the UNICESS, utilizing RBAC for administrators, staff, and students. This system provides a practical demonstration of a complete digital identity lifecycle. Therefore, achieving the goal of modernising student identification.
- A Forgery-Resistant Mechanism: A key innovation is the implementation of a dynamic QR code verification protocol. By combining a fresh, on-chain timestamp with a biometrically secured digital signature using Elliptic Curve Digital Signature Algorithm (ECDSA), the system generates time-sensitive credentials with a strict 300-second validation window. Furthermore, it incorporates an on-chain cryptographic burning mechanism to ensure QR codes are strictly single-use, rendering screenshot and replay forgery methods completely ineffective.
- A Demonstration of Decentralized Principles: By deploying a Solidity smart contract on the public Ethereum Sepolia Testnet, the project provides a demonstration of decentralized identity in action. It illustrates how core blockchain principles like immutability and transparency can be leveraged alongside Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) off-chain encryption and a native transaction audit explorer to create a highly secure, privacy-preserving, and trustworthy system for managing identity state.
- A Blueprint for Native dApp Development: The project's successful integration of modern development tools serves as a comprehensive blueprint for building a native mobile Decentralized Application (dApp).

1.5 Report Organisation

This report is structured into seven chapters to provide a comprehensive overview of the development of the UNICESS application based on the underlying FRDDSID system, from conception to implementation and evaluation.

- Chapter 1 introduces the project, outlining the problem statement, motivation, research objectives, project scope, and key contributions.
- Chapter 2 presents a detailed literature review of existing digital identity solutions, analyzing their strengths and weaknesses to establish the context and architectural foundation for this project.
- Chapter 3 details the system methodology and approach. It presents the system architecture, use case diagrams outlining RBAC, and activity diagrams mapping the dynamic verification and routing flows.
- Chapter 4 delves into the system design. It outlines the system block diagram, component specifications, the cryptographic architecture (AES-256-GCM and secp256r1 ECDSA), the RAG AI pipeline, and the precise interaction operations between the mobile frontend and decentralized backend.
- Chapter 5 documents the system implementation. It covers the hardware and software environments, crucial network configurations, and provides a detailed walkthrough of the system operations (with visual interfaces) alongside practical implementation challenges.
- Chapter 6 focuses on system evaluation and discussion. It presents quantitative performance metrics (read and write latencies), evaluates the system's security against specific threat vectors, discusses architectural limitations, and systematically verifies the completion of the project objectives.
- Chapter 7 concludes the report by summarizing the project's achievements and proposing tangible recommendations for future enhancements, such as transitioning to asymmetric on-chain encryption and decentralized file storage.

CHAPTER 2

Literature Reviews

2.1 Previous Works on Decentralized Digital Identity

2.1.1 Midy

Midy is a digital identity product from Norton, a globally recognized leader in consumer cybersecurity. It functions as an identity provider, aiming to simplify and secure online identity verification for the mainstream Web2 internet. Users create a digital wallet managed by Midy to store and share verified data with trusted entities. The core value proposition is built on Norton's brand trust and security infrastructure. Midy focuses on user privacy by minimising data sharing, but this occurs within a centralized trust model where Norton is the ultimate custodian of the identity system.

Figure 2.1 Homepage of Midy

Strengths

- Trusted Brand & Security
- Simple User Experience
- Controlled Data Sharing

Midy leverages Norton's strong reputation in cybersecurity, which builds immediate trust and can help ease adoption concerns among non-technical users. This focus on the mass market is reflected in its design, which prioritises a simple and intuitive user experience for identity verification and login. While operating on a centralized model, it maintains privacy by allowing users to control precisely what verified information is shared during transactions, preserving a degree of privacy

Weaknesses

- Centralized Control
- No User Sovereignty
- High Risk of Breaches

Midy's primary weakness is its centralized architecture. Norton acts as the central authority [16], creating a single point of failure and control that is contrary to the principles of SSI. This means users do not truly own their identities, which creates risks of censorship, de-platforming and vendor lock-in. Consequently, a centralized database of identity information presents a high-value target for cyberattacks, increasing the risk of large-scale data breaches.

2.1.2 Galxe

Galxe (formerly Project Galaxy) is a leading Web3 credentialing and community engagement platform [17]. It enables projects to build and manage communities by rewarding users with on-chain credentials for their participation. The Galxe Identity Protocol, which allows users to build a self-sovereign digital identity based on their achievements across the web. It uses Zero-Knowledge Proof (ZKP) technology, allowing users to prove facts about their credentials without revealing sensitive data [18].

Figure 2.2 Poster of Galxe Passport

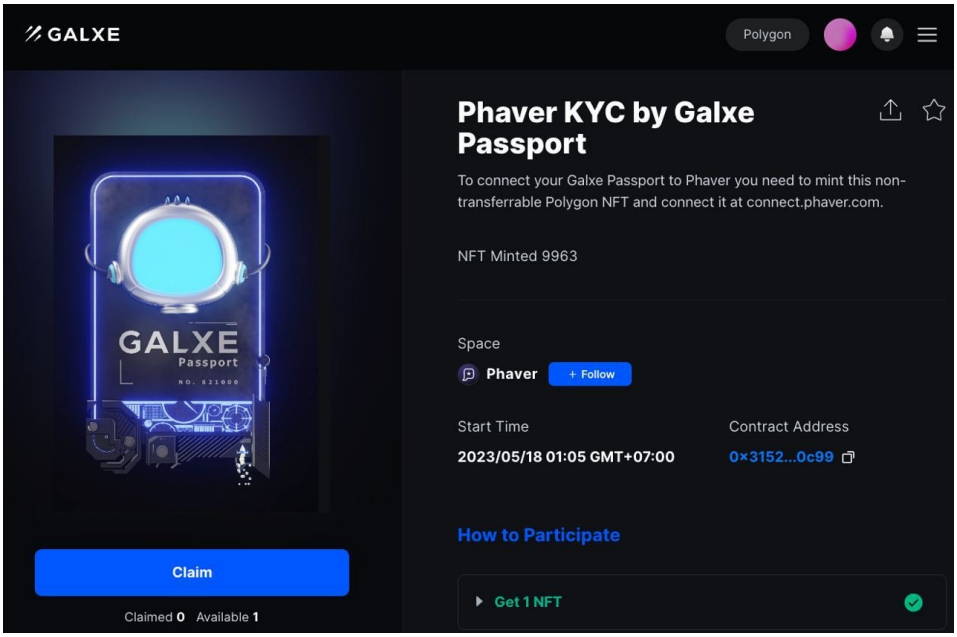

Figure 2.3 Non-Fungible Token (NFT) for Galxe Passport

Strengths

- Large Network Effect
- Engaging User Experience
- Privacy-Preserving Technology

Galxe's primary strength is its massive network effect, with a large user base and a vast network of integrated applications that make its credentials immediately useful. It fosters an engaging user experience by using quests, rewards and gamification to make building a digital identity interactive and fun. The platform is also privacy-preserving, as its use of ZKPs ensures users can maintain control over their personal information while still proving their qualifications.

Weaknesses

- High Complexity for New Users
- Marketing and Community Focus
- Risk of Credential Farming

A key weakness is the platform's complexity for newcomers with the sheer number of features and the Web3-native environment can be overwhelming for users unfamiliar with the technology. Furthermore, its primary use case is community building and marketing, which means it is not well-suited for formal identity verification. This incentive-based model also creates the potential for credential farming. The users perform tasks solely to collect credentials which could devalue it over time.

2.1.3 SpruceID

SpruceID is a company focused on building the open-source infrastructure for decentralized identity. Rather than a single application, they provide developer-focused tools, SpruceKit [19]. This enables developers to integrate user-controlled identity into their applications. SpruceID is a key proponent of open standards like "Sign-In with Ethereum" (SIWE), W3C DIDs and Verifiable Credentials. Their approach is to provide the secure blocks that allow others to create a decentralized web where users control their data.

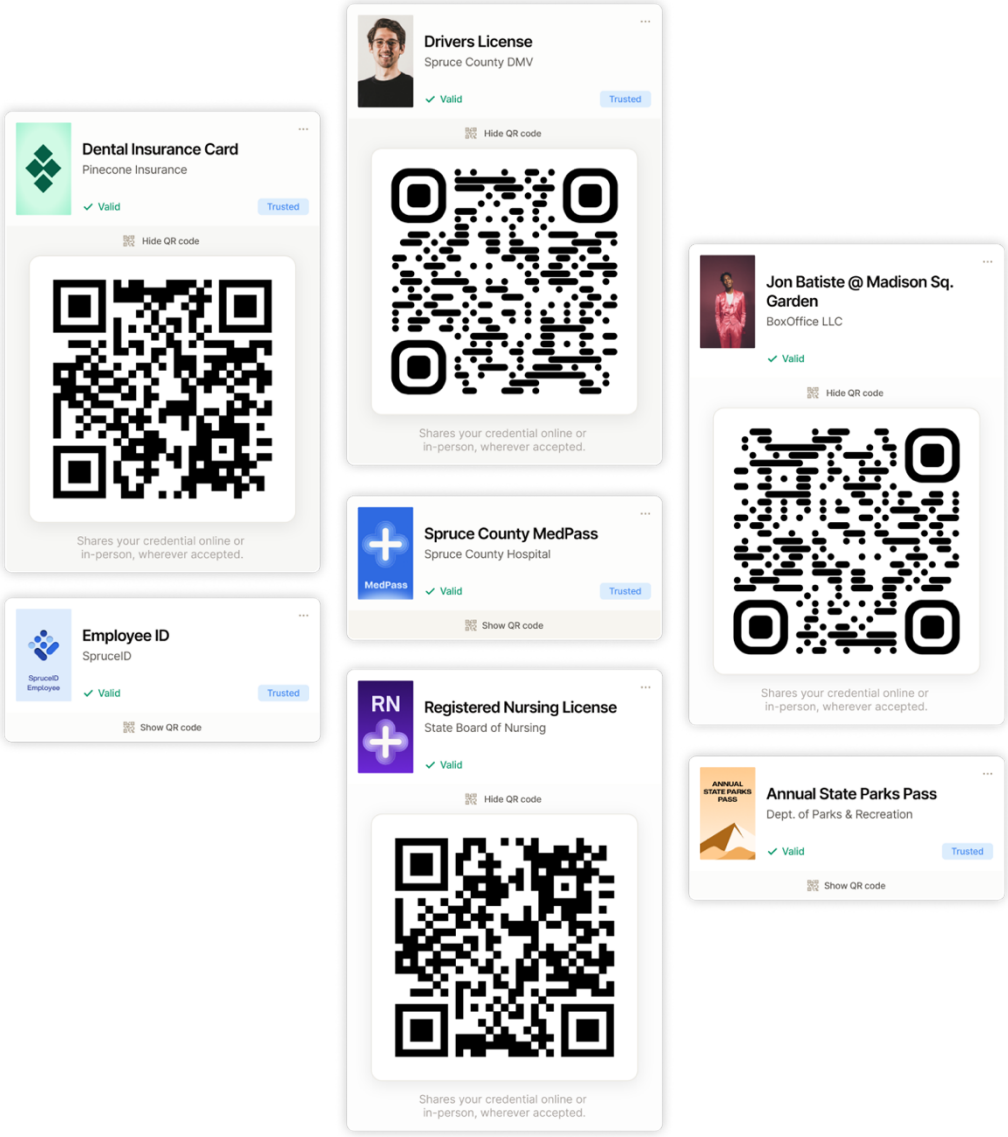

Figure 2.4 Homepage of SpruceID

Strengths

- Open Standards
- Developer-Centric Flexibility
- Enables User Sovereignty

SpruceID's strength lies in its strong commitment to open standards like SIWE which prevent ecosystem fragmentation. It provides powerful and low-level tools that give developers maximum freedom to build secure identity solutions [20]. Ultimately, SpruceID is designed to build systems where users have complete and sovereign control over their digital identity.

Weaknesses

- Requires Heavy Development
- No End-User Application
- Longer Time-to-Market

A primary weakness is that SpruceID requires significant development to implement. As a toolkit, it is not an out-of-the-box solution and demands a high level of technical expertise and resources. Furthermore, SpruceID does not provide a wallet or application for end-users, the implementing organisation is responsible for the entire user experience. This "build-it-yourself" approach leads to a longer time-to-market compared to using a more integrated platform.

Figure 2.5 Verifier of SpruceID

2.1.4 Fractal ID

Fractal ID is a decentralized identity solution designed for Know Your Customer (KYC) and Anti-Money Laundering (AML) [21]. Its core concept is to provide users with a single and reusable digital identity to eliminate the need for repetitive verification across different dApps and platforms. By issuing verifiable credentials, Fractal aims to balance the regulatory needs of projects with user convenience and privacy.

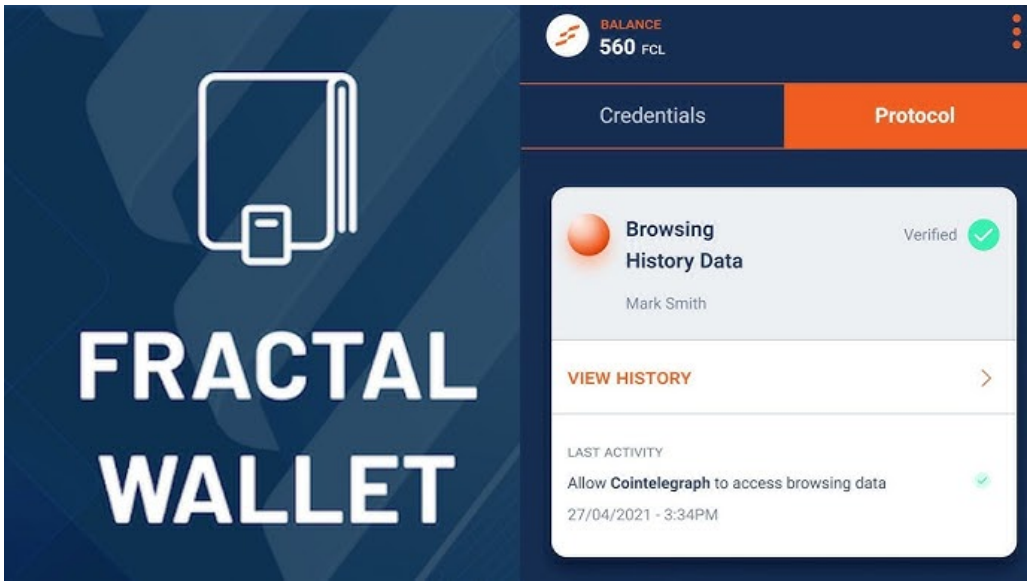

Figure 2.6 Wallet of Fractal ID

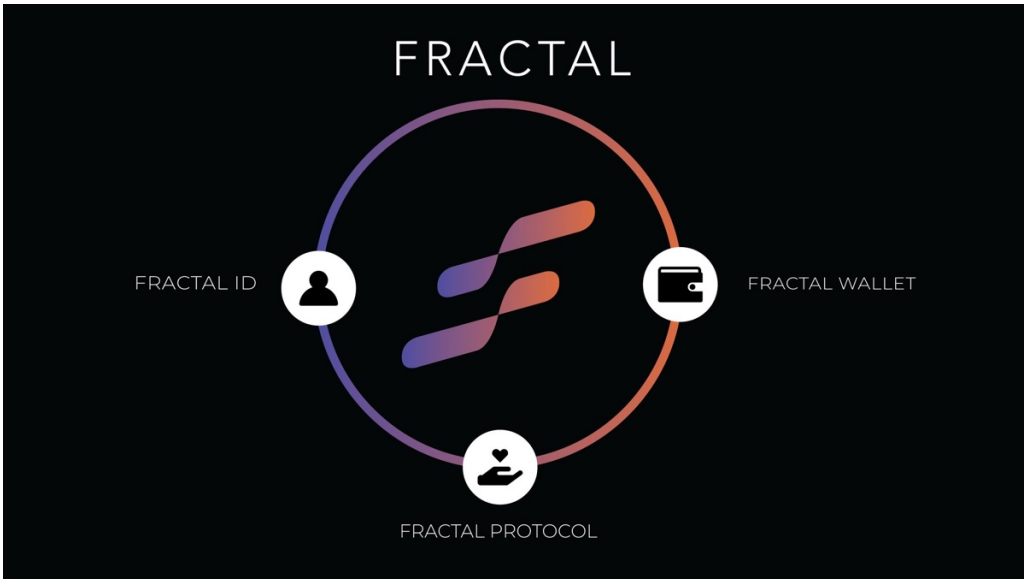

Figure 2.7 Ecosystem of Fractal ID

Strengths

- Solves a Key Compliance Problem
- Provides User Convenience
- Natively Designed for Web3

Fractal ID's main strength is that it directly addresses the critical need for KYC/AML compliance in Web3, a major hurdle for projects seeking to operate in regulated environments. Its reusable digital identity model is highly convenient for users, reducing the friction of onboarding with new services. As a Web3-native solution, it is designed specifically for the blockchain ecosystem, making it easier to integrate with dApps and Decentralized Autonomous Organisation (DAO).

Weaknesses

- Significant Centralisation Risks
- Major Security Vulnerabilities
- Inherent Privacy Trade-offs
- Dependence and Counterparty Risk

Despite its "decentralized" branding, Fractal ID's data storage processes have centralized components, creating centralisation risks and a single point of failure. A past data breach stands as a major security vulnerability that exposed sensitive user information and undermined trust in the platform [22][23]. The nature of KYC requires the collection of sensitive personal data, which creates a fundamental privacy trade-off against the ideals of the Web3 space. Consequently, projects and users become dependent on a single company, creating significant counterparty risk related to its security and business practices.

2.1.5 walt.id

walt.id provides an open-source, developer-focused infrastructure for building digital identity and wallet solutions [24]. It is not an end-user application but a toolkit that allows organisations to create their own customized identity systems. A key feature is its strong adherence to open standards from the W3C and the European Blockchain Services Infrastructure (EBSI) [25]. walt.id offers powerful features like customisable verification policies using the Open Policy Agent (OPA) which allows for fine-grained control over how credentials are verified. This makes it a robust choice for enterprises, governments and organisations that require a compliant and self-hosted identity solution.

Figure 2.8 Logo of walt.id

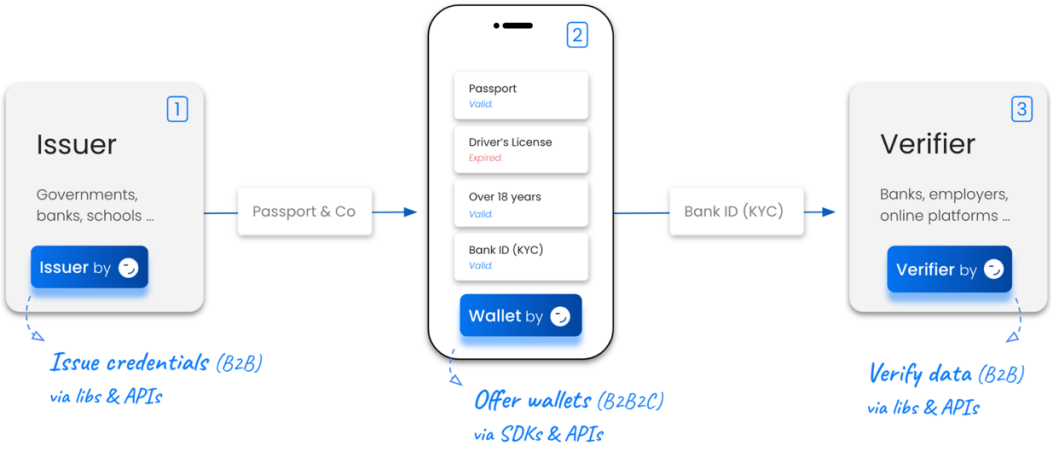

Figure 2.9 Process of walt.id

Strengths

- Open-Source and Transparent
- Enterprise-Ready
- Comprehensive and Interoperable

walt.id's open-source feature is a key strength as it promotes transparency, security audits and community collaboration while preventing vendor lock-in. The platform is enterprise-ready by offering a scalable and customisable solution designed to be compliant with regulations, making it highly suitable for institutional use. Furthermore, it is comprehensive and interoperable, providing a full suite of tools that support major standards, ensuring solutions built with it can interact with the wider digital identity ecosystem.

Weaknesses

- Requires In-House Development
- High Barrier to Entry
- Limited Library Support

A primary weakness is that walt.id requires in-house development. It has no Software-as-a-Service (SaaS) offering, meaning organisations must host, maintain and develop their own applications. This creates a high barrier to entry, as the complexity and resource requirements can be a major hurdle for organisations without a dedicated development team. Additionally, while support is growing, there may be limited libraries for certain programming languages, potentially restricting development options.

2.2 Comparison for Previous Works on Decentralized Digital Identity

The following table provides a concise comparative analysis of five distinct digital identity tools to evaluate their suitability for a university digital student ID project. This framework is designed to quickly highlight the fundamental differences between centralized products, community platforms and foundational developer toolkits.

Table 2.1 Comparison of Existing System

Tool	Use Case	Key Advantage	Key Disadvantage	Complexity
Midy	Single Sign-On (SSO)	Simple, trusted user experience.	Lacks a decentralized blockchain.	Low
Galxe	Web3 Community Engagement	Large, existing user network.	Not suitable for formal identity.	Medium
SpruceID	Foundational Toolkit	Full control, open standards, enables SSI.	Requires in-house development from scratch.	High
Fractal ID	KYC & Compliance Bridge	Official identity verification for Web3.	Centralized risks and a history of a data breach.	Medium
walt.id	Open-Source Infrastructure	Self-hosted, compliant with EU standards.	Requires a dedicated development team.	Very High

2.3 Summary

From the user-friendly, centralized model of Midy to the developer-focus, decentralized toolkits of SpruceID and walt.id reveals a critical trade-off in the current landscape. The platforms that integrate open standards and user control such as SpruceID and walt.id often present a high barrier to entry. It required significant in-house development and failing to provide a ready-made application for end-users.

Furthermore, specialized platforms like Galxe are optimized for community engagement rather than formal identity verification, while compliance-focused solutions like Fractal ID have demonstrated the security risks inherent in any remaining centralized components.

This analysis makes it clear that a successful digital identity system must achieve a difficult synthesis. First and foremost, it must combine the intuitive user experience of a Web2 product with the security and user-sovereignty principles of Web3 [26]. Therefore, the UNICESS project is designed to bridge this gap. By developing a complete application, it directly overcomes the high barrier to entry of developer-only toolkits. Moreover, it offers a tangible product that students and administrators can use out of the box.

Crucially, the project's architecture is fundamentally decentralized, avoiding the single points of failure like Midy and Fractal ID while prioritising usability. By integrating device-level biometrics with on-chain timestamping, UNICESS implements a dynamic and forgery resistant verification mechanism — **a specific security feature not explicitly addressed by the other platforms**.

Ultimately, this project synthesises the most vital lessons from the existing works. It includes a seamless user interface with multi-layered security and a truly decentralized backend, positioning it as a comprehensive and novel solution tailored for the specific needs of a university environment.

CHAPTER 3

System Approach

This project adopts the iterative Spiral methodology to systematically transition the FRDDSID prototype into the finalized UNICESS application. Through continuous planning, risk analysis, and engineering, this approach mitigates critical vulnerabilities early by testing and integrating smart contract logic with the mobile frontend.

3.1 System Design Diagram

3.1.1 System Architecture Diagram

The architectural design of the underlying FRDDSID system is based on a multi-tiered and decentralized model. This architecture clearly separates the client-side UNICESS application from the on-chain logic, utilizing established Web3 services as a bridge. The architecture diagram illustrates the primary components of the system and their fundamental interactions.

The platform consists of three distinct layers. The User's Device contains the UNICESS application, which serves as the primary frontend interface, alongside the MetaMask Wallet, which functions as the secure signer for all transactions. The External Web Services layer is represented by an Infura Remote Procedure Call (RPC) Node acting as a gateway to the blockchain, the Etherscan V2 API for retrieving audit logs, and a local Python-based RAG backend utilizing Flask and Ollama to process AI chat queries.

Finally, the Decentralized Network layer is the Ethereum Sepolia Testnet, which hosts the deployed FRDDSID Smart Contract. All state-changing operations such as registration, profile updates, and QR timestamping are initiated by UNICESS, signed by MetaMask, and submitted through Infura RPC. Read-only operations such as fetching a student's profile or resolving RBAC statuses are sent directly from UNICESS through Infura RPC to the smart contract using the Web3j library. Concurrently, off-chain operations like querying university knowledge are routed to the local AI backend.

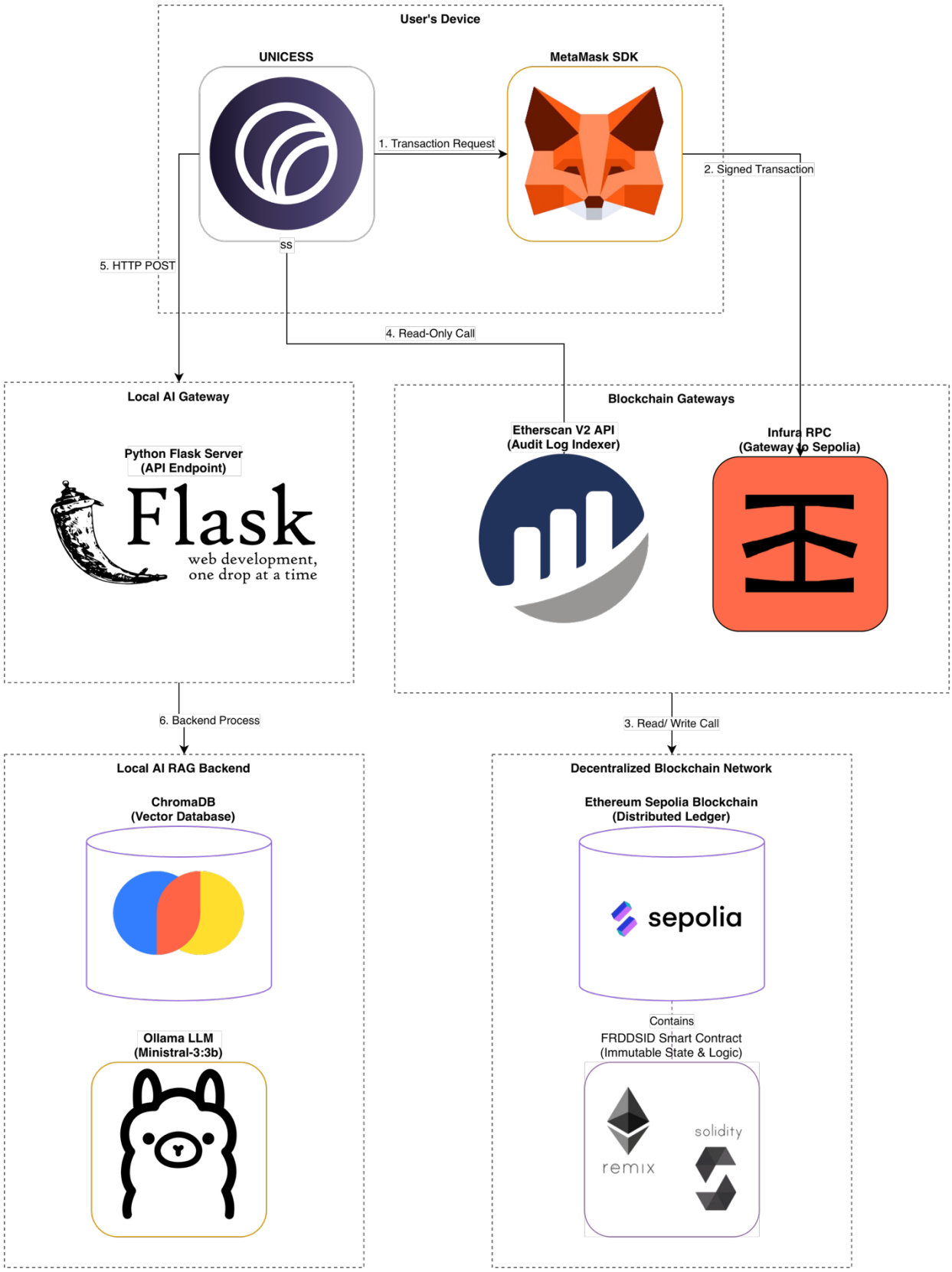

Figure 3.1 System Architecture Diagram of UNICESS

3.1.2 Use Case Diagram and Description

The Use Case Diagram provides a high-level conceptual overview of the system's intended functionality and the interactions between the system and its various users (actors). Due to the integration of RBAC the FRDDSID system utilizes the UNICESS interface that dynamically exposes specific functionalities based on the cryptographic privileges held by the connected MetaMask wallet.

Actors

The system identifies four primary human actors and two system actors:

1. Unregistered User: A user who has connected their MetaMask wallet to the application but has not yet recorded their identity details on the smart contract.
2. Student: A registered user whose identity is stored on the blockchain. Students possess a hardware-backed biometric private key used for generating cryptographic signatures.
3. Staff (Admin): A university administrator granted the STAFF_ROLE on the smart contract. They are responsible for identity lifecycle management and verification.
4. Owner (Admin): The deployer or primary administrator of the smart contract, holding the DEFAULT_ADMIN_ROLE. They manage the organizational hierarchy of the system.
5. Ethereum Blockchain (System Actor): The decentralized Sepolia Testnet network housing the smart contract, processing state changes, and enforcing RBAC rules.
6. RAG AI Backend (System Actor): The local Python-based Flask API that processes user queries against a ChromaDB vector database using the Ministral-3:3b Large Language Model (LLM).

Use Case Descriptions

UC1: Register Identity: An Unregistered User inputs their Personal Identifiable Information (PII). The system encrypts sensitive fields (Gender, Email, Phone, Faculty, Program), generates a biometric ECDSA key pair, and submits the data to the blockchain via MetaMask.

UC2: Connect Wallet and Route Role: The user connects their MetaMask wallet. The system automatically prompts a network switch to the Sepolia Testnet and queries the Ethereum Blockchain to determine the user's RBAC status, routing them to the appropriate dashboard (Registration, Home, Verify, or Owner Panel).

UC3: Manage Identity Data: A Student edits their profile information. The system re-encrypts the updated payload and submits an on-chain transaction. This action automatically reverts the student's status to "Pending", requiring Staff re-approval to maintain data integrity.

UC4: Manage Student Accounts: A Staff member accesses the identity management list to view all registered students. The Staff member can activate pending accounts, deactivate active accounts, or delete a student's identity from the smart contract.

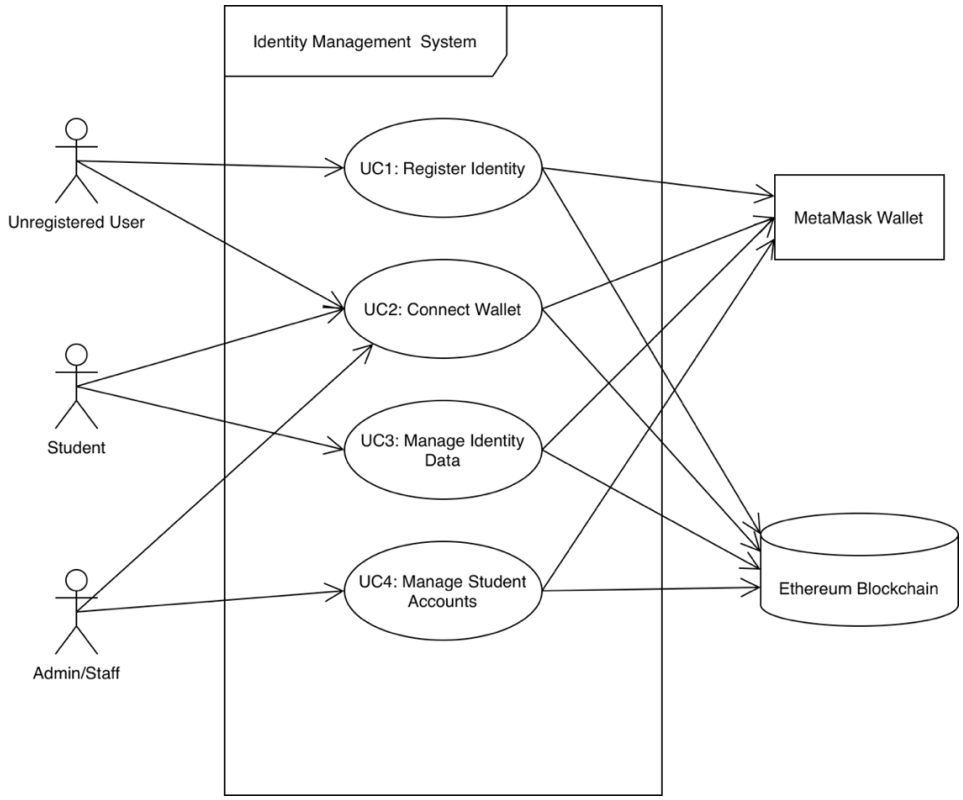

Figure 3.2 Activity Diagram of Identity Management System

UC5: Generate Dynamic QR Code: A Student passes a biometric prompt to access their private key. A transaction is sent to update their on-chain timestamp. Upon successful mining, the app signs the timestamp and generates a time-sensitive QR code containing the wallet address, timestamp, and signature.

UC6: Record Verification (Burn QR): Upon successful validation, the Staff member submits a transaction to burn the unique QR signature hash on the blockchain, permanently rendering the QR code single-use and preventing screenshot replay attacks.

UC7: Scan and Verify Identity: A Staff member scans a student's dynamic QR code. The system parses the payload, validates the 300-second Time-to-Live (TTL) expiration, fetches the student's on-chain public key and verifies the cryptographic signature.

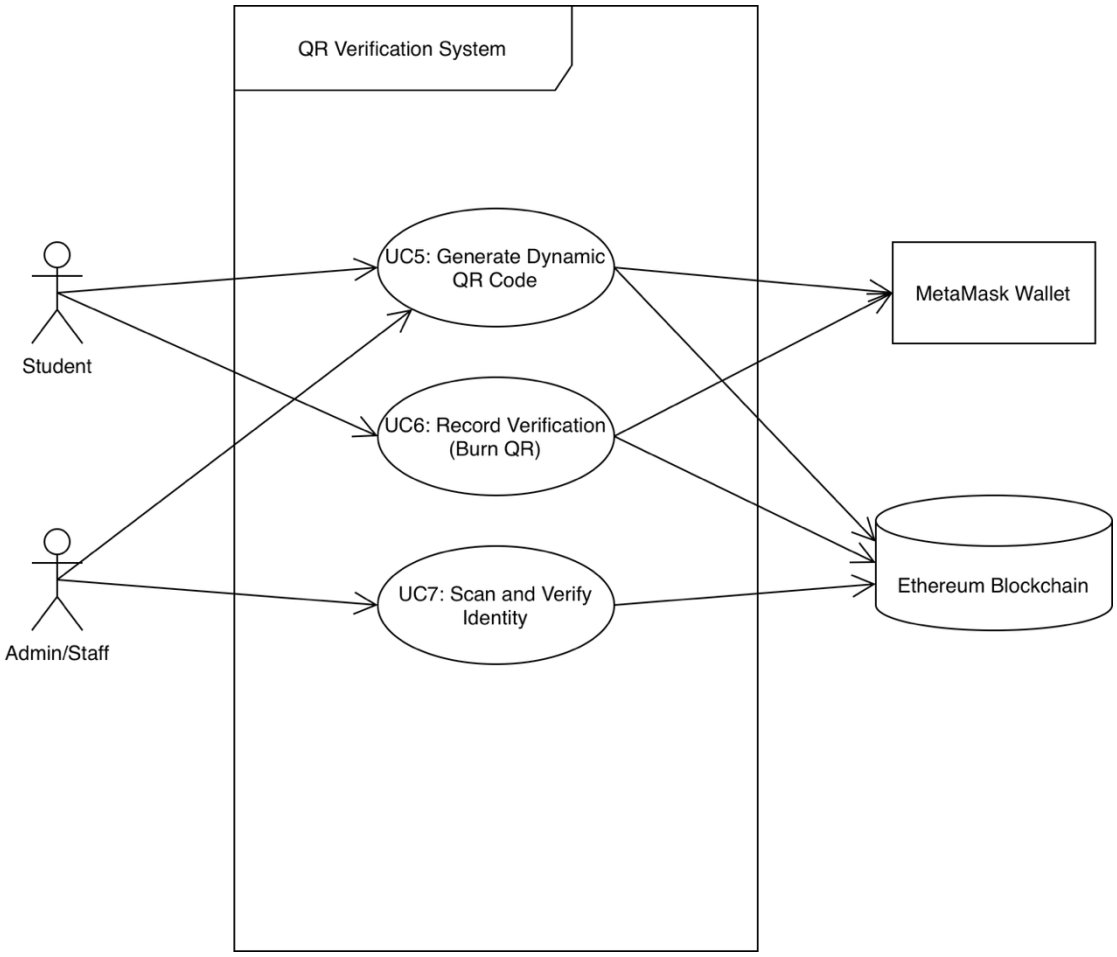

Figure 3.3 Activity Diagram of QR Verification System

UC8: Manage Organizational Roles: The Owner accesses the administrator management panel. They can assign the STAFF_ROLE to new wallet addresses, revoke existing staff roles, or transfer the DEFAULT_ADMIN_ROLE to a new wallet.

UC9: Monitor Blockchain Audit Log: Staff and Owners can view an in-app transaction history. The system queries the Etherscan V2 API, decodes raw hexadecimal method payloads into human-readable actions, and resolves raw wallet addresses into registered student or staff names.

UC10: Query AI Assistant: User can input a query into the chat interface. The app sends the prompt to the RAG AI Backend, which streams a response back to the Android interface based on the university's vectorized knowledge base.

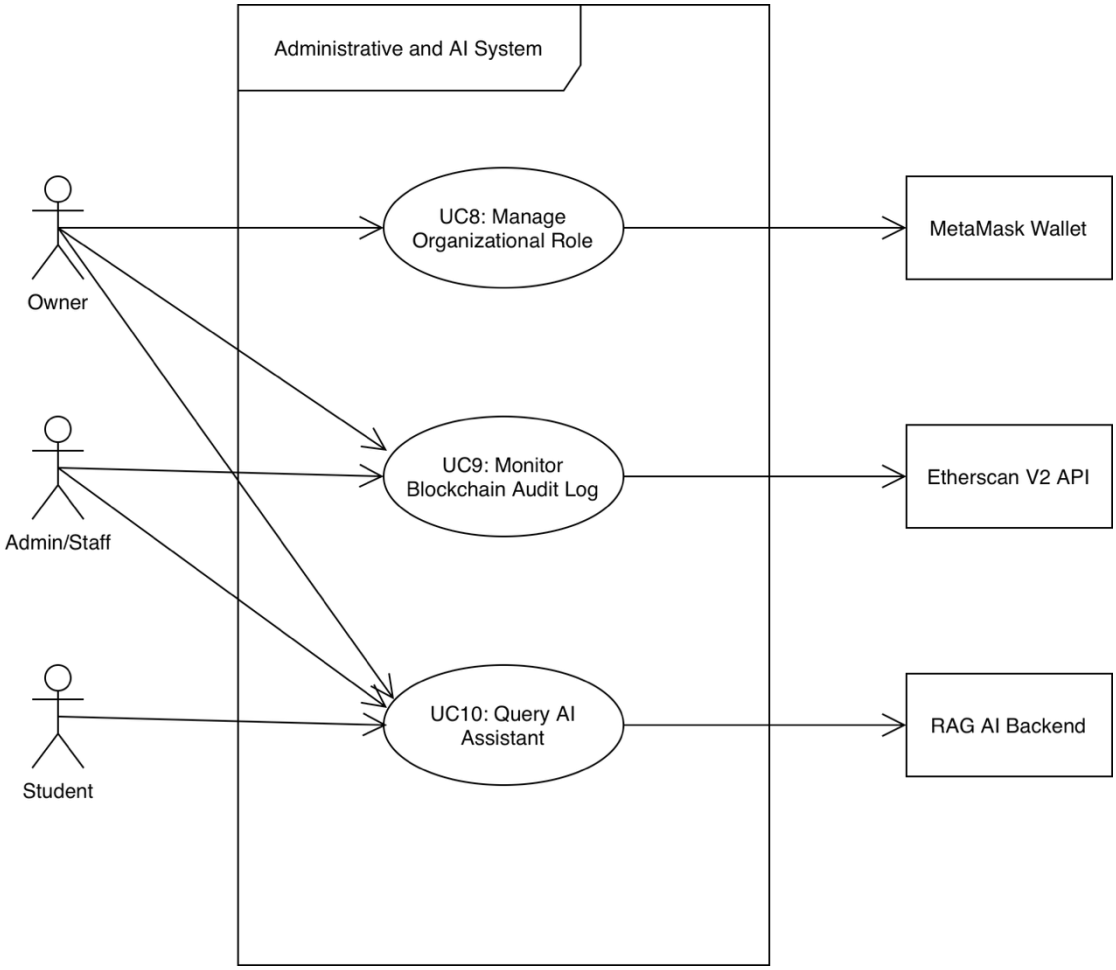

Figure 3.4 Activity Diagram of Administrative and AI System

3.2 Activity Diagram

3.2.1 User Authentication Flowchart

The initial user experience is critical for usability and security. The system employs an automated authentication and role-based routing mechanism to ensure that users are seamlessly directed to the appropriate interface based on their on-chain status.

Upon launching the app, a connection to the user's MetaMask wallet is initiated, automatically prompting a network switch to the Sepolia Testnet. Once the wallet address is retrieved, the system performs a sequence of on-chain queries to determine the user's RBAC status. It first checks if the wallet holds the DEFAULT_ADMIN_ROLE (Instituition). If not, it checks for the STAFF_ROLE (Admin). If neither applies, it verifies if the wallet is associated with a registered student profile.

Based on the result, the user is either routed to the main Student Home Screen if they are registered or to the Registration Form if they are a new user. This flow ensures a clear and secure separation of roles from the very start of the user session.

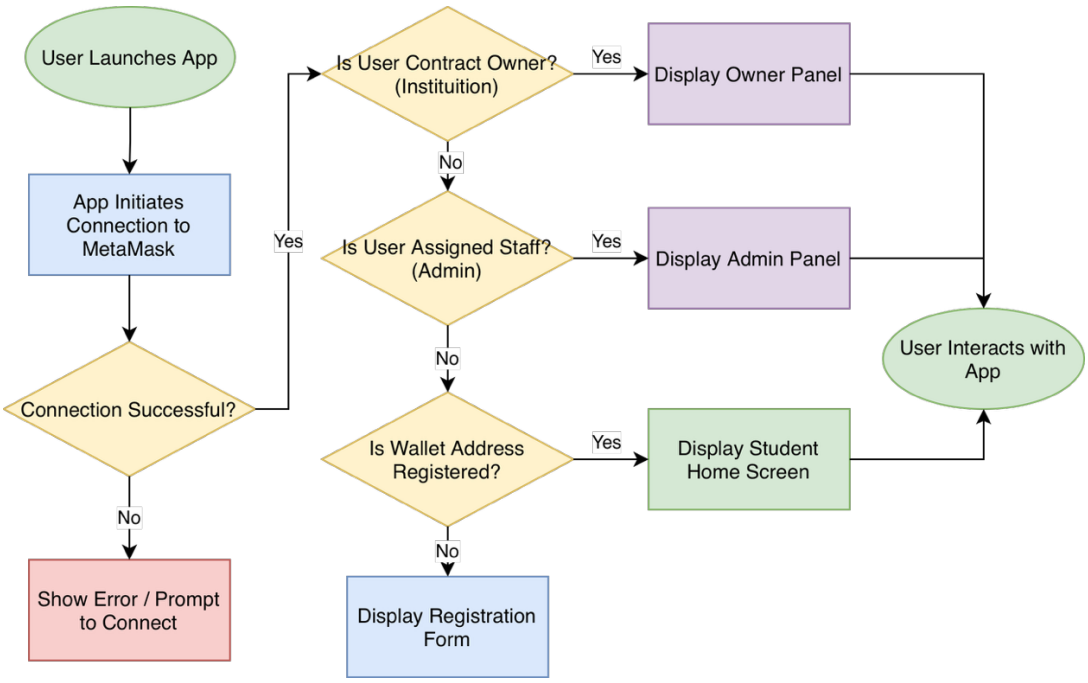

Figure 3.5 Flowchart of User Authentication

3.2.2 Dynamic QR Code Generation and Verification Flowchart

The core innovation of the UNICESS is the forgery resistant verification mechanism which relies on the generation and validation of a dynamic, cryptographically signed QR code. The following diagram illustrates this entire end-to-end process.

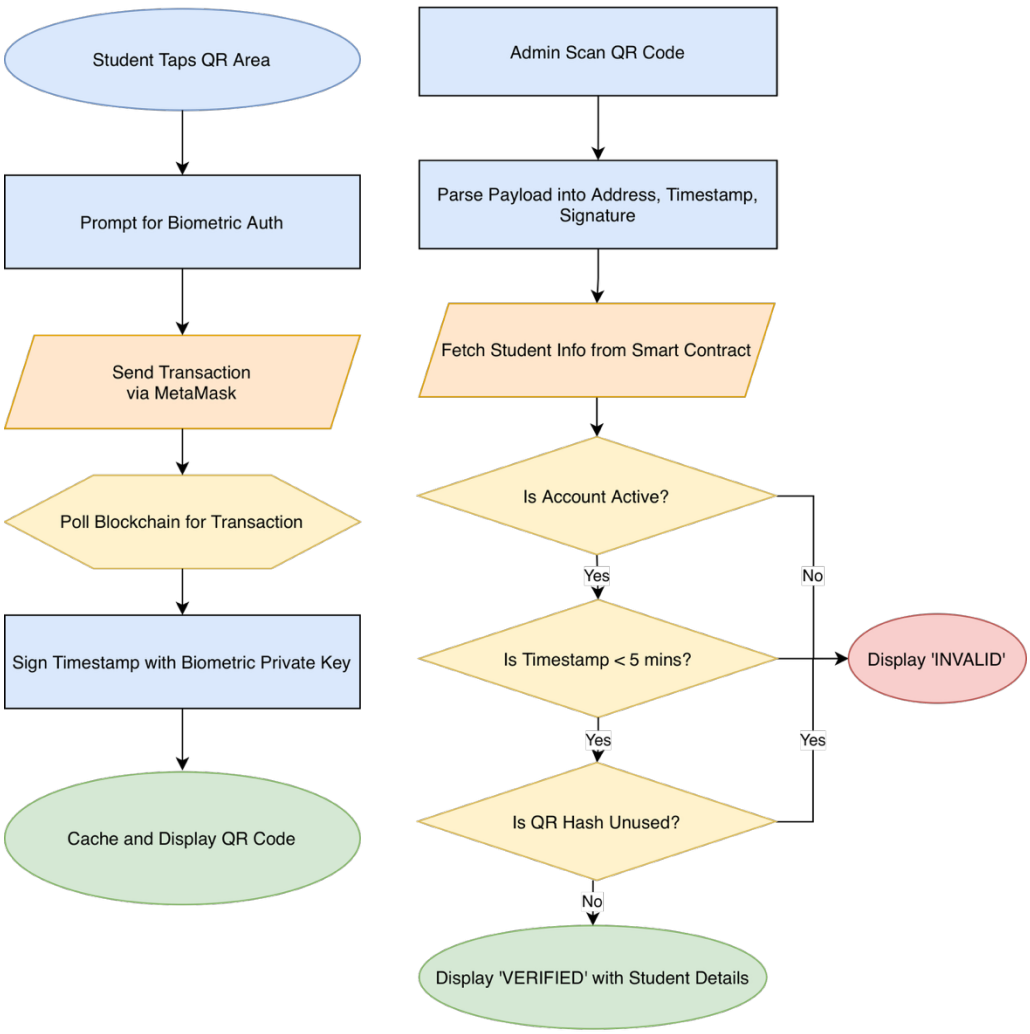

Figure 3.6 Flowchart of QR Code Generation and Verification

The Generation Flow begins when a student requests a verification code. The app first requires biometric authentication to unlock the student's device private key. It then sends a transaction to the blockchain to get a new, on-chain timestamp. This new timestamp is then signed with the biometric private key. The final QR code payload is a combination of the wallet address, timestamp and digital signature.

The Verification Flow starts when a staff member scans this QR code. The verifier's app parses the payload and performs a strict sequence of checks. First, it queries the blockchain to ensure the account is active. Next, it calculates the QR age, enforcing a strict 300-second TTL expiration to prevent replay attacks. It then fetches the student's public key from the smart contract and mathematically validates the ECDSA signature. Finally, the system checks if the QR hash has been previously used; if valid, the staff member submits a transaction to permanently burn the QR hash on-chain, rendering it single-use and securely logging the verification event.

CHAPTER 4

System Design

4.1 System Block Diagram

The system block diagram of the FRDDSID system illustrates the high-level functional modules and their data flow. As a software-centric decentralized architecture, the system is divided into four primary interactive blocks:

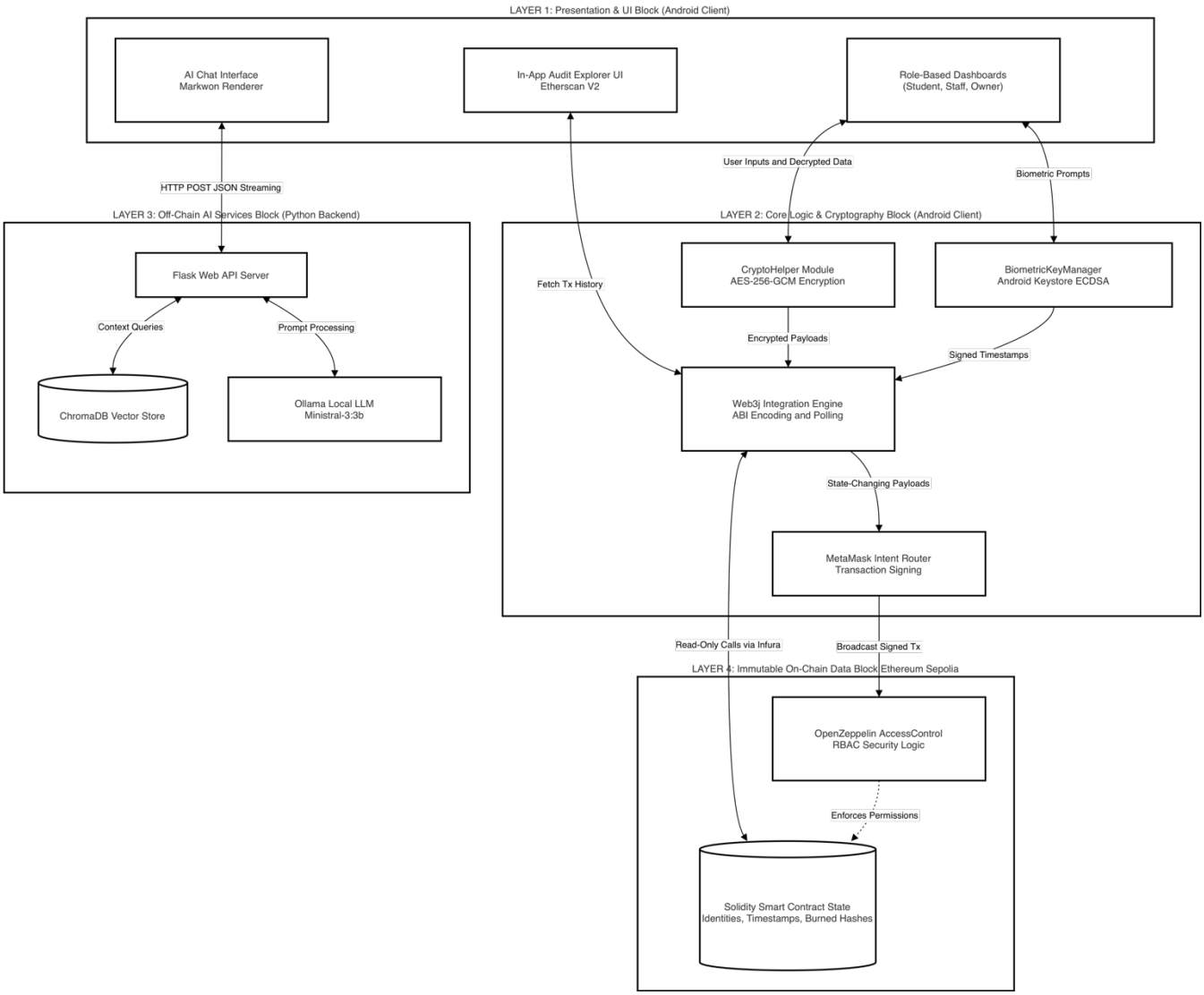

Figure 4.1 System Block Diagram

4.1.1 Client Application Block (UNICESS Frontend)

This block comprises UNICESS, the unified native Android application built in Kotlin. It includes the user interface (Material Design 3) for the Student, Staff, and Owner dashboards. Internally, it houses the CryptoHelper module for AES-256-GCM encryption/decryption of sensitive personal data and interacts with the Android Hardware Keystore to manage the biometric-backed secp256r1 ECDSA private key.

4.1.2 Authentication and Signing Block (Middleware)

This block utilizes the MetaMask Android SDK. It acts as the secure enclave for the user's Ethereum wallet. The client application delegates all state-changing transaction approvals (e.g., registering an identity, updating a timestamp, or burning a QR code) to this block, ensuring private keys are never exposed to the frontend app.

4.1.3 Blockchain Network Block (Backend)

This block represents the Ethereum Sepolia Testnet, accessed via an Infura RPC node. It hosts the deployed Solidity Smart Contract, which serves as the immutable ledger for public keys, encrypted payloads, and RBAC mappings. Read-only operations are executed directly against this block using the Web3j library.

4.1.4 Off-Chain Services Block (External APIs & AI)

This block handles data requests that are unsuitable for on-chain processing due to gas constraints or data structure limitations. It encompasses the Etherscan V2 API for fetching paginated transaction histories for the in-app Audit Explorer. Furthermore, it includes the local Python-based RAG backend, comprising a Flask Web Server, ChromaDB Vector Database, and the Ollama LLM (Ministral-3:3b), which processes natural language queries from the Android chat interface.

4.2 System Components Specification

4.2.1 Hardware

The hardware involved in this project includes a laptop and an Android mobile device. The laptop functions not only as the primary development environment for the dApp but also acts as the local host for the Python-based RAG AI server and vector database. The mobile device is utilized for deploying, testing, and analyzing the final UNICESS prototype.

Table 4.1 Specifications of Laptop

Description	Specifications
Model	MacBook Pro
Processor	Apple M4
Operating System	macOS Sequoia 15.5
Memory	16 GB Unified Memory
Storage	512 GB SSD

Table 4.2 Specifications of Mobile Device

Description	Specifications
Model	Google Pixel 10 Pro
Processor	Google Tensor G5
Operating System	Android 16
Storage	256 GB UFS 4.1

4.2.2 Software

The core software and technologies form the robust foundation of the UNICESS prototype, spanning mobile development, blockchain infrastructure, and artificial intelligence integration. In addition to Android Studio, Solidity, Sepolia, Web3j, and MetaMask, the final system incorporates Python (Flask), ChromaDB, and Ollama for the localized AI assistant, alongside the Etherscan V2 API for transaction auditing.

Table 4.3 Specifications of Software

Tool	Function	Description
Android Studio	The Integrated Development Environment (IDE) used for building Android dApp.	Provides comprehensive tools for coding, debugging and testing Android applications.
Solidity (Ethereum IDE)	The programming language used to develop and deploy the FRDDSID smart contract.	Essential for achieving the objective of decentralized system using blockchain.
Sepolia Testnet	The public Ethereum Testnet where the smart contracts will be deployed and tested.	Interact logic without incurring real transaction costs (gas fee).
Web3j	The Java/Kotlin library that provides a bridge between the Android application and the Ethereum blockchain network.	Essential communication layer between the mobile front-end and the blockchain back-end.
MetaMask	The crypto wallet and gateway to blockchain apps and used for user authentication and transaction signing.	Provides a secure interface for managing cryptographic keys and approving blockchain transactions.

4.3 Circuits and Components Design

This section presents the core architecture underpinning the system's backend, smart contracts, and RAG AI pipeline to ensure operational integrity and data privacy. Crucially, AES-256-GCM is implemented for off-chain authenticated encryption to guarantee payload confidentiality before public ledger submission. Concurrently, the ECDSA curve provides asymmetric cryptography, leveraging native Android Keystore support for optimal, hardware-backed mobile security without degrading performance.

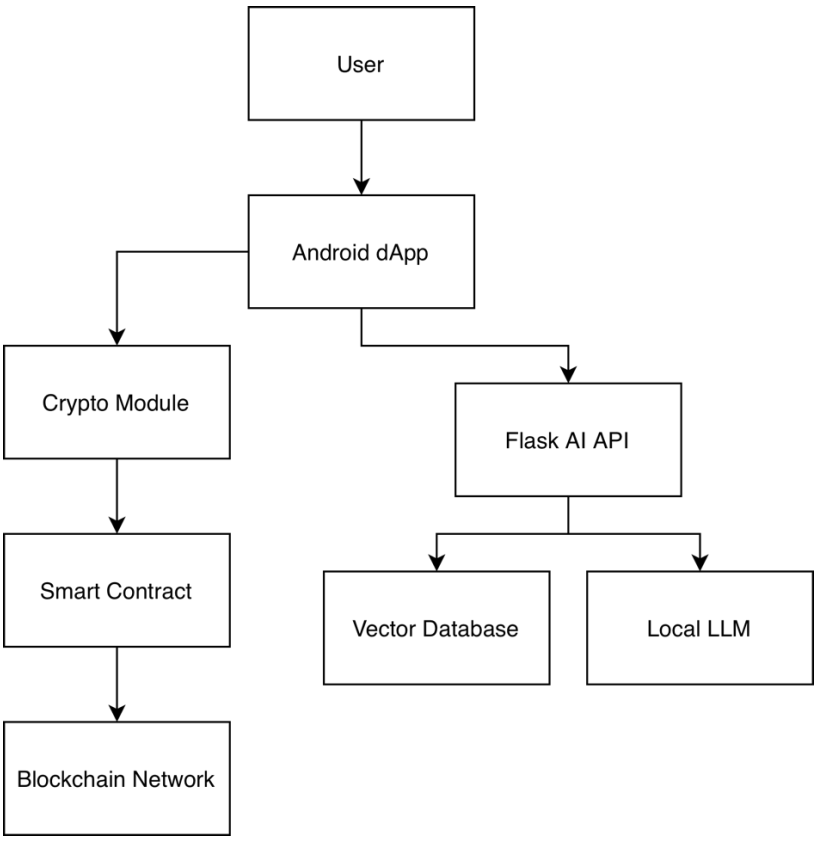

Figure 4.2 Circuits and Components Design

Explanation

- Android App: Main interface for user interaction
- Crypto Module: Handles encryption (AES) and key management (ECDSA)
- Smart Contract: Stores student data and handles verification
- Flask AI API: Processes AI queries
- Vector DB + LLM: Handles retrieval and response generation
- Blockchain Network: Executes and stores smart contract state

4.3.1 Smart Contract State Design

The core "circuitry" of the backend is the Solidity smart contract. To optimize gas efficiency and maintain strict privacy, the contract uses a specialized Student struct. Variables such as studentId and name are stored in plaintext to allow seamless querying by administrative staff. However, highly sensitive PII including gender, email, phone number, faculty, and program—is consolidated into a single bytes encryptedData payload. Additionally, the contract tracks a verificationCount and a lastGenerationTimestamp to log audit trails and prevent replay attacks. Access control is structurally defined using OpenZeppelin's RBAC, segregating DEFAULT_ADMIN_ROLE (Owner) and STAFF_ROLE (Administrators).

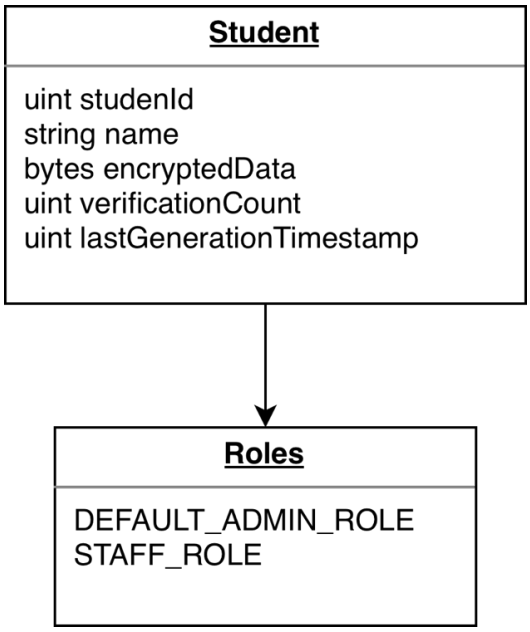

Figure 4.3 Smart Contract State Design

Explanation

- studentId, name: Publicly accessible identifiers
- encryptedData: Stores all PII securely
- verificationCount: Tracks number of verifications
- lastGenerationTimestamp: Prevents replay attacks
- Roles: Controls access using RBAC

4.3.2 Cryptographic Component Design

The system's security relies on a dual-cryptography model:

Symmetric Encryption (AES-256-GCM)

Handled entirely off-chain by the Android CryptoHelper class. PII is converted into a JSON object and encrypted using a symmetric master key. The resulting ByteArray is submitted to the blockchain. Only authenticated staff and the student's own device possess the logic to decrypt this payload, ensuring public blockchain observers cannot extract personal data.

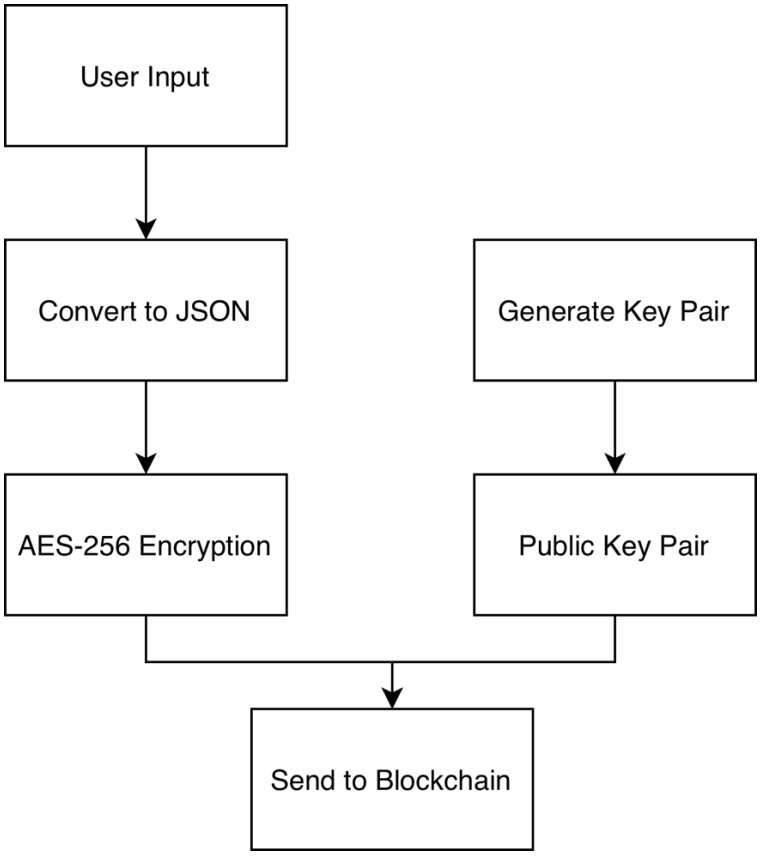

Figure 4.4 Symmetric Encryption Design

Asymmetric Cryptography (ECDSA secp256r1)

Handled by the Android BiometricKeyManager. A hardware-backed key pair is generated upon registration. The private key remains permanently locked inside the Android hardware chip, accessible only via a successful biometric prompt. The public key is submitted to the smart contract, enabling the verification module to mathematically prove that a QR code was generated by the legitimate device owner.

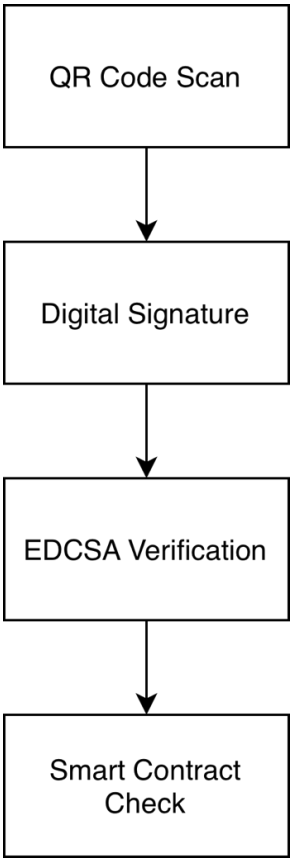

Figure 4.5 Asymmetric Cryptography Design

Explanation

- AES Encryption: Protects PII before storage
- Key Pair: Generated securely on device
- ECDSA Verification: Confirms authenticity of user
- Smart Contract Check: Validates legitimacy

4.3.3 RAG AI Pipeline Design

The AI component is designed as an isolated, standalone pipeline to prevent bloating the mobile app. A Python script utilizes trafilatura and BeautifulSoup to scrape university websites, chunking the text into a ChromaDB vector store using NomicOllamaEmbeddings. A lightweight Flask server exposes a /api/chat endpoint over the local network. When the Android app sends an HTTP POST request, the Flask server retrieves the top 30 most relevant contextual chunks, injects them into a PromptTemplate, and streams the generated response from the local Ministral-3:3b LLM back to the Android RecyclerView in real-time JSON chunks.

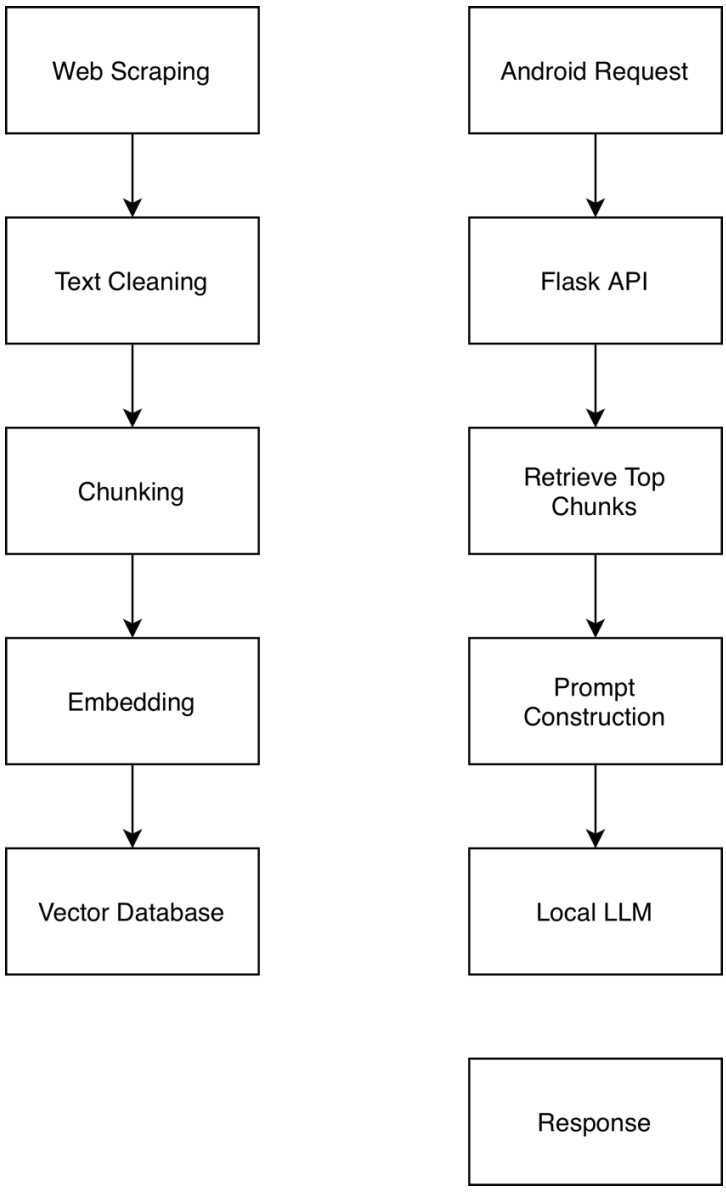

Figure 4.6 RAG AI Pipeline Design

The architectural decision to deploy a localized Python Flask server routing queries to an open-source, locally hosted LLM (Ministral-3:3b via Ollama) rather than utilizing commercial, cloud-based APIs (such as OpenAI's ChatGPT) was driven by strict institutional data privacy requirements. Utilizing a localized RAG pipeline ensures absolute data sovereignty; highly sensitive internal university policies, student queries, and intellectual property are never transmitted to third-party corporate servers for processing. Furthermore, hosting the ChromaDB Vector Database and the LLM natively on institutional hardware eliminates the unpredictable, recurring operational costs associated with token-based cloud API billing, creating a highly sustainable, self-sufficient technological ecosystem for the university.

4.4 System Components Interaction Operations

The system's operational integrity relies on the seamless interaction between its core software components. This decentralized architecture ensures that foundational elements securely communicate to execute complex features such as cryptographic verification, data management, and AI-driven assistance.

- Smart Contract & Mobile Application Interaction: The primary interaction involves the UNICESS reading and writing to the deployed FRDDSID smart contract. The smart contract acts as the "single source of truth" on the blockchain. Key interaction operations include:
- Data Encoding: UNICESS utilizes Web3j to encode complex parameters (such as the AES-encrypted data and the ECDSA public key byte array) into a raw hexadecimal payload matching the contract's Application Binary Interface (ABI).
- State Execution: For functions like registerStudent and recordVerification, the encoded payload is passed to the MetaMask SDK, which prompts the user to sign the transaction and broadcasts it to the Sepolia network.
- Role Validation: Upon UNICESS launch, the Android app sends read-only eth_call requests via Web3j to the contract’s RBAC mappings to dynamically determine if the connected wallet is a Student, Staff, or Owner.

With the on-chain backend established, specific interaction pipelines were configured to handle external APIs and asynchronous tasks. The main operational flows include:

- Wallet Connection & Network Enforcement: The Android app initiates an eth_requestAccounts call via the MetaMask SDK. To prevent the user from transacting on the Ethereum Mainnet, the app immediately sends a wallet_switchEthereumChain payload, forcing MetaMask to lock onto the Sepolia Testnet.
- Real-Time Data Streaming: The Android ChatFragment interacts with the Python Flask server via standard HttpURLConnection. The app sends a JSON payload containing the user's query and opens a BufferedReader to intercept the Continuous chunked HTTP response. As the AI generates text, the UI updates the Markwon markdown renderer progressively, simulating a real-time typing effect without blocking the main UI thread.

This phase focused on developing the project's core innovation which is the forgery resistant system.

- Biometric Key Integration (Generate cryptographic key pair in Android Keystore)
- Timestamp Transaction (Create a fresh, on-chain timestamp)
- Dynamic Polling (Dynamically wait for the timestamp transaction to be mined)
- QR Code Generation and Verification (Fetch the corresponding public key from the contract and cryptographically verify the signature)

CHAPTER 5

System Implementation

5.1 Hardware Setup

The hardware setup required for the practical implementation of the FRDDSID system involved the physical interconnection and deployment of the development machines. The primary development laptop (macOS) was configured not only to write and compile the Android and Solidity codebases but also to operate as a localized backend server. It hosted the Python virtual environment, the ChromaDB instance, and the Ollama LLM.

Simultaneously, the Android mobile device was connected to the same local Area Network (LAN) via Wi-Fi. This localized hardware configuration was critical to bypass standard Android cleartext traffic restrictions, allowing the UNICESS's frontend to communicate directly with the laptop's IP address (e.g., 192.168.x.x:5000) for testing the RAG AI chat functionality without deploying the heavy LLM to a public cloud server. The Android device was also configured in Developer Mode to allow direct USB debugging and instantaneous APK deployment from Android Studio.

5.2 Software Setup

The software implementation process transitioned the conceptual architecture into a functional prototype. The implementation was divided into three primary environments.

5.2.1 Blockchain Deployment Environment

The smart contract was written using Remix IDE (Solidity v0.8.20). OpenZeppelin's AccessControl library was imported directly from GitHub to provide battle-tested, standard-compliant RBAC. Upon successful compilation, the contract was injected into the Ethereum Sepolia Testnet using a developer MetaMask wallet funded by a Sepolia ETH faucet. The resulting Contract Address and ABI were extracted for integration.

5.2.2 UNICESS Frontend Environment

The UNICESS was constructed in Android Studio using Kotlin. The project was structured around a Single-Activity Architecture utilizing Fragments (HomeFragment, ScanFragment, VerifyFragment, etc.) to ensure seamless navigation via a BottomNavigationView. Crucial dependencies were integrated via Gradle, including Web3j for raw blockchain read operations, the MetaMask Android SDK for transaction signing, ZXing for QR code generation and scanning, and Markwon to render AI Markdown responses.

5.2.3 Python AI Backend Environment

A Python 3 virtual environment was initialized on the host laptop. The RAG pipeline was established using langchain, chromadb, and trafilatura. The ollama daemon was executed locally (ollama serve) to run the ministral-3:3b and nomic-embed-text:v1.5 models. Finally, a lightweight Flask web server was implemented to expose the /api/chat endpoint to the UNICESS.

5.3 Setting and Configuration

To ensure secure and reliable communication across the disparate layers of the system, precise configuration parameters were established.

5.3.1 Android Network Security Configuration

To permit the UNICESS to send HTTP POST requests to the local Python Flask server (which lacked an SSL/TLS certificate), the AndroidManifest.xml was updated with the android:usesCleartextTraffic="true" attribute.

5.3.2 MetaMask SDK and Sepolia Network Enforcement

The MetaMask SDK was configured within the AppModule.kt using Dependency Injection. To prevent users from accidentally executing transactions on the Ethereum Mainnet (which would incur real financial costs), a mandatory configuration payload was added to the login flow. Immediately upon connection, the application executes an eth_sendRequest containing wallet_switchEthereumChain with the parameter chainId = 0xaa36a7, forcing the MetaMask app to lock onto the Sepolia Testnet.

5.3.3 Infura RPC Configuration

For read-only blockchain queries (e.g., fetching a student's public key or checking RBAC roles), the Web3j instance was configured to route requests through a dedicated Infura Sepolia RPC endpoint (https://sepolia.infura.io/v3/...). This eliminated the need for the mobile device to synchronize a full Ethereum node.

5.3.4 Etherscan API Configuration

The in-app Audit Explorer was configured to query the Etherscan V2 API endpoint (api.etherscan.io/v2/api). A unique API key was registered and appended to the URL alongside the specific chain ID (11155111) and the FRDDSID Smart Contract address to fetch paginated, descending transaction histories.

5.4 System Operation

5.4.1 User Authentication and Network Routing

Upon launching the UNICESS, the user is presented with a clean, centralized "Connect MetaMask" interface. Initiating the connection triggers a deep link to the MetaMask application installed on the device. To safeguard the user from accidentally executing transactions on the Ethereum Mainnet (which would incur real financial costs), the application immediately broadcasts an RPC payload requesting a forced network switch to the Sepolia Testnet.

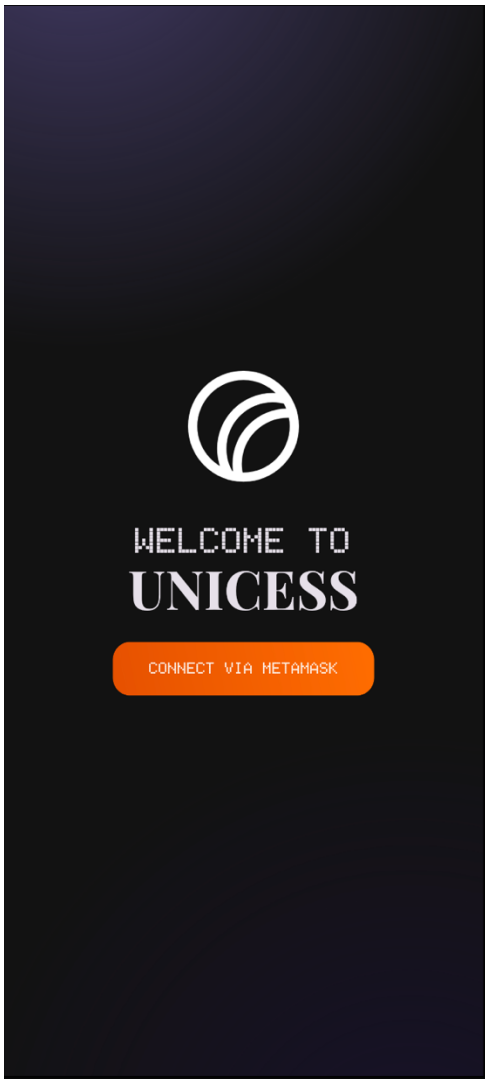

Figure 5.1 UNICESS Login Screen

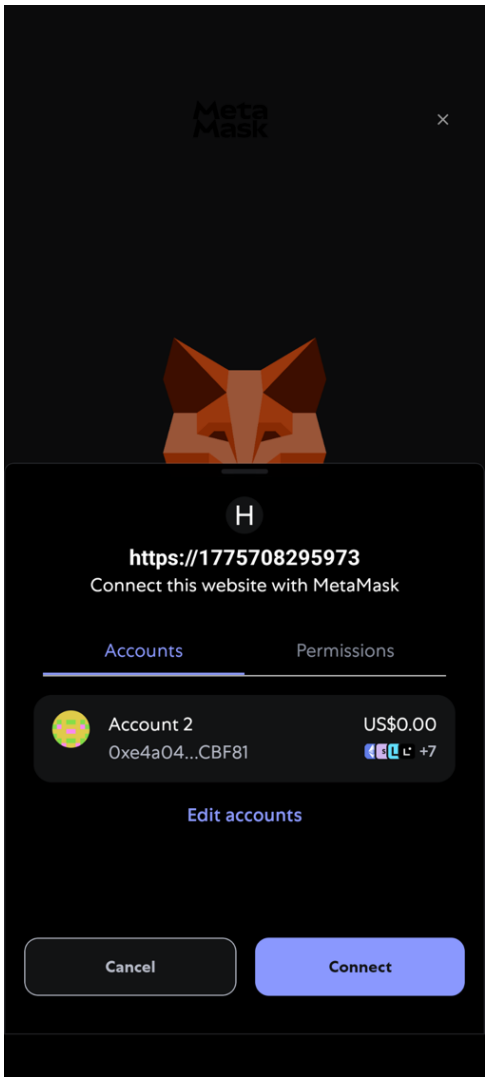

Figure 5.2 MetaMask Prompt for Network Switch to Sepolia

Once the connection is established and the network is verified, the system retrieves the user's wallet address and executes a sequence of read-only queries against the smart contract's RBAC mappings. The routing engine dynamically redirects the user: wallets holding the DEFAULT_ADMIN_ROLE are routed to the Owner Panel, STAFF_ROLE to the Verify Panel, and registered addresses to the Student Home. Unrecognized wallets are safely routed to the Registration Form.

5.4.2 Student Registration and Identity Management

Unregistered users are presented with a comprehensive registration form requiring their Full Name, Student ID, Gender, Email, Phone, Faculty, and Program. The UNICESS enforces strict input validation, including automated formatting for phone numbers and regex pattern matching for email addresses.

Upon submission, the UNICESS isolates the highly sensitive PII and encrypts it off-chain using AES-256-GCM. Concurrently, a biometric secp256r1 ECDSA key pair is generated securely within the Android Keystore. The encrypted payload, along with the generated public key, is then passed to MetaMask for transaction signing. Students can later update their encrypted details via the Edit Information Panel. For security and data integrity, submitting an update automatically reverts the account's on-chain status to "Pending", mandating a formal re-approval by university staff.

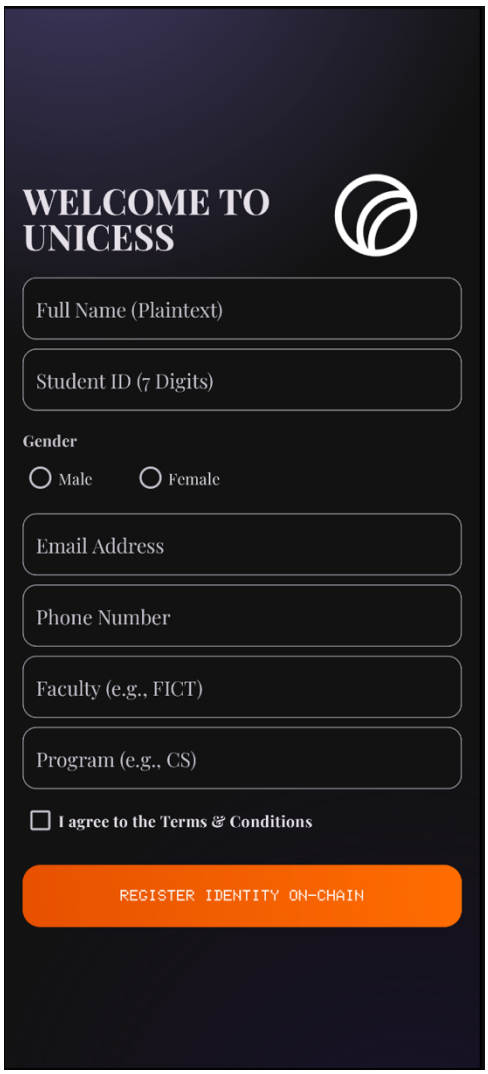

Figure 5.3 Registration Form

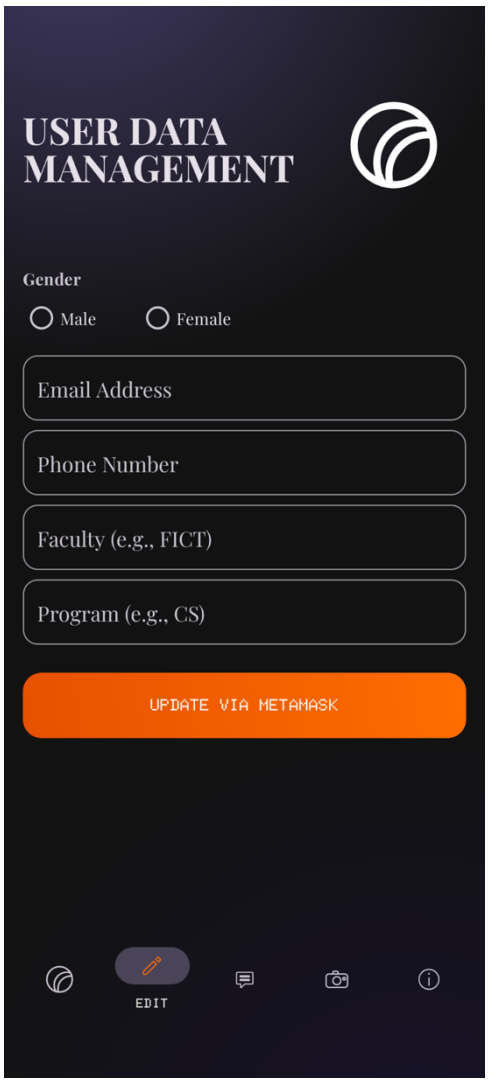

Figure 5.4 Edit Information Panel

5.4.3 Dynamic QR Code Generation (Student)

Upon successful registration and staff approval, the student gains access to the Home Screen, which prominently displays their Active status and a blurred QR code placeholder. To generate the verification credential, the student taps the placeholder, triggering the Android Biometric Authentication Prompt. It is highly notable that the Android Operating System natively blacks out the screen during this prompt if a screenshot is attempted, physically demonstrating the hardware-level security protecting the private key.

Following successful biometric authentication, a transaction is sent via MetaMask to update the on-chain timestamp. Once the network mines this transaction, the application utilizes the unlocked private key to digitally sign the fresh timestamp. A dynamic QR code is then rendered on-screen, containing the wallet address, timestamp, and cryptographic signature, accompanied by a strict 300-second valid countdown timer to visually indicate the TTL expiration. Furthermore, students can tap a "View More Details" button, which decrypts their on-chain PII payload locally and displays it in a secure pop-up window, showcasing the successful implementation of the off-chain encryption system.

Figure 5.5 Student Home Screen showing Status and QR Code

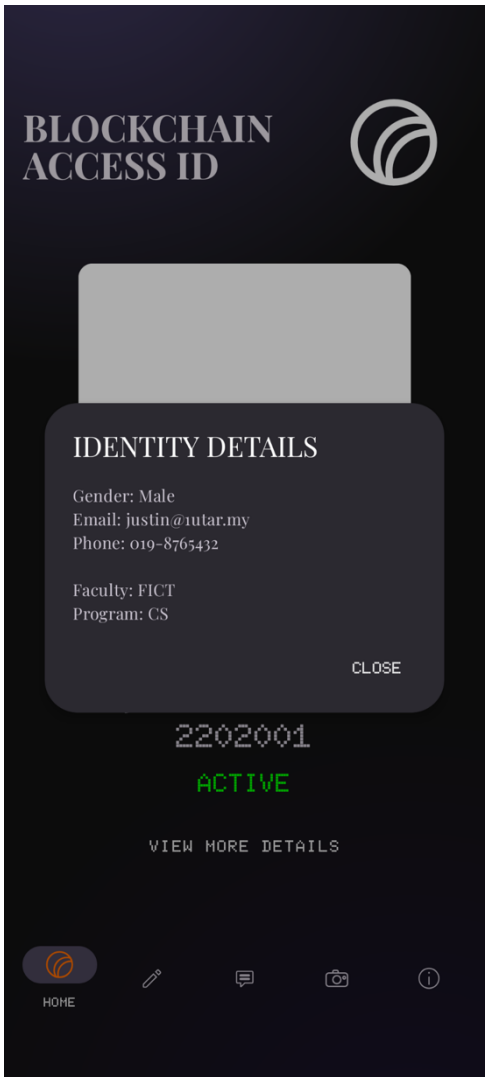

Figure 5.6 The "View More Details" Section

5.4.4 Verification and Identity Authentication (Staff)

A university staff member navigates to the Scan panel to capture a student's dynamic QR code using the device camera. The application instantly parses the scanned payload and executes a strict sequence of mathematical validations. It first ensures the 300-second TTL has not expired. It then fetches the student's public key from the smart contract and mathematically verifies the ECDSA signature to guarantee the QR code was generated by the legitimate device owner.

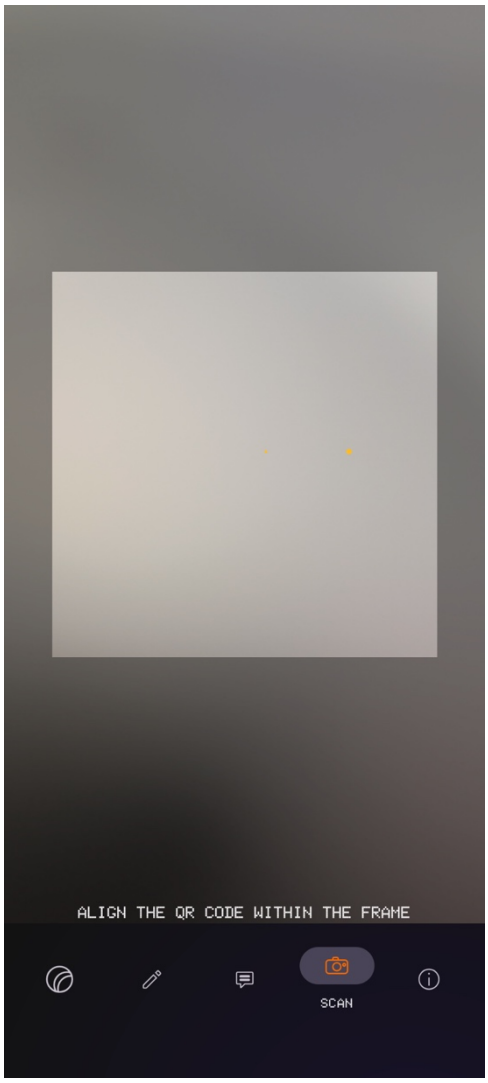

Figure 5.7 Scanner Interface Capturing QR code

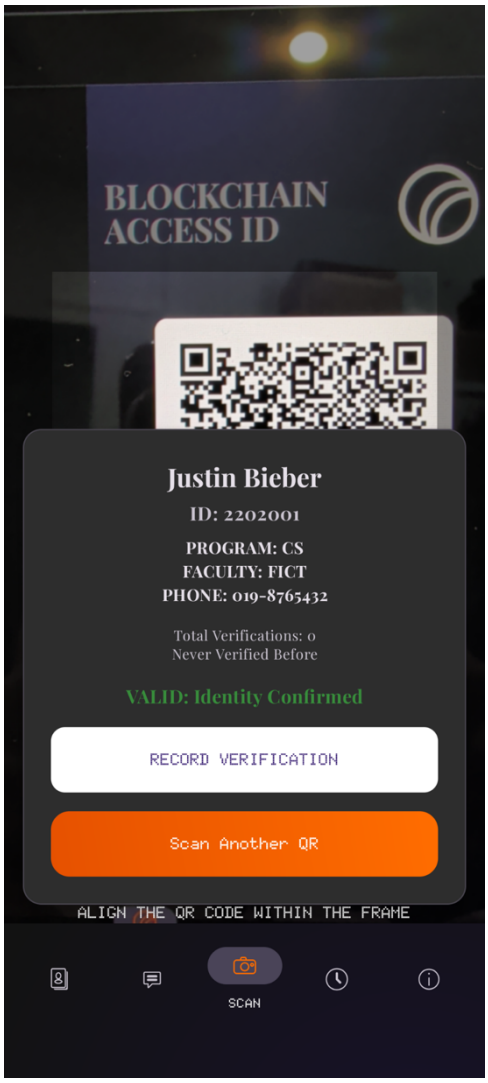

Figure 5.8 Verification Result showing Decrypted Student Details

If the validation fails (e.g., the credential has expired, the account is inactive, or the signature is invalid), the interface immediately displays a Failed Verification Result with explicit error reasoning. If valid, the system retrieves and decrypts the student's PII, displaying a Successful Verification Result alongside their historical verification count. The staff member can then tap the "Record Verification (On-Chain)" button, which executes a transaction to permanently burn the unique QR hash on the blockchain, effectively neutralizing any possibility of a screenshot replay attack.

Figure 5.9 Failed Verification Result indicating an "Expired QR Code"

5.4.5 Administrative Identity Management (Staff)

The Verify Panel acts as the primary administrative dashboard for university staff. It provides a comprehensive, scrollable list of all students registered on the smart contract. To optimize the administrative workflow, pending accounts awaiting approval are automatically prioritized and sorted to the top of the list.

Figure 5.10 Verify Panel showing the List of Students

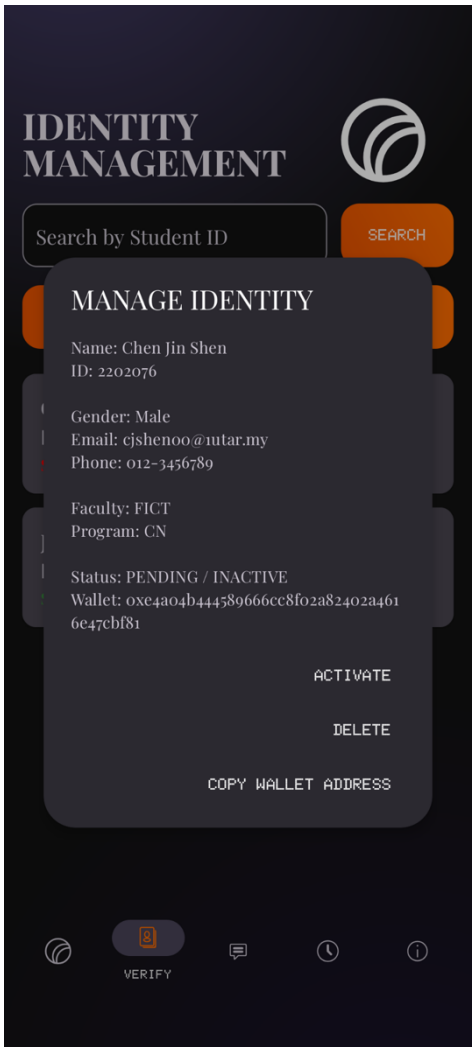

Figure 5.11 Details with "Activate", "Delete"& "Copy Wallet" options

Staff can utilize the search bar to locate specific students by their ID. Tapping on a student entry opens a detailed Material Dialog. This dialog displays the fully decrypted PII and provides definitive administrative actions to "Activate" or "Delete" the identity. Additionally, a "Copy Wallet" function is embedded within this dialog. This is specifically useful for extracting a student's exact blockchain address if their account needs to be escalated to an administrative role by the Owner.

5.4.6 Organizational Hierarchy Management (Owner)

The Owner Panel provides supreme administrative control to the deployer of the smart contract. The interface features a specialized text input field that automatically enforces hexadecimal formatting (prepending "0x"). The owner can input an Ethereum wallet address and execute transactions to directly Assign or Revoke the STAFF_ROLE.

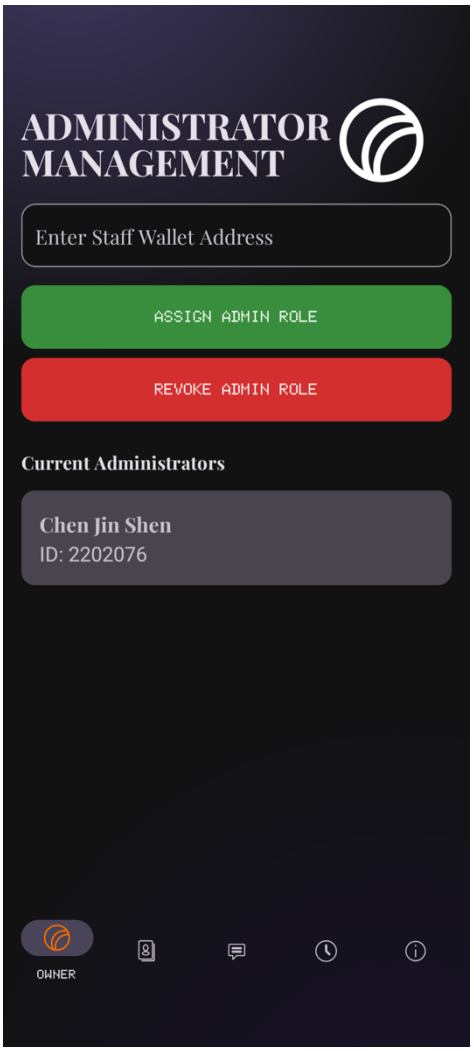

Figure 5.12 Owner Panel for Assigning/Revoking Staff

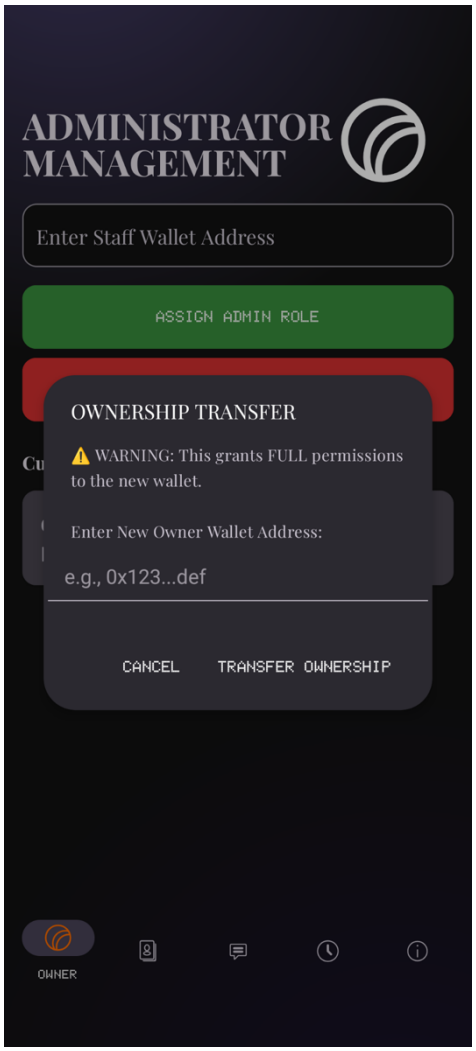

Figure 5.13 Ownership Transfer Dialog Confirmation

To accommodate institutional succession planning, a hidden security feature is embedded within the university logo at the top of the fragment. Tapping the logo summons a Transfer Ownership Warning Dialog. This interface allows the current owner to permanently grant the DEFAULT_ADMIN_ROLE to a new co-owner wallet address, safely shifting smart contract governance without requiring a complete redeployment of the system.

5.4.7 Blockchain Audit Explorer

To enforce absolute institutional transparency while significantly enhancing the user experience, an Audit Log Panel is integrated directly into the UNICESS. Traditionally, auditing smart contract activity requires users to navigate to external blockchain explorers (such as the Sepolia Etherscan website) and interpret raw, unreadable hexadecimal payloads—a process that demands technical expertise and creates unnecessary friction.

The UNICESS in-app explorer streamlines this process by securely querying the Etherscan V2 API to retrieve a chronologically sorted list of all smart contract interactions and presenting them within a native, highly legible UI. Instead of displaying raw bytecode, the application dynamically calculates Solidity method hashes to label transactions with clear, human-readable nomenclature (e.g., "Account Registered", "Updated Account Info"). Furthermore, it queries the blockchain to automatically resolve cryptic wallet addresses, replacing them with the registered names of students, staff members, or the UTAR Admin. By centralizing and translating this complex blockchain data within the app, the system empowers non-technical administrative staff to easily and efficiently monitor the system's audit trail without leaving the application.

Figure 5.14 Audit Log Panel

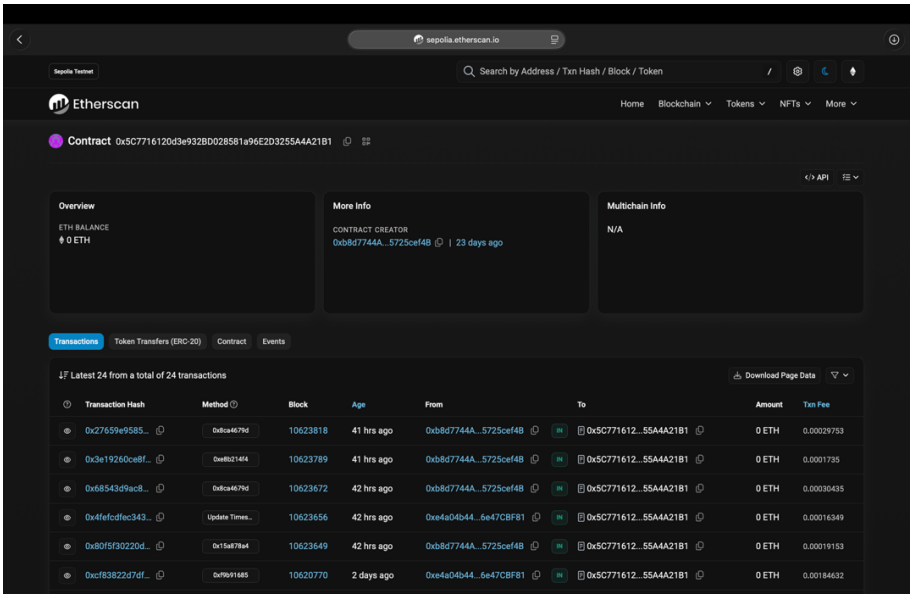

Figure 5.15 Etherscan Audit Interface

5.4.8 RAG AI Chatbot

The UNICESS features a built-in AI Assistant to answer natural language queries regarding university policies. To configure the connection to the localized Large Language Model (LLM), users tap the top-right logo to open a settings dialog and input the local Wi-Fi IP address of the Python server.

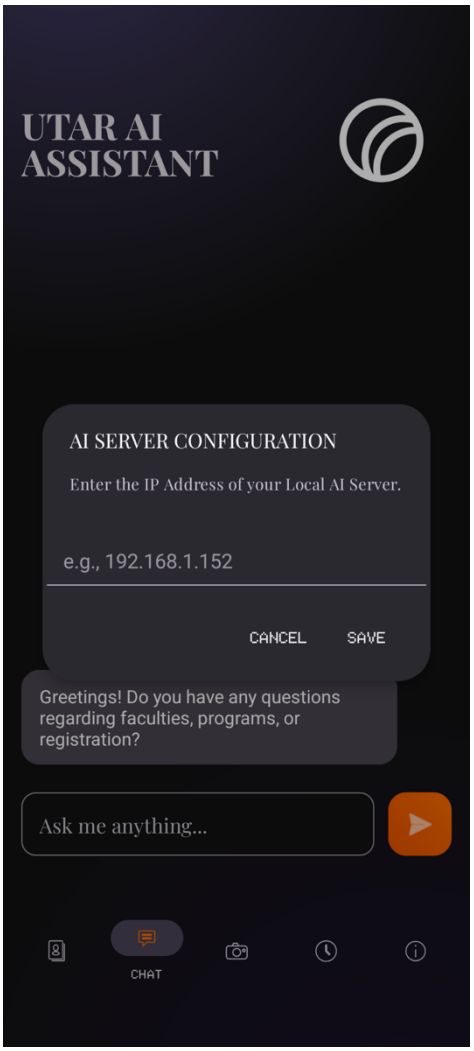

Figure 5.16 Settings Panel IP for local LLM server

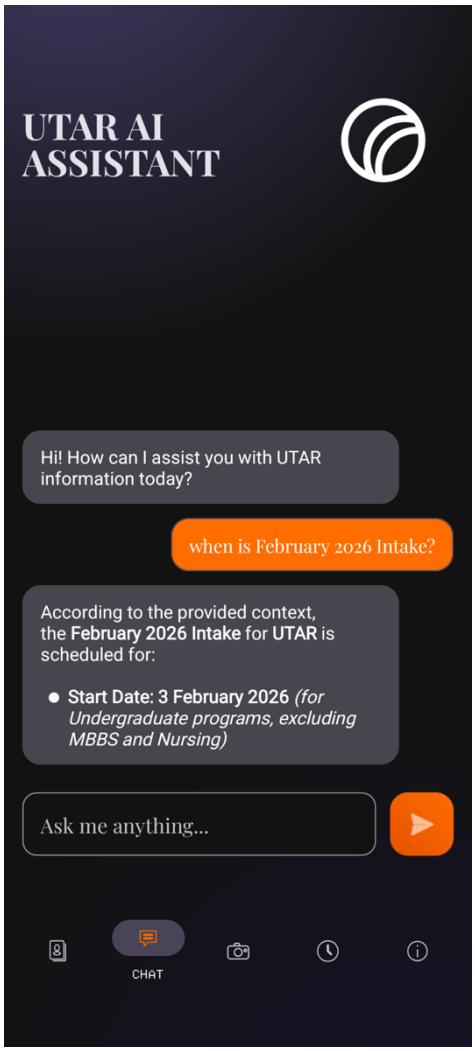

Figure 5.17 AI Chatbot Interface

Once configured, the UNICESS transmits user queries via HTTP to the Python Flask backend. The backend executes a RAG pipeline—searching the ChromaDB vector database and routing the context to the local Ollama LLM. The generated response is streamed back to the Android RecyclerView in real-time, providing a smooth typing effect. The text is parsed dynamically using the Markwon markdown library, allowing the AI to present structured lists, bold text, and formatted paragraphs seamlessly.

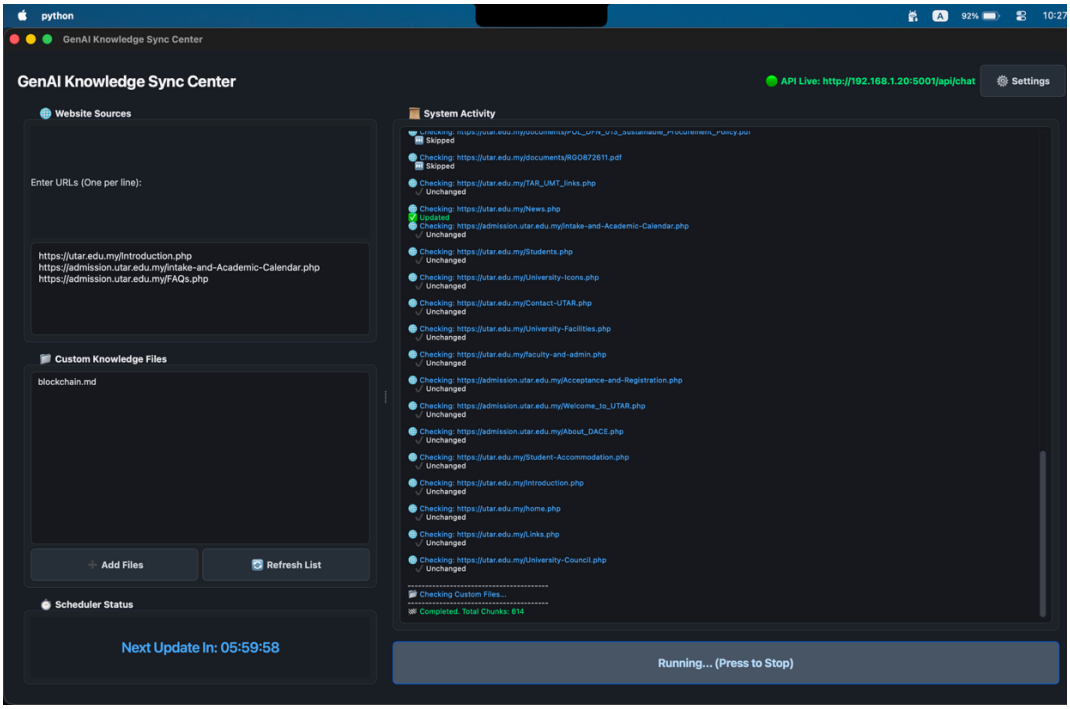

Figure 5.18 Python GUI running on the Host Server

5.4.9 System Information and Metadata

Finally, the UNICESS includes an "About" fragment accessible from the bottom navigation menu. This interface provides users with an overview of the system's architecture and offers quick-access options to view their own connected wallet address, the deployed smart contract address, and the resolved name of the contract owner. It also includes integration intents to open the smart contract address directly in the external Sepolia Etherscan web browser for independent verification.

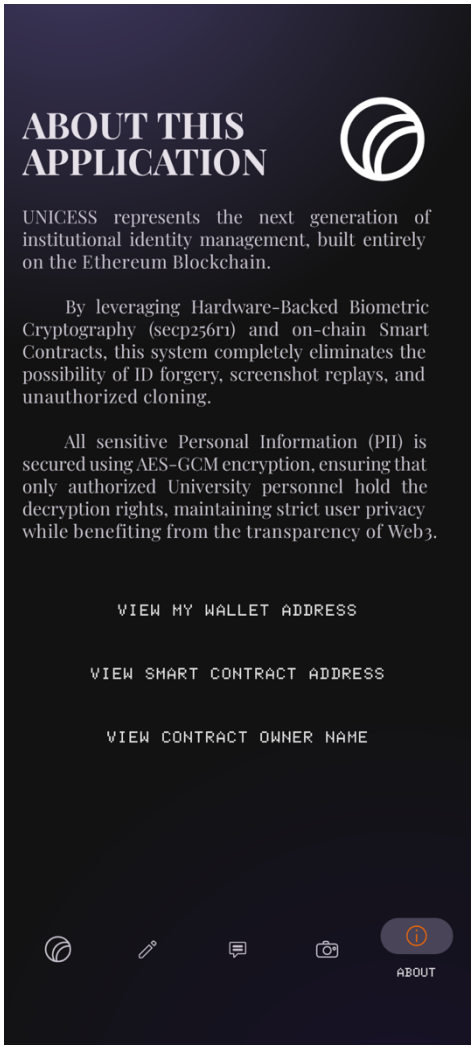

Figure 5.19 The About Section of UNICESS

5.5 Implementation Issues and Challenges

The practical execution of the FRDDSID system revealed significant challenges related to mobile-blockchain synchronization and Android OS lifecycle management.

5.5.1 Android Lifecycle and MetaMask Context Switching

A critical issue occurred during the QR code generation and registration phases. When the UNICESS initiated an eth_sendTransaction request, the Android OS forced the UNICESS app into the background to bring the MetaMask app to the foreground for user approval. Upon returning, the UNICESS app would occasionally lose its state or display a blank screen, as the MetaMask SDK required time to re-inject the selected wallet address. This was resolved by implementing a sophisticated polling mechanism within the onResume lifecycle method, forcing the UNICESS to wait and display a loading animation until the wallet address was successfully re-established.

5.5.2 Blockchain Propagation and Asynchronous Delays

Because blockchain transactions are not instantaneous, attempting to generate the cryptographic QR code immediately after the user approved the timestamp transaction resulted in critical failures, as the data had not yet been mined into a block. To mitigate this, a dynamic, non-blocking polling loop was engineered using Kotlin Coroutines (Dispatchers.IO). The UNICESS repeatedly queries ethGetTransactionReceipt every 3 seconds until the transaction is confirmed as "mined" by the Infura node, only then proceeding to sign the payload and generate the QR code.

5.5.3 UI Thread Blocking (NetworkOnMainThreadException)

During the integration of the Etherscan API and the Web3j polling functions, the UNICESS frequently crashed due to network operations executing on the main UI thread. This was systematically resolved by strictly encapsulating all HttpURLConnection and Web3j.send() calls within background coroutine scopes,

utilizing withContext(Dispatchers.Main) solely for updating UI elements such as TextViews and RecyclerView adapters.

5.6 Concluding Remark

In conclusion, the implementation phase successfully transformed the theoretical architecture into a robust, dApp. By leveraging Kotlin Coroutines for asynchronous blockchain polling, integrating hardware-backed biometrics for ECDSA signing, and establishing a unified UI with strict RBAC, the system effectively mitigates the vulnerabilities of traditional digital IDs. Furthermore, overcoming challenges related to context switching and propagation delays resulted in a highly polished user experience. The successful integration of an off-chain AES encryption pipeline, an in-app audit explorer, and a local RAG AI assistant confirms that complex Web3 technologies can be securely and practically deployed within a mobile university environment.

CHAPTER 6

System Evaluation and Discussion

6.1 System Testing and Performance Metrics

To quantitatively evaluate the technical performance and responsiveness of the finalized UNICESS, a series of structured latency tests were conducted. The objective was to measure the precise time taken for critical on-chain interactions, thereby benchmarking the application's efficiency in a real-world, decentralized scenario. All tests were performed on a Google Pixel 10 Pro device connected to a stable Wi-Fi network, interacting directly with the Solidity smart contract deployed on the Ethereum Sepolia Testnet via the Infura RPC node and MetaMask SDK. The evaluation focused on separating read-only operations (which do not require gas or mining) from state-changing operations (which require user interaction and block confirmations).

6.2 Testing Setup and Result

6.2.1 Read Latency Testing

This test was designed to measure the latency of a read-only call to the smart contract using the Web3j library. The function tested was getInfoByWalletAddress(), which is heavily utilized during the initial routing phase to determine a user's RBAC status and during the administrative search feature. The latency was measured programmatically from the moment the HTTP request was sent to the Infura node until the UNICESS received the complete, decoded student data payload.

The test was repeated 15 times to ensure a reliable average and to account for minor network fluctuations. The duration of each call was captured and logged for analysis.

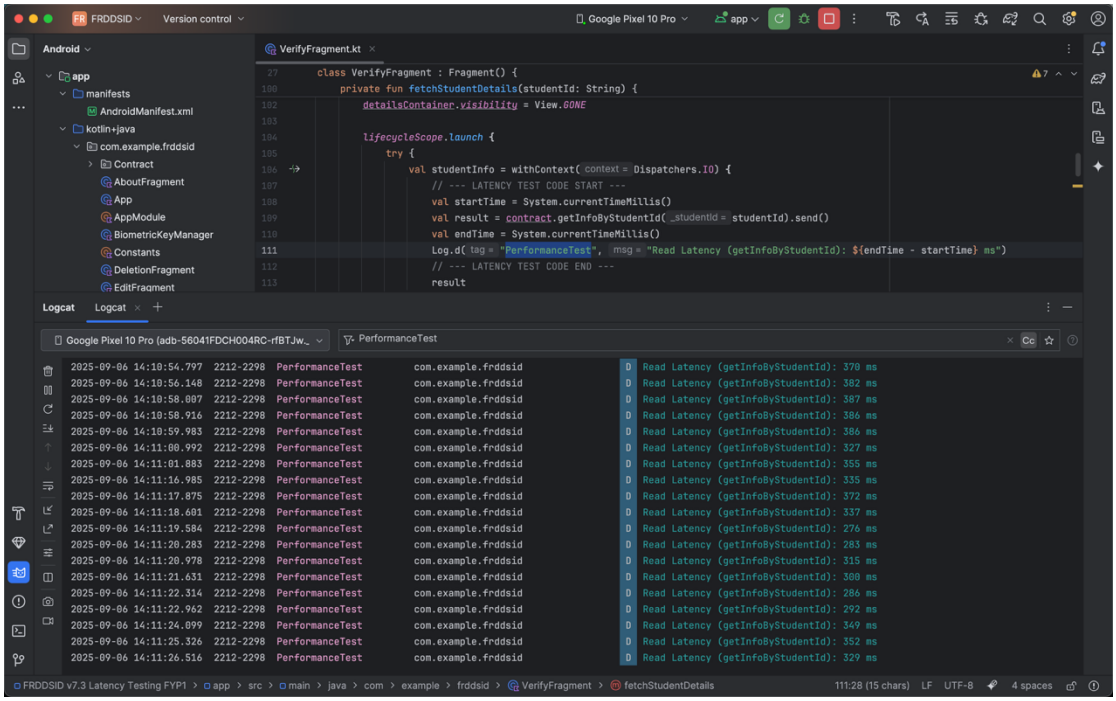

Figure 6.1 Logcat of Read Latency Testing

Table 6.1 Calculation of Read Latency Testing

Trial	Total Latency (ms)	Avg. Latency (ms)	Standard Deviation (ms)
15	5130	342	34.8

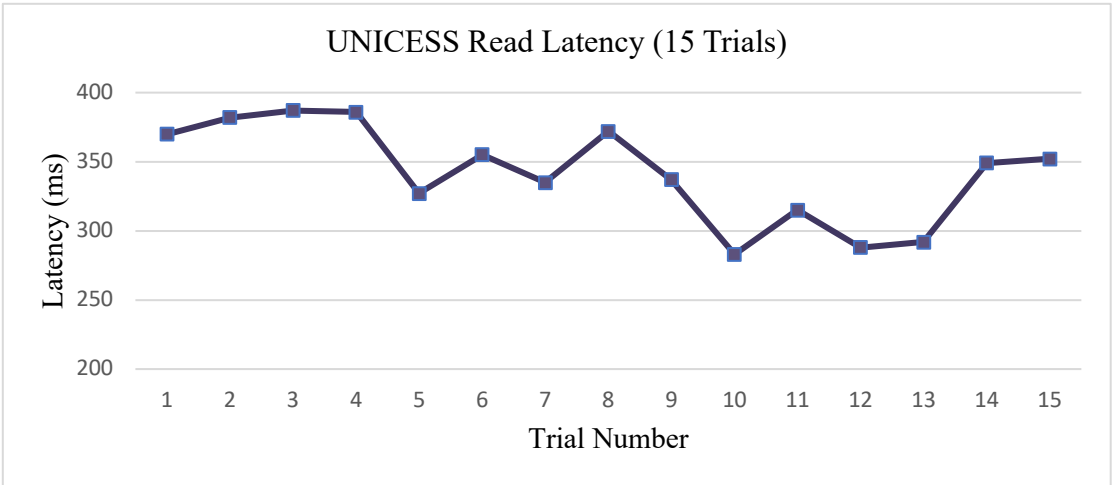

Figure 6.2 Chart of Read Latency Testing

The average read latency was found to be 342 milliseconds, with a low standard deviation of 34.8 ms. This result is excellent, demonstrating that read-only operations within the application such as loading the student profile or checking if a wallet holds the STAFF_ROLE—are highly responsive and provide a near-instant user experience comparable to traditional centralized databases.

6.2.2 QR Generation Latency Testing (State-Changing Transaction)

This test measured the time required for the entire user flow of sending a transaction. This includes the time the user spends interacting with the MetaMask wallet to approve the request, followed by the time required for the Ethereum network to mine the block. The measurement was taken from the moment the UNICESS initiated the transaction request (ethereum.sendRequest) to the moment the dynamic polling mechanism confirmed the transaction receipt was present on the blockchain. This metric is crucial as it represents the total "wait time" for a student to generate a valid QR code.

The transaction triggered during the "Update Timestamp" phase of the QR code generation was used as the test case. The test was repeated 10 times to ensure a reliable average.

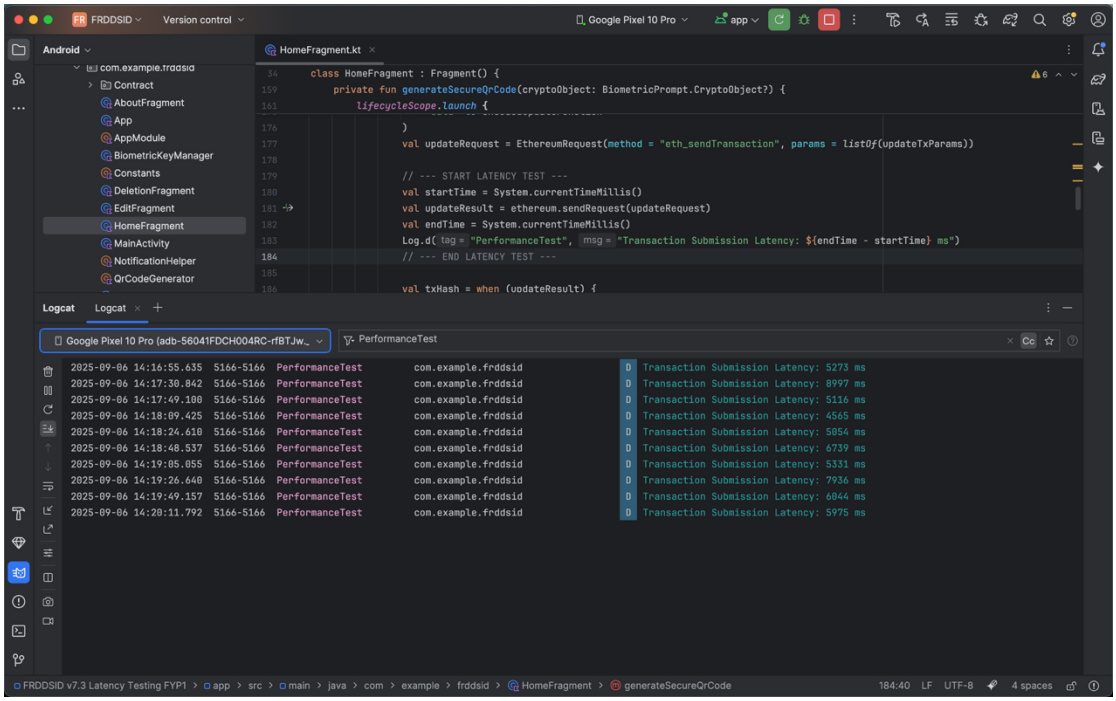

Figure 6.3 Logcat of QR Generation Latency Testing

Table 6.2 Calculation of QR Generation Latency Testing

Trial	Total Latency (ms)	Avg. Latency (ms)	Standard Deviation (ms)
10	60942	6094.2	1412.8

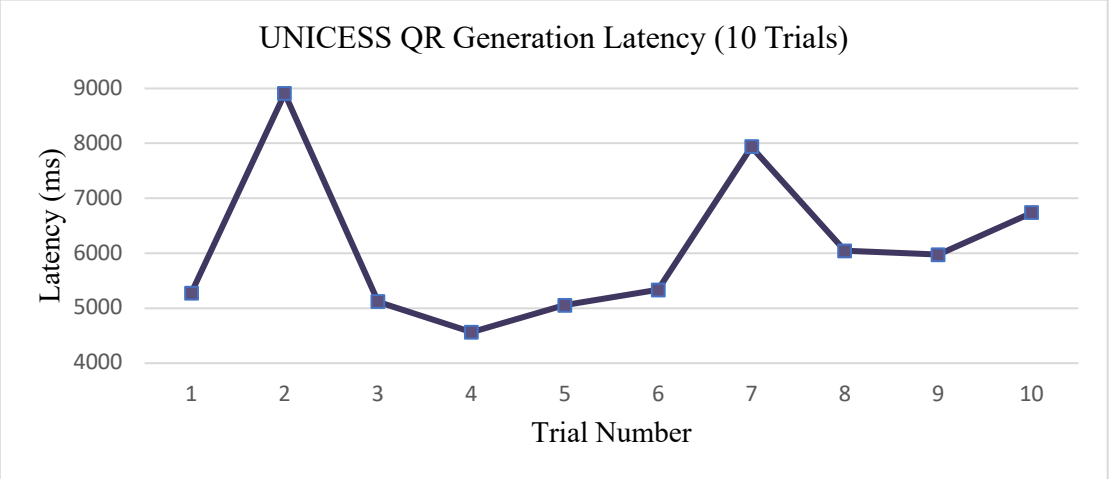

Figure 6.4 Chart of QR Generation Latency Testing

The average transaction submission and confirmation latency was approximately 6.1 seconds. It is important to note that this metric is heavily influenced by user interaction time (how quickly they press "Confirm" in MetaMask) and the current block time of the Sepolia Testnet (which averages 12 seconds). Nonetheless, the result confirms that the transaction flow is highly efficient. The implementation of the background polling mechanism successfully masks the network delay, ensuring the QR code is only displayed once the cryptographic timestamp is securely immutable on-chain.

6.3 Security and Threat Mitigation Evaluation

A primary objective of the FRDDSID system is to establish a secure, forgery-resistant environment. To validate the system's defensive capabilities, it was evaluated against three primary threat vectors targeting digital identity platforms. The system successfully neutralized all simulated attacks through its multi-layered cryptographic architecture.

6.3.1 Screenshot Replay Attack

In this scenario, a malicious actor attempts to bypass physical security by capturing a screenshot of a legitimate student's generated QR code and presenting it to a staff scanner later. To mitigate this, the system enforces a strict 300-second TTL expiration mechanism. During scanning, the UNICESS compares the timestamp embedded within the QR code against the current block time, and if the code exceeds five minutes in age, it is immediately rejected. Additionally, upon a successful scan, the staff application executes the `recordVerification` smart contract function, which permanently "burns" the unique cryptographic hash of the QR signature into the blockchain's `usedQRHashes` mapping. This ensures that any subsequent attempt to reuse the same QR code—even within the valid time window—is rejected as a duplicated payload.

6.3.2 Public Ledger Data Leakage

This attack assumes that an external adversary monitors the public Ethereum Sepolia block explorer (e.g., Etherscan) to extract PII, such as phone numbers or email addresses, from the smart contract state. The system mitigates this risk by enforcing zero-knowledge privacy principles on the public ledger. Prior to any blockchain transaction, the UNICESS's CryptoHelper module encrypts the entire PII payload off-chain using the AES-256-GCM. As a result, any data visible on the blockchain appears only as a randomized hexadecimal byte array (`encryptedData`), rendering the original personal information mathematically unreadable to public observers.

6.3.3 Unauthorized Device Use (Device Theft)

In this case, an unauthorized individual gains physical access to a student's unlocked mobile device and attempts to generate a fraudulent verification QR code. The system counters this by leveraging the Android OS Hardware Keystore. During registration, an asymmetric secp256r1 ECDSA private key is generated and securely stored within the device's hardware-backed secure enclave. The UNICESS itself cannot directly access this key for signing QR timestamps without explicit user authentication. Consequently, the QR generation process is halted until the user completes an un-bypassable biometric verification step, such as fingerprint or facial recognition, thereby ensuring that only the legitimate account owner can authorize the operation.

6.4 Project Challenges

Despite the successful implementation of the prototype, several limitations inherent to decentralized architectures must be acknowledged.

Firstly, the transaction confirmation time, which averages around 12 seconds on the Sepolia Testnet (and up to 6.1 seconds of user-facing latency), does not reflect the near-instant experience users expect from centralized Web2 applications. Achieving faster throughput on a production Mainnet would require migrating the smart contract to Layer 2 scaling solutions (such as Arbitrum or Optimism) to drastically reduce block propagation times and mitigate gas fee volatility.

Secondly, the current UNICESS prototype utilizes a symmetric encryption model (AES-256-GCM) to protect off-chain data. While this effectively obscures data on public block explorers, the symmetric master decryption key is currently embedded within the UNICESS's source code to facilitate the prototype's RBAC viewing mechanics. In a production enterprise environment, this poses a vulnerability to reverse engineering. A fully secure iteration must transition to an asymmetric Elliptic Curve Integrated Encryption Scheme (ECIES) or utilize a decentralized Key Management Network (e.g., Lit Protocol). This would ensure that student data is encrypted using the university's public key, guaranteeing that only the institution’s secure, private backend servers possess the corresponding private key required for decryption.

Finally, the system currently lacks a decentralized key recovery mechanism. If a student loses access to their MetaMask wallet mnemonic phrase, their digital identity and historical on-chain audit logs are irrecoverable without an off-chain administrative intervention. Furthermore, while the student identity records are decentralized, the supreme administrative role (DEFAULT_ADMIN_ROLE) is centralized to a single contract owner address. A more resilient, trustless model would mandate the use of a Decentralized Autonomous Organization (DAO) or a Multi-Signature (MultiSig) wallet, requiring consensus among multiple university administrators before critical system parameters can be altered.

6.5 Objectives Evaluation

The finalized UNICESS application and its underlying FRDDSID system architecture were rigorously evaluated against the primary objectives defined at the inception of the project:

1. To digitalize and modernize the student identity experience by delivering a user-friendly Android mobile application:

   The project delivered a unified, native Android application built with Material Design 3. The app seamlessly integrates a Web3 wallet (MetaMask) and replaces legacy physical cards. It successfully utilizes device-level biometrics (Android Keystore) to secure the user's private key, ensuring a modernized, highly secure user experience.

2. To implement a robust, dynamic QR code mechanism to combat forgery:

   The system successfully generates a time-sensitive QR code containing an on-chain timestamp, wallet address, and a secp256r1 ECDSA digital signature. The integration of a strict 300-second TTL expiration and a permanent on-chain hash-burning mechanism (recordVerification) provides absolute resistance against static forgery, screenshot duplication, and replay attacks.

3. To demonstrate a decentralized and transparent identity system founded on blockchain technology:

   A Solidity smart contract was successfully deployed on the Ethereum Sepolia Testnet. The integration of Web3j enabled the app to perform robust RBAC checks directly against the blockchain. The system further demonstrated transparency through a native in-app Blockchain Audit Explorer (utilizing the Etherscan V2 API), while simultaneously ensuring privacy by encrypting sensitive off-chain data via AES-256-GCM prior to blockchain submission.

6.6 Concluding Remark

In conclusion, the system evaluation confirms that the UNICESS prototype is a highly functional, secure, and responsive solution. The latency testing proved that read-operations are fast enough for seamless administrative use, while the asynchronous polling effectively manages the inherent delays of blockchain writing. Despite the acknowledged challenges of network speeds and key recovery, the system successfully fulfilled all defined objectives, demonstrating that a dynamic, cryptographically signed decentralized identity system is vastly superior to static, centralized digital IDs in preventing forgery.

CHAPTER 7

Conclusion and Recommendation

7.1 Conclusion

This project was motivated by the critical security and privacy deficiencies inherent in traditional and centralized digital student identity frameworks, which are highly susceptible to forgery and create single points of failure. The primary objective was to design, develop, and validate a modern identity solution that leverages the unique properties of blockchain technology to create a more secure, transparent, and user-centric ecosystem. To address this, the project proposed the FRDDSID system as an architectural prototype and successfully delivered its finalized realization: the UNICESS application, a functional end-to-end platform built on the Ethereum Sepolia Testnet featuring a secure Solidity smart contract.

The novel contribution of the UNICESS application is its unique synthesis of decentralization, advanced cryptography, and intuitive usability. Unlike legacy centralized models, the underlying FRDDSID architecture fundamentally avoids single points of failure by distributing state management across the blockchain while securing sensitive PII using AES-256-GCM off-chain encryption. By integrating device-level biometrics with on-chain timestamping and an automated hash-burning protocol, UNICESS implements a dynamic and forgery-resistant verification mechanism that directly neutralizes common forgery vectors such as screenshots and replay attacks.

Furthermore, the integration of OpenZeppelin RBAC, a native Etherscan-powered Audit Explorer, and a localized RAG AI Chatbot establishes a comprehensive, highly transparent institutional ecosystem. Ultimately, the UNICESS prototype successfully achieves its objectives and serves as a blueprint, proving that building a user-centric identity system that is more secure, transparent, and trustworthy than its centralized counterparts is not only possible but sets a potential new standard for the higher education sector.

7.2 Recommendation

Building upon the strong foundation of this prototype, several features for future work could significantly enhance the system's capabilities for a production-scale deployment.

A primary next step would be the integration of the InterPlanetary File System (IPFS) for decentralized image storage [28]. While the current system encrypts text-based PII, integrating IPFS would allow high-resolution student profile photos to be securely assigned with an immutable Content Identifier (CID) stored in the smart contract, completely removing the need for any centralized web servers.

Another critical enhancement involves transitioning from the current symmetric encryption model (AES-GCM) to a fully asymmetric, on-chain encryption scheme such as the ECIES or a decentralized Key Management Network like Lit Protocol. In a production environment, hardcoding a symmetric master key within the UNICESS poses a security risk if the application is decompiled. Utilizing an asymmetric model where student data is encrypted using the university's public key, ensuring that only the institution’s secure, private servers can decrypt the information would establish absolute, zero-knowledge privacy while retaining the immutable transparency of the public blockchain.

REFERENCES

[1] L. Scanlon, L. Rowling, and Z. Weber, "'You don't have like an identity … you are just lost in a crowd': Forming a Student Identity in the First-year Transition to University," *Journal of Youth Studies*, vol. 10, no. 2, pp. 223–241, May 2007, doi: https://doi.org/10.1080/13676260600983684.

[2] Zheng, "Implementing Student ID Cards: A Guide for Schools & Universities," RFID Card, Mar. 13, 2025. https://www.rfidcard.com/rfid-vs-traditional-student-id-cards-a-technical-breakdown-of-security-efficiency/ (accessed Apr. 25, 2025).

[3] "Campus ID Solutions: Transforming Higher Ed With Technology," Transact Campus, 2018. https://www.transactcampus.com/resources/blog/campus-id-solutions-in-depth-guide-to-transforming-higher-education-through-technology (accessed Apr. 25, 2025).

[4] R. Behrooznia, "What College Campuses Teach Us About Digital ID Adoption," Idtechwire.com, Jan. 29, 2025. https://idtechwire.com/what-college-campuses-teach-us-about-digital-id-adoption/ (accessed Apr. 25, 2025).

[5] P. Augstein , "Web Authentication Redirect," Bitly.com, Apr. 03, 2024. https://bitly.com/blog/how-cryptocurrency-software-uses-qr-codes/ (accessed Apr. 25, 2025).

[6] Nupura Ughade, "The types of Fraud in Education Sector & How to Prevent them?," hyperverge.co, Mar. 09, 2023. https://hyperverge.co/blog/fraud-in-education-sector/ (accessed Apr. 25, 2025).

[7] G. Marinoni, T. Jensen, T. Agasisti, and F. Bolzoni, "Digitalisation of Administrative Services in Universities: Findings from a Global Pilot Study," International Association of Universities and Politecnico di Milano, 2023.

[8] S. E. Haddouti and M. D. Ech-Cherif El Kettani, ""Analysis of Identity Management Systems Using Blockchain Technology," presented at the Rabat, Morocco, 2019 International Conference on Advanced Communication Technologies and Networking (CommNet), Apr. 2019, pp. 1–7.

REFERENCES

[9] S. Wynn-Jones, “Decentralised identity in higher education: Empowering students to take control of their digital identities,” *ProofID*, May 09, 2023. https://proofid.com/blog/decentralised-identity-in-higher-education-empowering-students-to-take-control-of-their-digital-identities/ (accessed Apr. 25, 2025).

[10] everycred, “EveryCRED,” EveryCRED, Dec. 16, 2024. https://everycred.com/blog/decentralized-identity-in-education/ (accessed Apr. 25, 2025).

[11] A. Goel and Y. Rahulamathavan, “A Comparative Survey of Centralised and Decentralised Identity Management Systems: Analysing Scalability, Security, and Feasibility,” Future Internet, Dec. 2024.

[12] R. Behrooznia, “What College Campuses Teach Us About Digital ID Adoption,” Idtechwire.com, Jan. 29, 2025. https://idtechwire.com/what-college-campuses-teach-us-about-digital-id-adoption/ (accessed Apr. 25, 2025).

[13] J. A. Inyangetoh and E. A. Johnson, “Development of QR Code-Based Authentication System for Admitting Students into Examination Hall for Polytechnics in Nigeria,” Mar. 2025, pp. 20–42.

[14] J. Cura and Extrimian, “Decentralized Identity in Education | Extrimian,” Extrimian | Securely verify any information in seconds, Nov. 29, 2024. https://extrimian.io/decentralized-identity-in-education/ (accessed Apr. 25, 2025).

[15] H. Prabowo and A. Bandur, “Digital transformation in higher education: Global trends and future research direction,” in Journal of Innovation in Business and Economics, Dec. 2021, pp. 103–118.

[16] “Biometrics Notice - Midy,” *Midy*, Jan. 06, 2025. https://www.midy.com/biometrics-notice/ (accessed Sep. 01, 2025).

[17] “Introducing Galxe Passport | Help Center - Galxe,” *Galxe.com*, 2023. https://help.galxe.com/en/articles/9424571-introducing-galxe-passport/ (accessed Sep. 01, 2025).

[18] “Galxe Identity Protocol,” *Galxe Docs*. https://docs.galxe.com/identity/introduction/ (accessed Sep. 01, 2025).

REFERENCES

[19] “SpruceKit Introduction | SpruceKit,” *Sprucekit.dev*, Jul. 11, 2025. https://www.sprucekit.dev (accessed Sep. 02, 2025).

[20] “Decentralized Identity Overview | SpruceKit,” *Sprucekit.dev*, Jul. 11, 2025. https://www.sprucekit.dev/sprucekit-introduction/decentralized-identity-overview (accessed Sep. 02, 2025).

[21] Latsan, “How Fractal ID, Fractal Wallet, and Fractal Protocol Interact,” *Medium*, May 06, 2022. https://latsan04.medium.com/in-depth-analysis-of-fractals-424cf5dacadc (accessed Sep. 02, 2025).

[22] Z. Abrams, “Fractal ID data breach traced to 2022 hack of employee who reused password,” *The Block*, Jul. 20, 2024. https://www.theblock.co/post/306504/fractal-id-data-breach-traced-to-2022-hack-of-employee-who-reused-password/ (accessed Sep. 03, 2025).

[23] admin, “Fractal ID Data Breach Post Mortem - Fractal ID - Web3 Identity Solution Provider,” *Fractal ID - Web3 Identity Solution Provider*, Jul. 19, 2024. https://web.fractal.id/fractal-id-data-breach-post-mortem/ (accessed Sep. 03, 2025).

[24] “walt.id - Web3 Wallets - Alchemy,” *Alchemy*, 2025. https://www.alchemy.com/dapps/walt-id (accessed Sep. 03, 2025).

[25] “Verifiable Credentials (W3C),” *Walt.id*, 2018. https://docs.walt.id/community-stack/concepts/digital-credentials/verifiable-credentials-w3c (accessed Sep. 03, 2025).

[26] J. Doe, "The Future of Decentralized Identities," Web3 Insights Today, Nov. 06, 2024. [Online]. Available: https://www.example.com/future-of-did/ (accessed Sep. 03, 2025).

[27] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, May 1988, doi: 10.1109/2.59.

[28] “How IPFS works | IPFS Docs,” *Ipfs.tech*, 2025. https://docs.ipfs.tech/concepts/how-ipfs-works/#how-ipfs-transfers-data (accessed Sep. 03, 2025).

APPENDIX

Ethereum Smart Contract Code Sample

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v5.0.2/contracts/access/AccessControl.sol";

contract FRDDSID is AccessControl {
    string public constant schoolName = "Universiti Tunku
Abdul Rahman";
    bytes32 public constant STAFF_ROLE =
keccak256("STAFF_ROLE");

    struct Student {
        string studentId;
        string name;
        bytes encryptedData;
        bytes publicKey;
        bool isValid;
        address ownerAddress;
        address delegateKey;
        uint256 lastGenerationTimestamp;
        uint256 verificationCount;
    }

    struct VerificationLog {
        address verifier;
        uint256 timestamp;
        bool isStaff;
    }

    mapping(address => Student) public identities;
    mapping(string => address) public studentIdToAddress;
    mapping(bytes32 => bool) public usedQRHashes;
    mapping(address => VerificationLog) public
lastVerification;
    mapping(address => address) public delegateToOwner;

    address[] public allRegisteredAddresses;

    event StudentRegistered(address indexed studentAddress,
string studentId);
```

```
    event DelegateLinked(address indexed owner, address
indexed delegateKey);
    event TimestampUpdated(address indexed studentAddress,
uint256 timestamp);

    constructor() {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    }

    modifier isOwnerOrDelegate(address _targetWallet) {
        address owner = delegateToOwner[msg.sender] !=
address(0) ? delegateToOwner[msg.sender] : msg.sender;
        require(owner == _targetWallet || msg.sender ==
identities[_targetWallet].delegateKey, "Not authorized.");
        _;
    }

    function assignStaff(address _staffAddress) public
onlyRole(DEFAULT_ADMIN_ROLE) {
        grantRole(STAFF_ROLE, _staffAddress);
    }

    function revokeStaff(address _staffAddress) public
onlyRole(DEFAULT_ADMIN_ROLE) {
        revokeRole(STAFF_ROLE, _staffAddress);
    }

    // Allows background Delegate Keys to inherit their
Owner's Staff Role
    function checkIsStaff(address _account) public view
returns (bool) {
        address owner = delegateToOwner[_account] !=
address(0) ? delegateToOwner[_account] : _account;
        return hasRole(STAFF_ROLE, owner) ||
hasRole(DEFAULT_ADMIN_ROLE, owner);
    }

    // Staff must call this once via MetaMask to link their
Android background key
    function linkDelegate(address _delegateKey) public {
        delegateToOwner[_delegateKey] = msg.sender;
        emit DelegateLinked(msg.sender, _delegateKey);
    }

    function registerStudent(
        string memory _studentId,
        string memory _name,
        bytes memory _encryptedData,
        bytes memory _publicKey,
        address _delegateKey
```

```
    ) public {
        require(identities[msg.sender].ownerAddress ==
address(0), "Already registered.");
        require(studentIdToAddress[_studentId] == address(0),
"ID in use.");

        identities[msg.sender] = Student({
            studentId: _studentId,
            name: _name,
            encryptedData: _encryptedData,
            publicKey: _publicKey,
            isValid: false,
            ownerAddress: msg.sender,
            delegateKey: _delegateKey,
            lastGenerationTimestamp: block.timestamp,
            verificationCount: 0
        });

        studentIdToAddress[_studentId] = msg.sender;
        delegateToOwner[_delegateKey] = msg.sender;
        allRegisteredAddresses.push(msg.sender);

        emit StudentRegistered(msg.sender, _studentId);
    }

    function updateMyInfo(address _targetWallet, bytes memory
_newEncryptedData) public isOwnerOrDelegate(_targetWallet) {
        identities[_targetWallet].encryptedData =
_newEncryptedData;
        identities[_targetWallet].isValid = false;
    }

    function updateTimestamp(address _targetWallet) public
isOwnerOrDelegate(_targetWallet) returns (uint256) {
        uint256 newTimestamp = block.timestamp;
        identities[_targetWallet].lastGenerationTimestamp =
newTimestamp;
        emit TimestampUpdated(_targetWallet, newTimestamp);
        return newTimestamp;
    }

    function recordVerification(address _studentWallet,
bytes32 _qrSignatureHash) public {
        require(identities[_studentWallet].ownerAddress !=
address(0), "Student does not exist.");
        require(!usedQRHashes[_qrSignatureHash], "QR already
used.");

        usedQRHashes[_qrSignatureHash] = true;
        bool isVerifierStaff = checkIsStaff(msg.sender);
```

```
        lastVerification[_studentWallet] = VerificationLog({
            verifier: msg.sender,
            timestamp: block.timestamp,
            isStaff: isVerifierStaff
        });

        identities[_studentWallet].verificationCount += 1;
    }

    function setStudentValidity(address _studentWallet, bool
_isValid) public {
        require(checkIsStaff(msg.sender), "Not authorized
Staff.");
        identities[_studentWallet].isValid = _isValid;
    }

    function deleteStudent(address _studentWallet) public {
        require(checkIsStaff(msg.sender), "Not authorized
Staff.");
        string memory id =
identities[_studentWallet].studentId;
        delete identities[_studentWallet];
        delete studentIdToAddress[id];
        // Note: For array iteration, we leave the address in
allRegisteredAddresses.
        // Our Android app will filter out addresses where
ownerAddress == address(0).
    }

    function getInfoByWalletAddress(address _walletAddress)
public view returns (Student memory) {
        return identities[_walletAddress];
    }

    function getAllRegisteredStudents() public view returns
(address[] memory) {
        return allRegisteredAddresses;
    }
}
```

POSTER


FORGERY RESISTANT
UNICESS
DECENTRALIZED DIGITAL STUDENT ID
Traditional Student IDs Are Insecure, Centralized and Easily Forged. Decentralized ID On The Blockchain Addresses These Critical Security Flaws.
BLOCKCHAIN MANAGEMENT
REFRESH BLOCKCHAIN DATA
Generated QR Code
Changed Account Status
Account Registered
BLOCKCHAIN ACCESS ID
Valid QR Code, Expires in 30s
Justin Bieber
2202001
ACTIVE
VIEW MORE DETAILS
- Dynamic Forgery-Resistant QR
Time-sensitive, biometrically signed QR codes to eliminate risks
- Instant On-Chain Verification
Rapidly authenticates credentials by validating cryptographic signatures and timestamps on the blockchain
- Secure Admin Management
Utilizes role-based access control (RBAC) for staff to easily activate, revoke and audit student identities
- Localized AI Assistan
Features a built-in, private RAG chatbot to answer user queries regarding university policies
Secure, User-friendly dApp for Student ID
Dynamic QR Code to Prevent Forgery
Transparent ID System powered by Solidity
USER'S DEVICE
EXTERNAL WEB3 SERVICES
DECENTRALIZED NETWORK
UNICESS
MetaMask SDK
Infura RPC
Ethereum Blockchain
FRDDSID System Smart Contract
Transaction Request
Signed Transaction
Read/Write Call
sepolia
remix
solidity
Contains
Read-Only Call
UNICESS Demonstrates That Decentralized Digital IDs Are A Viable Alternative To Traditional Systems.
It Offers A Blueprint For User-Centric, Forgery-Resistant Identity Solutions On The Blockchain.
2026 Chen Jin Shen supervised by Ts Dr Gan Ming Lee
UTAR