UNIVERSITY OF THE PHILIPPINES
OPEN UNIVERSITY

MASTER OF INFORMATION SYSTEMS

PAOLO F. ESCOTIDO

Verifiable Academic Records for ISM (VERACISM) : A Permanent, Decentralized, Blockchain-Backed Storage for Immutable Reports

Adviser:

KATRINA JOY M. ABRIOL-SANTOS
Faculty of Information and Communication Studies

06 MAY 2026

Permission of the classification of this academic work access is subject to the provisions of applicable laws, the provisions of the UP IPR policy and any contractual obligations:

Invention (I)	YES/NO
Publication (P)	YES/NO
Confidential (C)	YES/NO
Free (F)	YES/NO

Student's signature:

Adviser's signature:

University Permission Page

Verifiable Academic Records for ISM (VERACISM) : A Permanent, Decentralized, Blockchain-Backed Storage for Immutable Reports

I hereby grant the University of the Philippines a non-exclusive, worldwide, royalty-free license to reproduce, publish and publicly distribute copies of this Academic Work in whatever form subject to the provisions of applicable laws, the provisions of the UP IPR policy and any contractual obligations, as well as more specific permission marking on the Verifiable Academic Records for ISM (VERACISM).

I specifically allow the University to:

1. Upload a copy of the work in the theses database of the college/school/institute/ department and in any other databases available on the public internet
2. Publish the work in the college/school/institute/department journal, both in print and electronic or digital format and online; and
3. Give open access to the work, thus allowing “fair use” of the work in accordance with the provision of the Intellectual Property Code of the Philippines (Republic Act No. 8293), especially for teaching, scholarly and research purposes.

PAOLO F. ESCOTIDO/ Date
Signature over student name and date

Approval Page

This paper prepared by PAOLO F. ESCOTIDO with the title: ”Verifiable Academic Records for ISM (VERACISM) : A Permanent, Decentralized, Blockchain-Backed Storage for Immutable Reports” is hereby accepted by the Faculty of Information and Communication Studies, U.P. Open University, in partial fulfillment of the requirements for the degree MASTER OF INFORMATION SYSTEMS.

KATRINA JOY M. ABRIOL-SANTOS
Adviser — Date Signed

DR. RIA MAE H. BORROMEO
MIS Program Chair — Date Signed

DR. ROBERTO B. FIGUEROA, JR.
Dean
Faculty of Information and Communication Studies

Date Signed

ACKNOWLEDGEMENT

The researcher wishes to convey profound gratitude to all individuals and offices who played a role in bringing this capstone project to fruition, entitled "**VERACISM**: *Verifiable Academic Records for International School Manila*."

Heartfelt thanks go first to my project adviser, **KATRINA JOY M. ABRIOL-SANTOS**, for her constant guidance, timely feedback and patience in the process of writing this proposal and designing this study. Her feedback and suggestions to the project on scope, structure and technical aspects were pivotal in developing the system concept, and the manuscript.

The researcher also extends his appreciation to each member of the panel for the questions that they raised and the tips that are provided to strengthen the problem statement and to make the methodology more valid and in line with the needs of the institution while maintaining the technical validity of VERACISM.

The school's appreciation also goes to **International School Manila** for sharing their current practices and issues with report authenticity, verification and long-term maintenance of reports. They helped to develop the needs of the project, use cases and the whole VERACISM workflow.

The researcher also acknowledges the technical advice, detailed review and constant support provided by the **IT Development Team** during the project development.

Last but not least the researcher would like to thank his family members and friends for their encouragement, patience and understanding during the countless hours he spent researching, writing and developing a system. These constant sources of encouragement helped us to persevere and complete this work.

More than anything else, the researcher expresses gratitude to **God** for his grace, power and perseverance in the completion of this work. **SALAMAT SA DIOS !**

ABSTRACT

International School Manila (ISM) has extensive collection of Academic and Administrative Reports that are crucial for accreditation, audits and external verification, and the authenticity and integrity of these reports is critical. Currently, no cryptographic bindings exist between the reports and their metadata in centralized storage systems, which can be modified without being detected, and which have no IT-independent audit trails or slow verification.

This project suggests a *hybrid platform* called **VERACISM** (*Verifiable Academic Records for ISM*) that will allow to attach finalized reports to decentralized content-addressed storage while maintaining identity, access control, and operational monitoring within the governance of ISM. Reports are created with the existing ISM applications are uploaded via api integreation. In VERACISM,scanned files were validated using ClamAV and YARA, and by MIME type and file size, then hashed with SHA-256, and written to IPFS with Filecoin-backed deals, using Lighthouse. The file is given a unique Content Identifier (CID) and is stored with report metadata and immutable audit events in MSSQL under strict RBAC/ABAC rules.

Both a web-based console and a public verification portal enable tenant and integrator administration and provide both internal and external parties with the ability to verify report integrity via CIDs or QR codes, with an optional option of cross-checking storage evidence in public explorers.

The project is implemented using a DevSecOps lifecycle approach which includes secure design, automated checks of security and scheduled VAPT, following the OWASP Top 10 risks. Planned evaluation integrates task-based user testing for administrator, tenant offices, integrators and verifiers, structured security testing and defined acceptance criteria.

VERACISM will help enhance the record governance of ISM, lower verification overhead, and offer a pattern for storing academic records in a tamper-proof fashion.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Background of the Project

International School Manila (ISM) handled a wide range of academic and administrative reports that had to remain authentic, traceable, and tamper-proof. These records provided accountability, accreditation and academic credibility. Under ISM's standard process, reports were generated using various applications and uploaded directly to on-premise servers or cloud storage via various APIs that each created one student record. This process had no immutability version tracking nor an audit trail. Files or metadata could be modified or replaced without a durable record of who changed what and when. Since there was no cryptographic binding between a stored file and its database record, undetected alterations were possible and unverifiable. When reports were shared or reviewed, external parties such as colleges and universities had no independent way to prove that the files matched the originals released by ISM. This required multiple email exchanges between ISM and the external organization. The absence of tamper-evidence and end-to-end traceability weakened ISM's data governance and institutional trust.

The volume of existing records gave this risk concrete institutional consequences. Between 2014 and 2025, ISM's student report repositories, which covered report cards, CAS reports, Measures of Academic Progress results, and related documents, already contained **112,612 files**, with a further **16,108 files** generated by Finance. As of 2025, ISM had **2,445** active students, each of whom accumulated dozens of academic and financial records even before alumni and long-term archives were counted. In such an environment, just a minor inauthenticity, provenance or version control issue could impact tens of thousands of files simultaneously.

Given these circumstances, those using the reports inside ISM had no easy way to trace the history of a file over time, and no way to witness the tampering of a report. They would have had to rely on the assumption that what in front of them was the "correct" version that the institution had intended because they didn't check a cryptographic record

that proved the report was unchanged. This dependence on institutional assurances, as opposed to mathematically proven proof, over time had proven to be an operational and compliance risk (Cflow, 2025; Encrypthos, 2025; National Research Council, 2005; Simple But Needed, Inc., 2025). Therefore, there was uncertainty when the audit was performed and when the accreditation review was performed and when the interdepartmental review was performed. The process was slow, manual and IT-dependent, particularly as there was a possibility of multiple copies of the same file existing in different systems.

In various industries, such as universities, hospitals, financial institutions, and others, teams implemented various partial solutions, such as cloud storage versioning, checksums, or IPFS pinning services, to enhance the recovery and tracking of files. IPFS (InterPlanetary File System) was a decentralized, peer-to-peer protocol used for data storage and sharing via content-addressed identifiers, where every file was identified by a cryptographic hash instead of a server address.(Crypto.st, 2025; Protocol Labs, 2025). These tools improved the resilience, but typically were just standalone utilities, running away from the day to day activities of academic offices. Their guarantees were still reliant on the logs within their local systems and ecosystem trust, rather than on an end-to-end, independently verifiable trail.(Amazon Web Services, 2025; BlockchainReporter, 2025; Filebase, 2025; National Institute of Standards and Technology, 2023; Pinata, 2025; Schneider, 2025).

To fill this gap, **VERACISM** (Verifiable Academic Records for ISM) introduced content-addressed, decentralized persistence for reports storage and verification. ClamAV and YARA were used to scan each uploaded report, which was validated by MIME type and size, then hashed using SHA-256, and stored on the Lighthouse system on IPFS with deals backed by Filecoin. A different Content Identifier (CID) is generated for each file the moment there is a change in content, thus ensuring tamper resistance and publicly verifiable. A secure upload for both .NET 8 Web API (MSSQL) and tenant-aware metadata and immutable audit events were watched, and they had strict RBAC/ABAC. Verification took place at the portal level by QR or CID lookups which re-hashed content and compared it with evidence of deals on Filfox without the need of ISM credentials. The baseline integrity, permanence and independent verification was achieved by fully

content-addressed, decentralized storage. If additional needs came along, such as rules for access, payment or cross-application automation on the blockchain, it was reserved for future versions.

VERACISM adopted an architecture that was a mix of on-premise control and decentralized permanence. ISM assumed control of identity, access restrictions, metadata and operational audit trails in its trusted space, while content integrity and long-term durability were backed up by public decentralized infrastructure, such as IPFS/Filecoin, which can be independently verified. This model enabled full flexibility of existing workflows and compliance controls and made the proof of integrity easily verifiable, without depending on the trustworthiness of ISM systems.

The goal of VERACISM was to break the cycle of implicit trust for explicit proof, long term and visible record keeping, and easier audits. The proposed system sought to enhance the credibility of institutions, minimize the cost of verification checks, and to overcome report modification by silence or by unauthorized third parties by applying cryptographic integrity, decentralized data storage, and an outer layer of verification.

Objectives

The proposed system was designed to modernize and secure creation, storage and independent verification of ISM academic reports with the help of content-addressed decentralized infrastructure, namely IPFS/Filecoin with the help of Lighthouse. The main goal was to eliminate hacks, and simplify verification with CID , QR-based and File upload checks, which would give cryptographic, tamper-evident proof of integrity, without the need of smart contracts.

Specific Objectives

1. Implement file security check and validation of file type.
2. Designed and developed web application which used role-based access control.

3. Defined a threat model and mitigations such as input sanitisation, rate limits, CSRF, key rotation and least privilege.

4. Clearly and effectively designed, developed and applied integration, security and performance testing with clear success criteria – including correct end-to-end handling of reports across integrated systems, no critical/High severity security findings, response times and throughput were maintained within agreed service levels under agreed load etc.

5. Deployed, maintained and provided runbooks for backup and restore, monitoring, incidents, and compliance.

Scope and Limitations

The system maintained immutable storage, verification and controlled retrieval of finalized reports, complete audit trail and cryptographic proof of integrity. It did not create any report content, or supplant academic systems. These were some of the project boundaries, goals and benefits summarized:

Scope

- Verification Portal
  - Made CID lookup and file upload capabilities in the portal available for independent validation, without the need for ISM credentials.
- Document Security Feature
  - Included pre-upload security controls, such as ClamAV, YARA rules, MIME validation, size validation and quarantine workflow.
  - Content-addressed storage through Lighthouse with IPFS/Filecoin.
  - Computed SHA-256 hashing and CID revalidation during verification.
- Events/Access Management

  - Provided REST APIs for upload, metadata retrieval, verification, and controlled download.
  - Stored tenant-aware metadata and immutable audit events in MSSQL.
  - Implemented role-based and attribute-based access control with scoped API keys.
  - Performed scheduled health checks for CID availability, replica status, and retrieval latency.
  - Provided structured logging, alerting, and runbooks for operations across Dev, UAT, and Production.

- Integration
  - Integrated API endpoints with existing ISM applications through approved service accounts.

VERACISM was integrated with completed institutional reports, guaranteeing long term claims that are integrity and traceable, while also being auditable, and without the need of the original party to ensure the integrity of the report. VERACISM is equivalent to secure intake/scanning/hashing, and decentralized storage and metadata registration/digestion/verification of documents per registered hashes (with whitelisted ISM workflows). Instead it was designed to be integrated with existing academic administration systems that could be used to create and/or manage student information, but were not designed to be a replacement to the systems used by students to create and/or manage student information. VERACISM did the research on the aforementioned threats and reported on the following: Tamper evidence, auditability, controlled access and continued compatibility with ISM existing systems.

Limitations

- Report authoring or workflow approvals.
- Replacement of SIS or LMS and non-records business functions.

- Smart contracts implementation and on-chain access rules, or payment logic.
- Student self-service uploads.
- Semantic content moderation beyond malware and known bad pattern checks.
- User crypto wallet management or token handling.
- Automated retry mechanism for transient Lighthouse storage failures; a document that passed scanning might not have been anchored to IPFS if the Lighthouse service was temporarily unavailable.
- The audit log UI did not enforce tenant-level scoping; audit metadata visibility was limited to the Admin role.
- The public verification endpoint was tested with 100 virtual users and a more comprehensive load test of the entirety of the scan-upload pipeline was beyond the scope of the project.
- During UAT testing, redirect links were not found on Administrator dashboard / Tenant Dashboard cards, which was in scope for the next development phase.

VERACISM is not designed for report generation/workflows, report grading workflows, Self Service Portal uploads or Enterprise document management solutions. Other improvements include more advanced load testing, automatic retries and more detailed drilling within the dashboard. These options were removed, so that the initial release could concentrate on the main challenge: to provide signed off records to be verified, immutable and auditable.

REVIEW OF EXISTING ALTERNATIVES

Staff from the International School Manila and outside reviewers currently depend on centralized attestation, registrar portals, and provider immutability for skew authenticity.The current International School Manila's staff and outside evaluators rely on centralized attestation, registrar portals, and provider immutability as proof of skew authenticity. Access logs and versioning are implemented for files.In certain higher education institutions (HEIs) in the Philippines, Formal checks are done via CHED eCAV that standardizes the requests for the checks in a platform-bound and location-addressed manner. Verifiers should only believe the metadata from the portal and should not try to validate a content bound proof from the bytes of the file (Commission on Higher Education, 2025). Pilots, such as DPWH Integrity Chain and GovChain, show national enthusiasm for tamper evident public records but only address public sector records and cannot oversee the range of academic reports, each with its own separate and independent third party authentication of the notoriously fraught report-by-byte level. (Better Government Philippines, 2025; Department of Public Works and Highways, 2025).

Education is based on verifiable credentials as the strongest resources. EduCredPH forwarded a concept of a permissioned blockchain network for educational credential verification in the Philippine context where HED and third party verifiers can issue credentials and verified on a common Hyperledger Fabric network, all while learners control and consult their credentials (Sy, Marasigan, & Festijo, 2024).On OpenCerts, this is extended across institutions by each institution minting a signed payload which can be signed by any verifier without communication with the issuer. The technical annex of the SkillsFuture Transformation provides an understanding of the data model and the verification flow in easy-to-understand operational steps and the work with the ones NTU has demonstrated during the institution's adoption of the SkillsFuture Transformation is illustrated in a daily fashion (Government Technology Agency of Singapore, 2019; Nanyang Technological University, 2025; SkillsFuture Singapore, 2018). End to end issuance, revocation, and verifier experience with research and university application

demonstrates feasibility in ASEAN and even worldwide areas. All such efforts continue to be credential-centric and fail to offer content addressed permanence for unstructured academic reports on scale.(Mohebi & Aghdam, 2024; Naji, Jawad, Hafiz, Al Obaidi, & Abd, 2023).

VERACISM is unique in taking the credential verification ideology to each completed report. When ingesting a file is scanned and validated, and then hashed to retrieve its content identifier (SHA256) which differs if any of the file's bytes are altered. Files are stored on decentralized infrastructure for which deal evidence can be seen using public explorers for durability, and for public corroboration. Auditors, educators and students can rehash a copied file and compare the rehashed identifier to the recorded identifier, without any ISM credentials, and confirm storage evidence. This design is putting cryptographic proofs under the current portals. eCAV can continue to be a national front door for policy and records processing, while VERACISM provides content addressed evidence and independent verification for any report artifact. Local solutions like GovChain and Integrity Chain prove that this is feasible on public records that need to be tamper evident. VERACISM extends this concept to the academic setting by supporting role based access, tenant aware metadata, and immutable audit events. In the end, VERACISM extends beyond the concept of focusing on credentials or centralizing trust; it's the tangible and independently verifiable evidence offered in every report.(Better Government Philippines, 2025; Commission on Higher Education, 2025; Department of Public Works and Highways, 2025; Government Technology Agency of Singapore, 2019; Mohebi & Aghdam, 2024; Naji et al., 2023; Nanyang Technological University, 2025; SkillsFuture Singapore, 2018; Sy et al., 2024).

System (License/Type)	Platform	Brief Description	Basic Functionalities	Relation to the Proposed System
VERACISM	.NET 8 API, MSSQL, IPFS, Filecoin	Content addressed storage and verification for ISM reports.	Scan and validate files, hash with SHA 256, store on decentralized storage, log actions, enforce role based access.	–
CHED eCAV (Government portal)	Web portal	National portal for electronic verification of academic records.	Receive verification requests, check records, issue electronic certifications.	External authority; VERACISM can add file level proof for ISM records used in eCAV.
DPWH Integrity Chain and GovChain (Government pilots)	Public blockchain platforms	Pilots for transparent and tamper evident government project data.	Record project data on a blockchain and expose entries for public viewing and checking.	Demonstrate feasibility of tamper evident records but focus on government datasets, not ISM reports.
EduCredPH (Philippine credential network)	Permissioned blockchain (Hyperledger Fabric)	Credential network for Philippine higher education institutions.	Issue, store, and verify academic credentials across participating institutions.	Addresses credential fraud and multi institution verification, but not unstructured ISM reports or malware scanning.
OpenCerts (Singapore) (National system)	Public verification system	Singapore platform for digital education and training certificates.	Issue structured certificate files and support public verification without contacting issuers.	Mature credential model but tied to its ecosystem and not designed as general ISM report storage.
Research systems: UTM BADVES and DIAR (Research)	University prototypes	Blockchain based academic credential issuance and checking.	Model issuing, revoking, and verifying awards using distributed ledger technology.	Provide design patterns and feasibility evidence but are prototypes, not ready for ISM production with diverse reports.

Table 1. Summary of Existing Alternatives

System	Content-addressed storage for all reports	Decentralized storage for reports	Malware scanning at ingest	Credential-focused model	Government / public - records focus	Internal ISM storage and checking layer
VERACISM	Yes	Yes	Yes	No	No	Yes
CHED eCAV	No	No	No	Yes	No	No
DPWH Integrity Chain / GovChain	No	Yes	No	No	Yes	No
EduCredPH	No	No	No	Yes	No	No
OpenCerts (Singapore)	No	No	No	Yes	No	No
UTM BADVES / DIAR	No	No	No	Yes	No	No

Table 2. Feature Mapping of Existing Alternatives

Summary of Existing Alternatives

The systems in Table 1 each solve a part of the trust and verification problem, but they are designed for their own scope. Together they strengthen credentials and public sector records, yet they still leave ISM without a single, content based pipeline for all finalized reports under local control.

CHED eCAV is a national gateway to the verification of academic records, however, it only receives requests and provides responses and does not access the files and does not encroach on the protection provided by ISM on how it stores them. DPWH Integrity Chain and GovChain demonstrate that the government project-related information can be placed on a blockchain, but it is configured to student reports. EduCredPH provides higher education schools with a common means of issuing and verifying credentials, whereas OpenCerts provides Singapore with the same but nationwide, both remain limited to formal certificates. UTM BADVES and DIAR university prototypes investigate the way a university can produce and revoke credentials using distributed ledger technology, but these are not yet products, only research projects. These systems combined aid in external verification, credential fraud, but each of them cannot serve as the internal storage and checking layer of ISM to all report files, and thus cannot substitute VERACISM.

The biggest gap in the existing systems is that they look either at credentials alone or at government project data, so none of them gives ISM lasting protection for every report together with malware checks and audit trails that follow ISM roles and students. VERACISM is worth building because it pulls those missing pieces into one flow that staff already use. When a report is uploaded, the system first runs antivirus and YARA scans, then checks the file type and size, computes a hash, and saves the content on decentralized storage while writing every step into an audit log that cannot be changed. It treats each finalized report much like a diploma, giving it a content identifier that is tied to ISM metadata and access rules. Afterwards any authorized person can retrieve the file, recalculate the hash, and compare the result with the one stored to assure that the report hasn't been changed and is still available in storage. This is a change to the habit of trust around ISM records and makes it visible and verifiable.

PROJECT DETAILS

The scope of the proposed project VERACISM as well as its operational workflow was explained in detail starting with its basic design up to its operational workflow. It explained to them how the ISM systems are integrated with this platform, how the reports are processed, received to the platform and verified there and how stakeholders can access to these capabilities. It also highlighted some of the characteristics of security, integrity, or governance that differentiate VERACISM from existing capabilities.

Overview

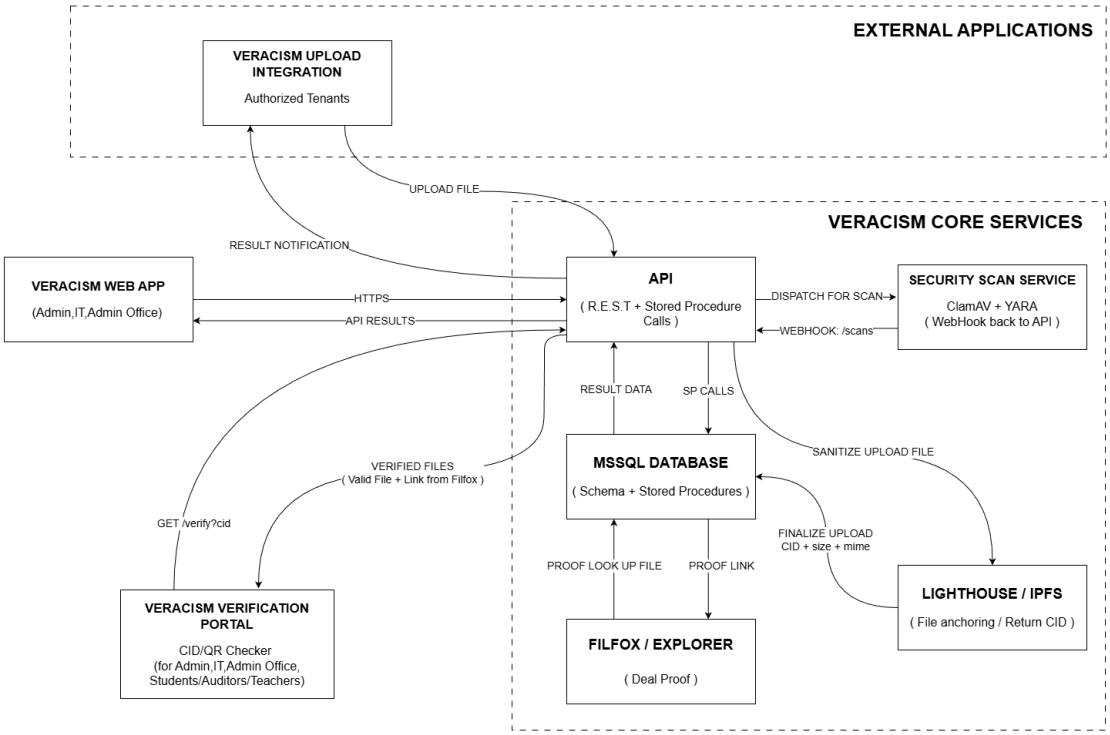

Figure 1. System Structure

VERACISM was a storage and verification service for finalized academic and administrative reports of International School Manila. The existing ISM systems prepared and submitted the reports via a secure .NET 8 Web API. All files have been scanned for

malware, validated, hashes with SHA-256 and stored on IPFS with Filecoin-backed deals using Lighthouse. Each report was also uniquely identified using a Content Identifier (CID) and a web accessible verification portal was provided for users to freely verify the integrity and authenticity of a report with no reliance on the ISM provided storage.

Project Framework

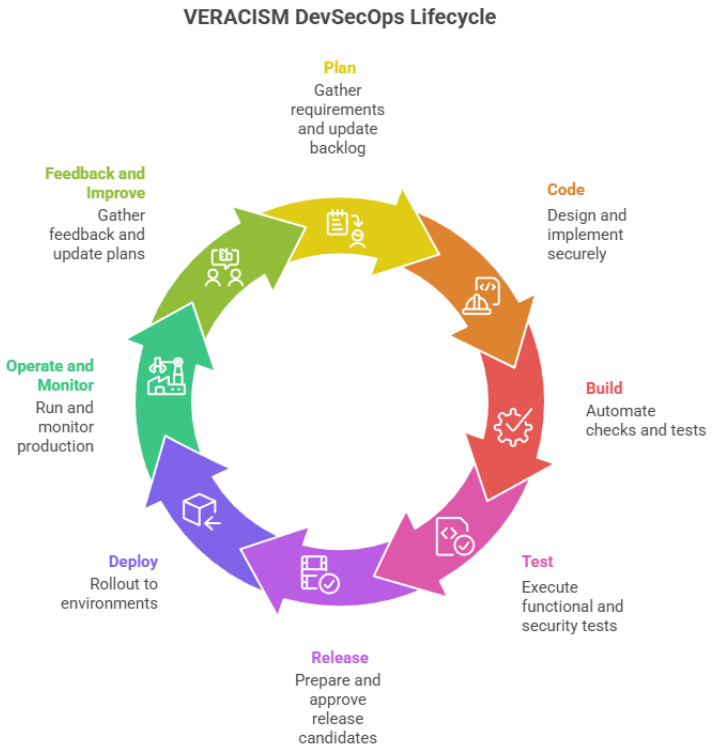

Figure 2. DevSecOps Lifecycle

The VERACISM project is based on the DevSecOps lifecycle used by the ISM development team. Security is integrated in each step, and it is repeated indefinitely. When work loops are identified for problems (functional problems, NFR gaps or security findings), the planning and requirements phase rather than a one-off activity.

The main DevSecOps phases for VERACISM are:

1. **Plan (Information gathering and requirements)**

   Gather business requirements, technical constraints, and security requirements from ISM stakeholders. Add new features, risks and compliance requirements to the backlog.

2. **Code (Secure design and implementation)**

   Produce and employ features that are secure: Decentralized, immutable audit

logging in MSSQL, hybrid architecture, RBAC/ABAC. For long-term storage, and to ensure verifiability, the data is stored on IPFS/Filecoin and apply secure coding practices.

3. **Build (Automated checks)**

   The .NET and front-end was compiled and package using CI/CD pipelines. Run automated unit tests, static analysis, dependency checks (such as Snyk, ISM), etc. Basic security check for each build (equivalent).

4. **Test (Functional, NFR, and security)**

   Perform functional tests, non-functional tests (performance, reliability, usability) and security testing (VAPT) based on ISM VAPT Guidelines. Test specifically for common OWASP Top 10 risks such as broken access control, injection, authentication failures, and logging/monitoring gaps.

5. **Release (Approval and hardening)**

   Build release candidates, with documented changes, risk assessment and security evaluation results. Make sure that critical or high findings are dealt with or formally assesses and confirms a student's risk level per ISM policy before promoting.

6. **Deploy (Controlled rollout)**

   Automated and repeatable pipelines deploy VERACISM to Dev, UAT and Production. Implement hardened configuration, secret control and application of environment-specific settings according to ISM operational standards.

7. **Operate and Monitor**

   Use VERACISM for production, with immutable audit trails, structured logging and health monitoring. Monitor and identify anomalies, performance problems and security events through dashboards and alerts.

8. **Feedback and Improve**

   Feedback from incidents and test failures, monitoring alerts, and user feedback are returned to the Plan phase. Requirements, designs and controls are updated and

the next DevSecOps cycle begins. This is a loop that is repeated all of the life time of the VERACISM platform.

Materials/Technologies Used

The following key materials are required for the implementation of VERACISM: technologies:

- **Application Backend:** `.NET 8` Web API deployed on ISM Windows servers.
- **Database Layer:** MSSQL 2019 or later for metadata and immutable audit records.
- **Decentralized Storage:** Lighthouse/IPFS with Filecoin-backed deals and Filfox verification.
- **Security Tooling:** ClamAV/YARA scanning, MIME and size validation, and SHA-256 hashing.
- **Front-end Design:** Angular and TypeScript SPA for the web pages and public portal.
- **Access Control and Identity:** ISM identity integration with RBAC and ABAC.
- **Monitoring and Operations:** ISM logging, health checks, CID checks, and alerts.
- **Verification Aids:** QR code and CID lookup for report authenticity checks.

Component	Main Technologies	Role in VERACISM
Backend and database	`.NET 8` Web API, MSSQL 2019+	Coordinate uploads, apply policies, and store metadata and audit events.
Decentralized storage layer	Lighthouse, IPFS, Filecoin, public gateways, Filfox	Provide content-addressed, durable storage and public evidence of persistence.
Security and integrity tools	ClamAV, YARA, MIME/size validation, SHA-256 hashing	Block malicious files and generate cryptographic proofs of file integrity.
User-facing interfaces	Angular/React SPA, QR codes, CID lookup tools	Expose internal console and public verification portal for staff and external verifiers.
Identity, access, and operations	ISM identity systems, RBAC/ABAC, existing monitoring and logging stack	Control who can act on reports and keep the system observable and supportable.

Table 3. Outline of materials and technologies used in VERACISM Application

In Table 3, VERACISM used an enterprise stack with .NET 8, MSSQL, ISM-managed servers, Lighthouse/IPFS/Filecoin storage, and security tools such as ClamAV, YARA, and SHA-256 hashing. Its front end, identity integration, and monitoring supported external verification within ISM infrastructure.

System Design

The functional requirements, along with the non-functional requirements of VERACISM, formed the basis for designing it. It also employed standard design models like use case diagrams, sequence diagrams, data models, and deployment diagrams. This section described briefly the key system features and data structure.

Functional Requirements. On the high level level the functional requirements were:

- **Upload of report file/encryption:** Report files (PDF, DOC, DOCX) with its metadata were uploaded to the system via a secure API. If the upload was valid and authenticated, it would be temporarily stored using a tracking ID temporarily. Uploads were not accepted if they were invalid, exceeded the size limit, or were not authorised (giving good error messages).
- All uploaded files were scanned and validated for malware, file type and file sizes prior to being placed in permanent storage. The following files could not be scanned, or were found to be invalid, and were not saved.
- **Hashed and generated CID:** A Content Identifier (CID) was generated and stored for every accepted file. A SHA-256 hash was also computed for each file. The hash and CID were used as the source of truth, and would later be used for verification.
- **Decentralized storage:** Removed accepted report files and stored them on IPFS via Filecoin-backed deals from the system. Also saved CID and storage information.
- **Storage of metadata and immutable audit trail:** Metadata was stored in MSSQL along with an audit path that was tenant-aware. The metadata stored in the report was an identifier of the source report, student identifier, term, academic level, file hash, CID, upload time and producing system. When auditing events recorded who or what was involved, what action has taken place, when and what the outcome.
- Access to administrative functions and report retrieval were according to role-based and attribute-based access control: RBAC and ABAC rules. Access and operation

were only allowed for authorised staff and service accounts.

- **Public Verification Portal and CID/QR verification:** There was a public verifier portal and CID/QR verifier in the system. The users could enter a CID, upload a file, or read a QR code. Then the service confirmed if the report was present or not and showed the hash of that report and if the stored file is equal to the original.
- **Student search via both internal console and report:** Authorized ISM staff could search reports on students, term, report identifier, and other metadata. They could also use their internal console to view reports and audit history.
- **Health monitoring & storage checks:** Events were recorded and ISM tools for monitoring raised alerts when the thresholds were exceeded. Health status was presented on an “Operator Dashboard.
- **Configuration and key management:** Secrets were safely stored in a secret management system. They were encrypted and were not stored in the source code. Key and critical changes were logged and recorded.

Non-Functional Requirements. The principal non-functional requirements were are:

- **Performance:** Expected time for the upload API response is less than 30 seconds. The CID checks were supposed to take no more than 3 seconds. More than one hundred concurrent verification requests were anticipated, without unacceptable decrease in speed.
- **Security:** All traffic was secured with secure versions of HTTPS (and TLS). No attempt was made to record passwords and/or secrets. External interfaces extrapolated and searched for output encoding and input validation. Monthly vulnerability scans were performed and there were no critical or high risk findings. Disagreed time limits to resolve issues were not met.
- **Availability and reliability:** Core functions for upload and verification to be expected to be up 99.95% monthly or better. MSSQL database backups and restorations were

used to cover the topics of recovery time and recovery point objectives. It helped to ensure that data could not be lost and that a service can recover if necessary.

- **Usability:** The usability verification results was presented in plain language. The user interface was done with ISM and accessibility method. It was crafted to meet WCAG 2.2 to the extent that it applies. QR code verification was designed to be easily accomplished on a mobile device.

- **Maintainability and supportability:** Front end is a single page application (Angular/TypeScript). It was based on a component design pattern following ISM coding conventions and dogma. Runbooks were created for development, staging and production.

- **Audit and compliance:** All the changes to reports and their configurations were audited, indexed and can be downloaded as a CSV or PDF for auditors. A system was in place to store integrity and provenance data. Provenance and integrity data was stored in the system. Records were kept for the required time according to the numbers of reports.

Use case diagram

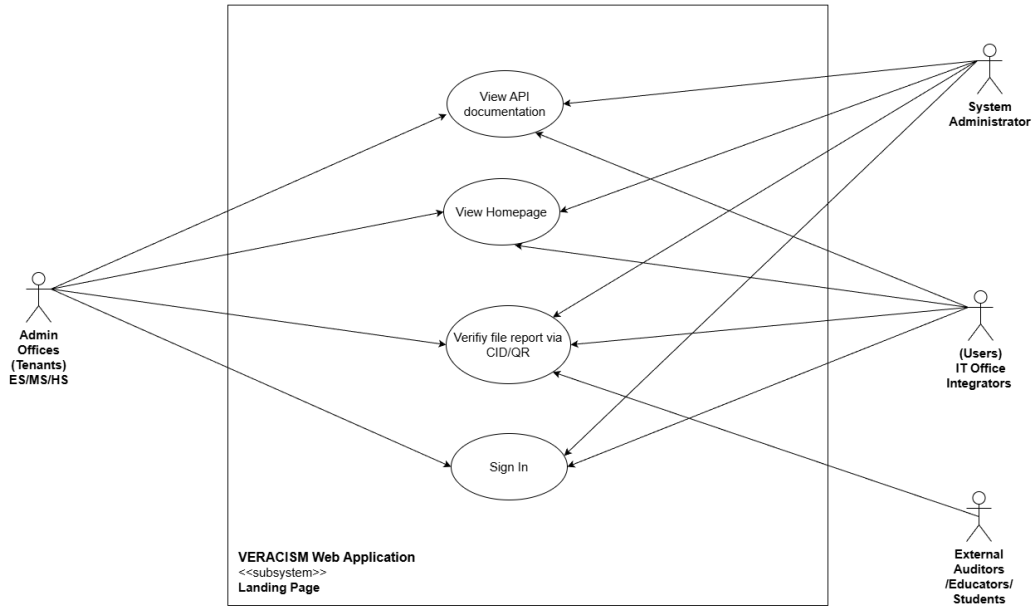

Figure 3. Use Case Diagram - Landing Page

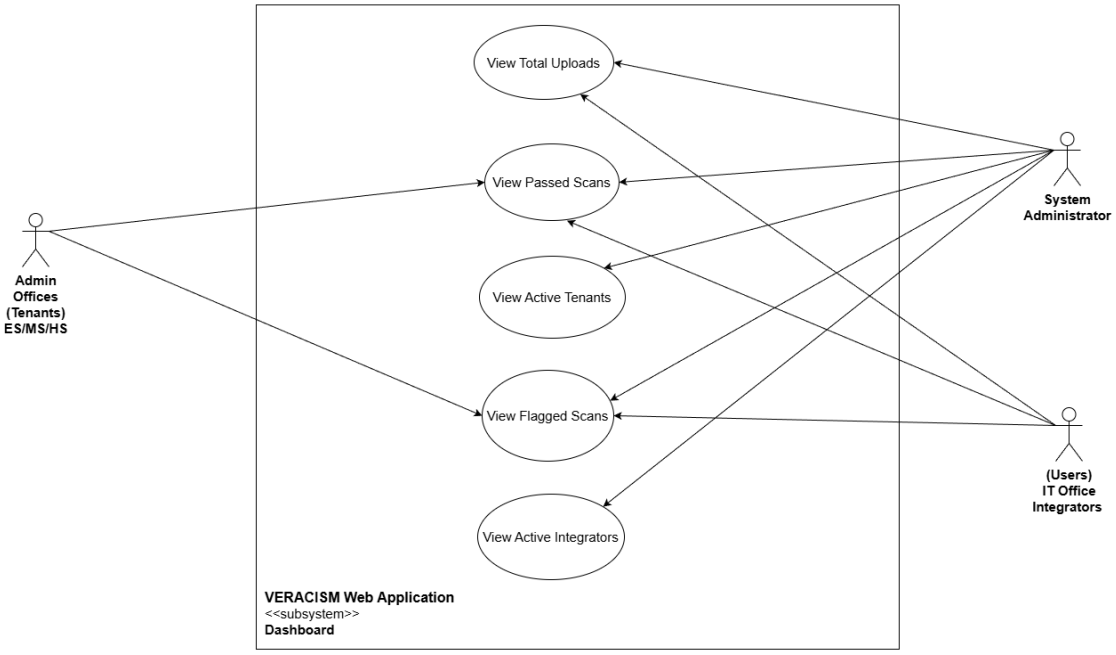

Figure 4. Use Case Diagram - Dashboard

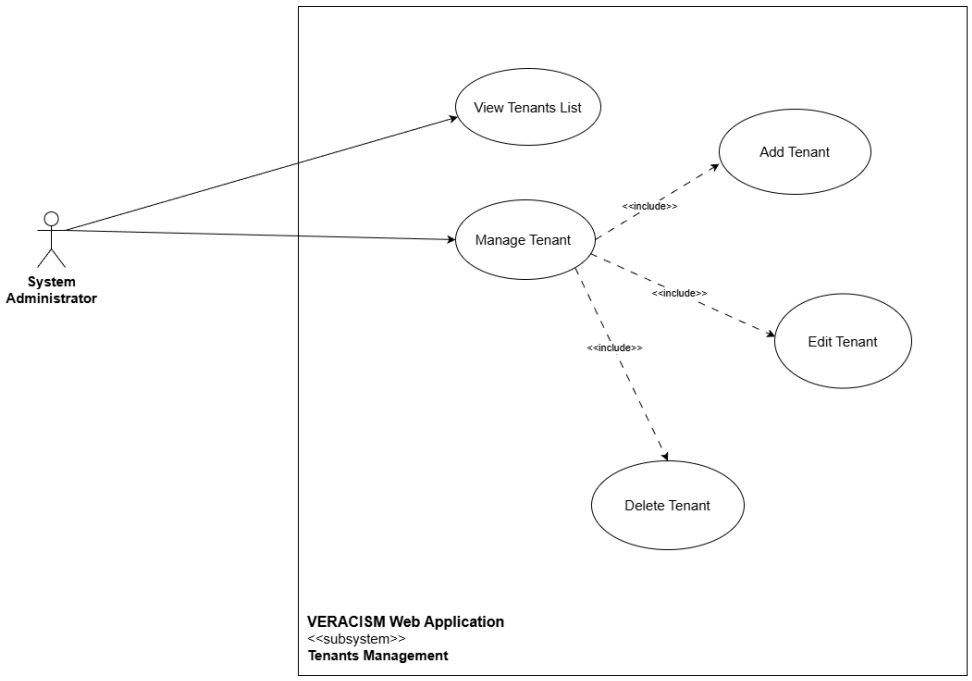

Figure 5. Use Case Diagram - Tenants Management

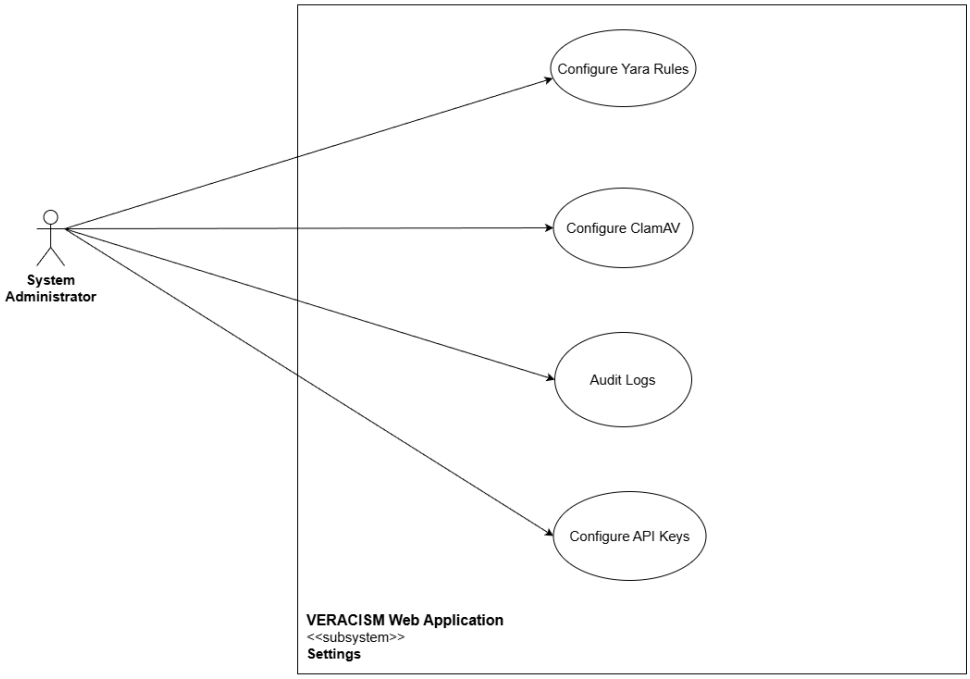

Figure 6. Use Case Diagram - Settings

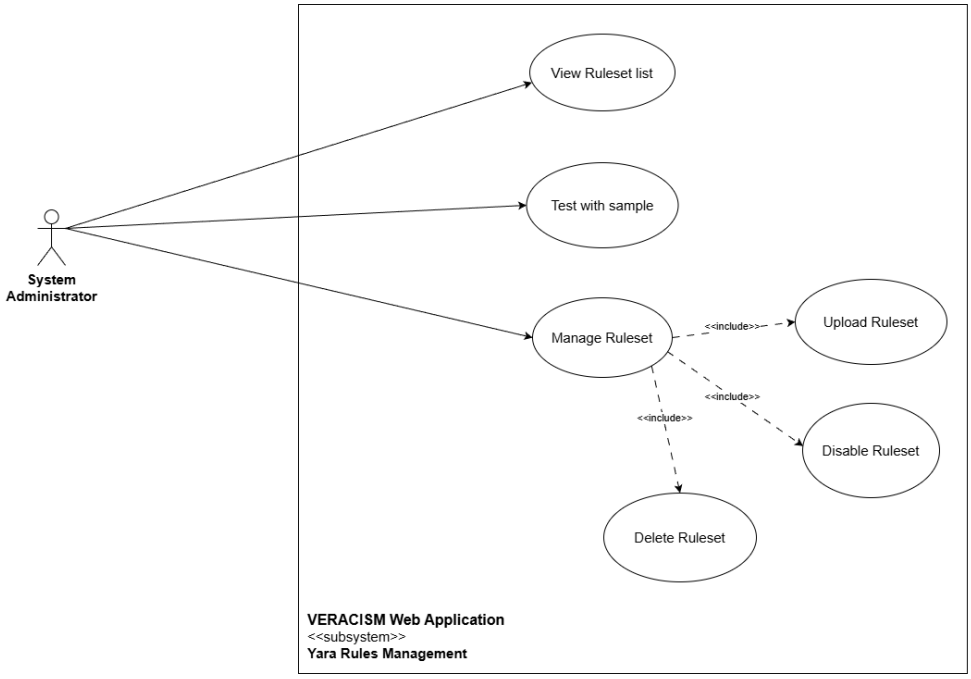

Figure 7. Use Case Diagram - Yara Rules Management

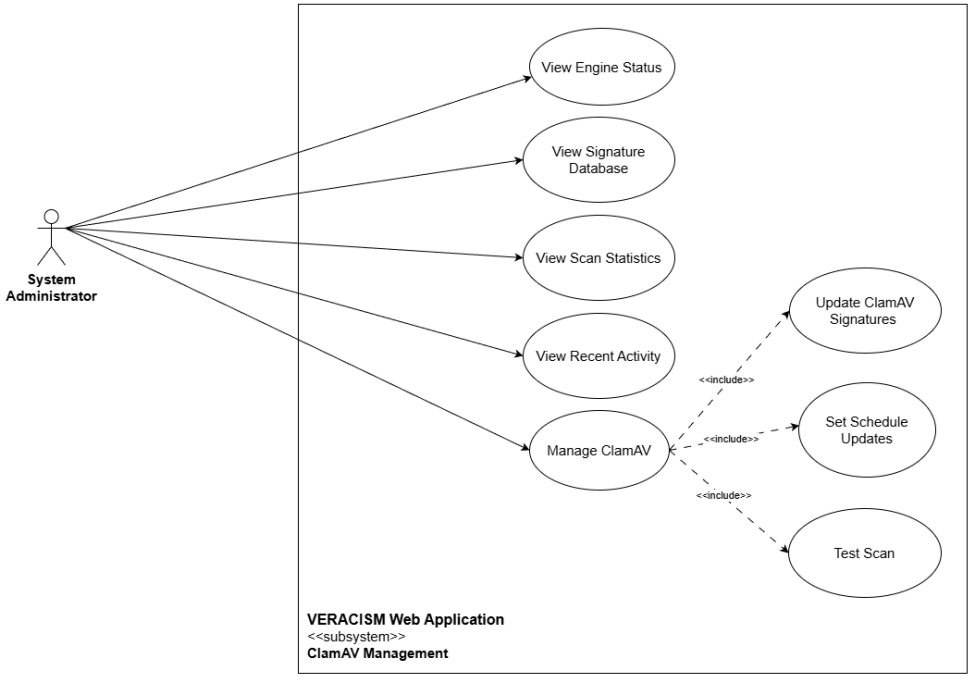

Figure 8. Use Case Diagram - ClamAV Management

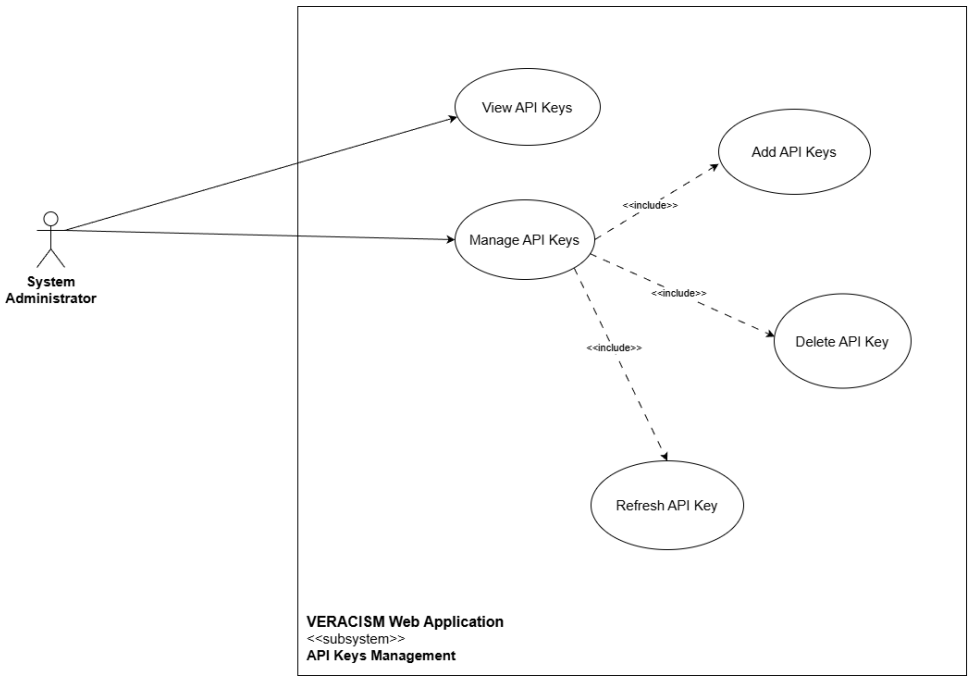

Figure 9. Use Case Diagram - API Keys Management

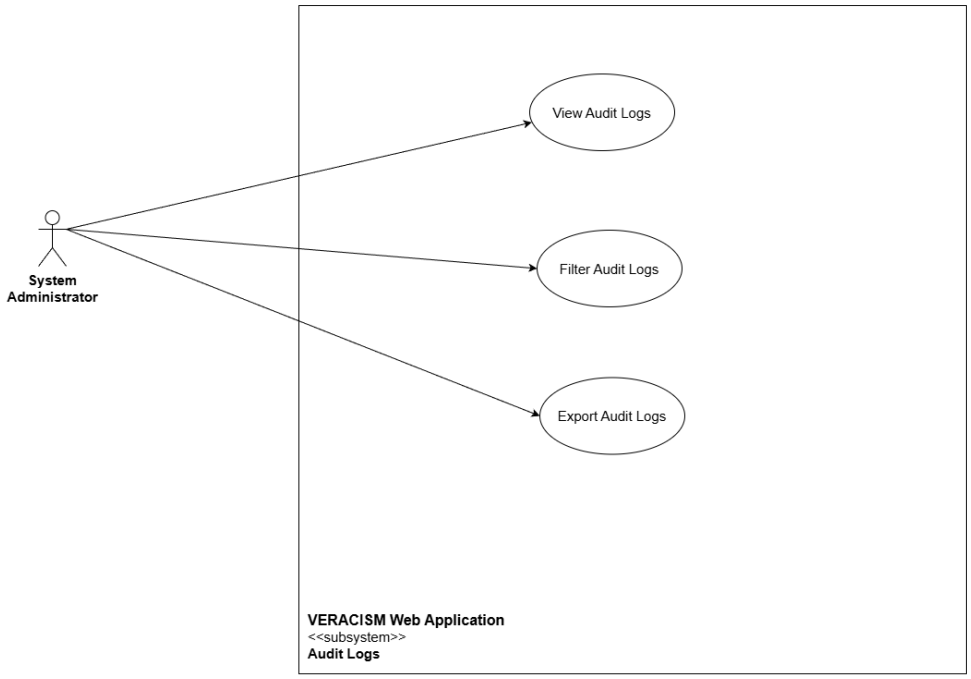

Figure 10. Use Case Diagram - Audit Logs

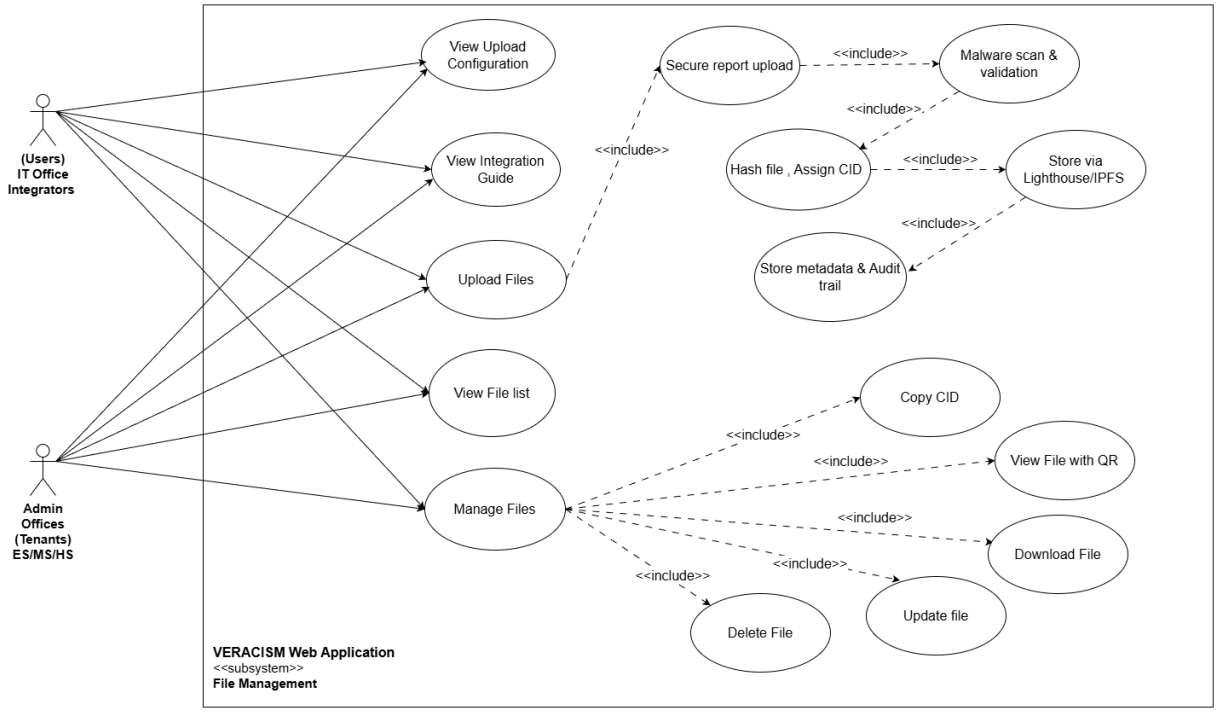

Figure 11. Use Case Diagram - File Management

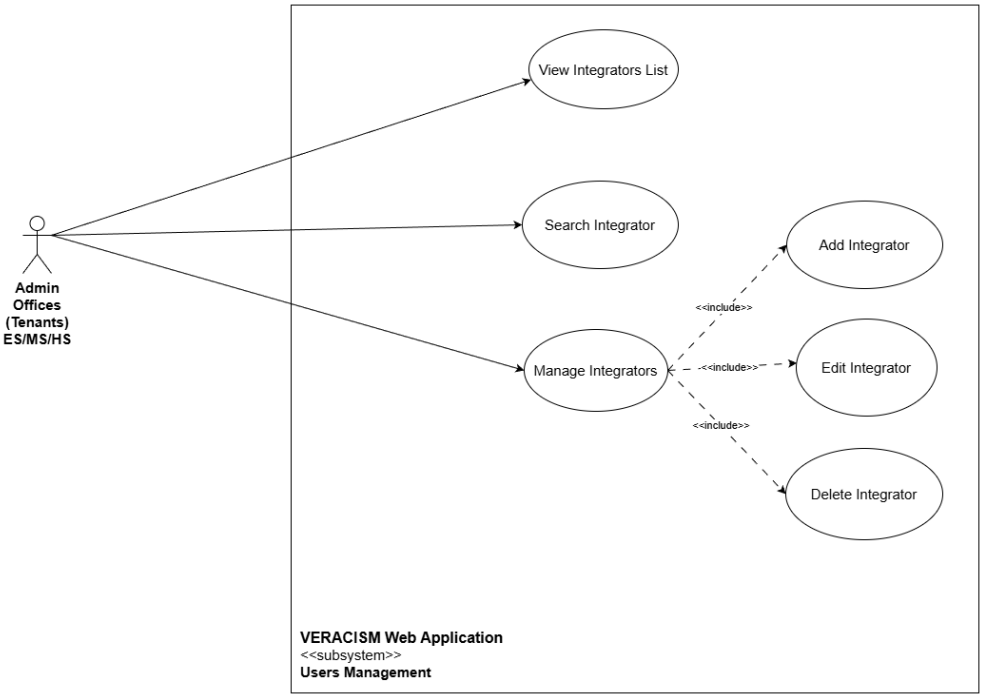

Figure 12. Use Case Diagram - Users Management

VERACISM was composed of several cooperating subsystems: the Landing Page and Dashboard for high-level entry and monitoring, Tenants Management and Users Management for multi-tenant administration, File Management for secure report handling, and configuration-focused subsystems for Yara Rules Management, ClamAV Management, API Keys Management, and Audit Logs. The actors (Admin, Tenants, Integrators) had different subsystems with different functional responsibilities. These actors were involved in the system, and demonstrated the allocation of responsibilities between modules.

The Landing Page and Dashboard subsystems (Figures 3 and 4) presented how Admin, Tenants, and Integrators had their initial interaction with VERACISM. From the Landing Page, all roles could access the application's homepage, view the API documentation and log in to the app. The Dashboard diagram then displayed the overall system status: total uploads, passed scans, flagged scans, active tenants (for Admin) and active integrators (for Tenants). These diagrams were combined to give users a broad overview of how users were entering the system and the current security and operational state of this system.

The Tenants Management and Users Management subsystems (Figures 5 and 12) focused on multi-tenant control and delegating integration responsibilities. The Tenants Management diagram displayed the Admin's view on the tenant list and how tenants were managed using the use cases for adding, editing and deleting tenants. This was the way VERACISM was responsible for the onboarding or offboarding of entire organizations. The Users Management diagram illustrated how the Tenant user manages their own integrators: viewing the list of integrators, searching for integrators and using the actions included in the diagram to add, edit or delete integrator accounts. These diagrams explained the interaction between central administration and tenant-level administration in order to gain control of whom to include systems to VERACISM.

The Settings, Yara Rules Management, ClamAV Management, API Keys Management, and Audit Logs subsystems (Figures 6, 7, 8, 9, and 10) illustrated how security and configuration were handled by the Admin. The Settings diagram summarized the main configuration areas: Yara Rules, ClamAV, API keys, and viewing

audit logs. The Yara Rules Management diagram displayed the Admin's view of rule sets, including their testing, upload, disable, and delete. Admin could see the status of the engine, signatures, scan statistics, recent activity and manage updates, test scans, and logs in the ClamAV Management diagram. The API Keys Management diagram showcased how the Admin was able to view, add, delete and refresh API keys for integrations. The Audit Logs diagram demonstrated Admin's ability to view, filter and export logs for traceability, investigations and compliance. Combined these diagrams outlined VERACISM's approach to centralizing security and operational control within a limited Admin user role.

The File Management subsystem (Figure 11) captured the core business flow for Tenants and Integrators. The diagram depicted the Tenants and Integrators' pre-send upload settings and guides. The Upload Files use case included secure upload, scanning for malware, validation, hashing, assigning CID, uploading to Lighthouse/IPFS and metadata and audit logging. Once uploaded, users were able to view files, copy CIDs, view QR codes, download files and (if permitted) update or delete files. This demonstrated, how VERACISM ensured a secure, traceable and content-addressed workflow for report files.

The use case diagrams demonstrated the workings of VERACISM in the field. They described how they entered into the system, how they managed the tenants and integrators, how the security and configuration was provided by the Admins, and how the reports were uploaded, scanned, stored, and retrieved. This enabled identification of the actors and responsibilities required for the platform's security, auditing and adherence to ISM's operational requirements, as well as interactions between actors.

Sequence diagram – Upload Workflow

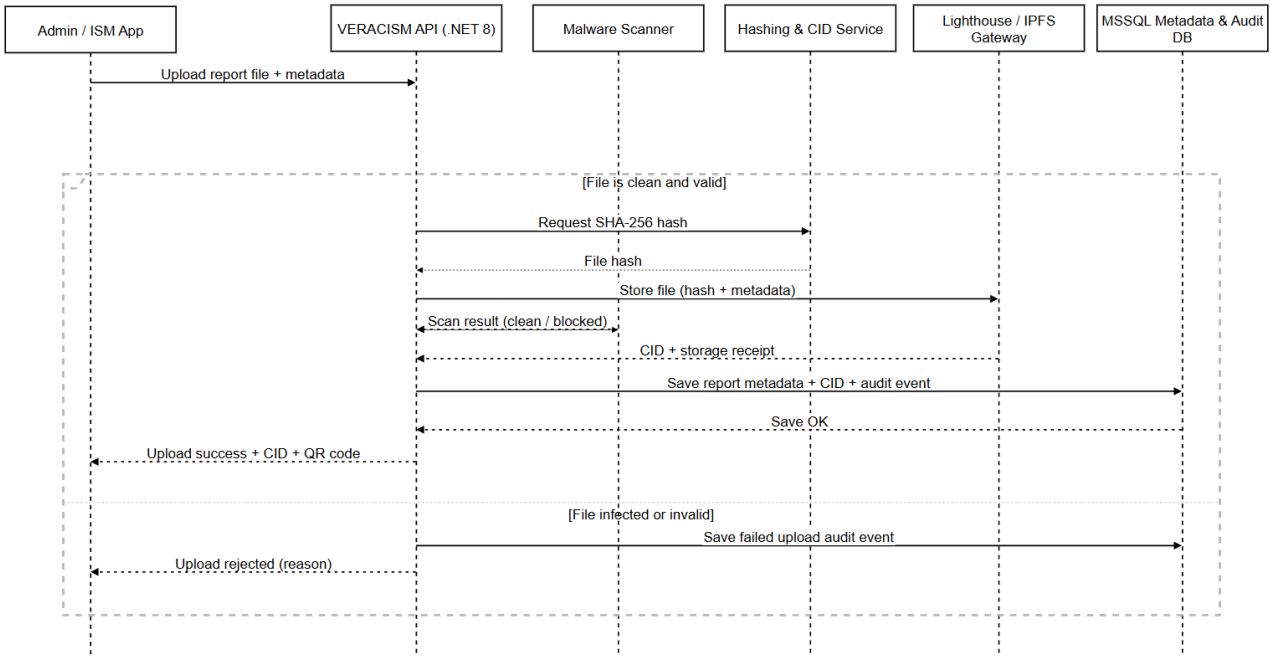

Figure 13. Sequence diagram – Upload Workflow

Figure 13 showed how a report file moved from submission to acceptance or rejection. The key lifelines included the Admin / ISM App, the VERACISM API, the Malware Scanner, Hashing and CID Service, the Lighthouse/IPFS gateway, and a MSSQL metadata and audit database. The diagram presented the time of security checks, hash creation, CID creation, file storage and audit record creation.

The report file and metadata were uploaded to the VERACISM API in the upload workflow. The API sent the file to the Malware Scanner. If the file was clean and valid, the API requested a SHA-256 hash, sent the file to Lighthouse/IPFS, got the CID and storage receipt and added the metadata and audit event to MSSQL. The API then sent back the CID and QR code for a success message.

Failure path is also shown in the diagram. The Malware Scanner blocked or returned an invalid result if the file was infected or was not a valid file. The API aborted, logged a failed audit event to MSSQL, and sent an “upload rejected” message back containing the reason. This made it possible to track which files were rejected and to make sure that

anything that was uploaded was not put into permanent storage.

Sequence diagram – File Verification Workflow

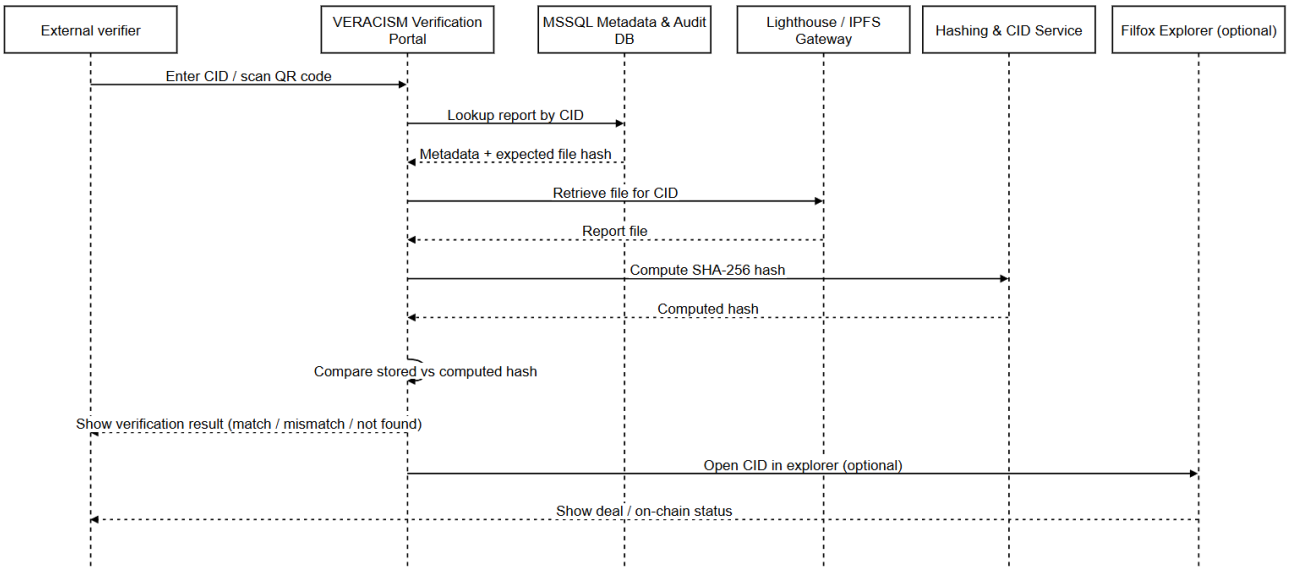

Figure 14. Sequence Diagram – File Verification Workflow

Figure 14 showed the main participants in a CID or QR code verification request. These consisted of External Verifier, VERACISM Verification Portal, MSSQL metadata and audit database, Lighthouse/IPFS gateway, Hashing and CID Service, and Filfox. The process of lookup, retrieval of files from disk and comparing the hashes was illustrated in the diagram.

The External Verifier input a CID or scanned a QR code in the normal flow. The portal retrieved the metadata for the report from the MSSQL, such as the expected hash and report information. The portal found the metadata and retrieved the file from Lighthouse/IPFS using the CID, and forwarded it to the Hashing and CID Service.

The Hashing and CID Service was able to calculate a new hash using the SHA-256 algorithm and send the portal the new hash. The portal has compared it with the stored hash of MSSQL. The portal indicated ”match” if the hashes matched or key metadata; or ”mismatch” or ”not found” if the hashes differed or the CID or report could not be found.

The verifier can also go to the Filfox and open the CID to see deal or on-chain status. This provided transparency by enabling the verifier to validate VERACISM with the public

decentralized storage evidence.

Database diagram

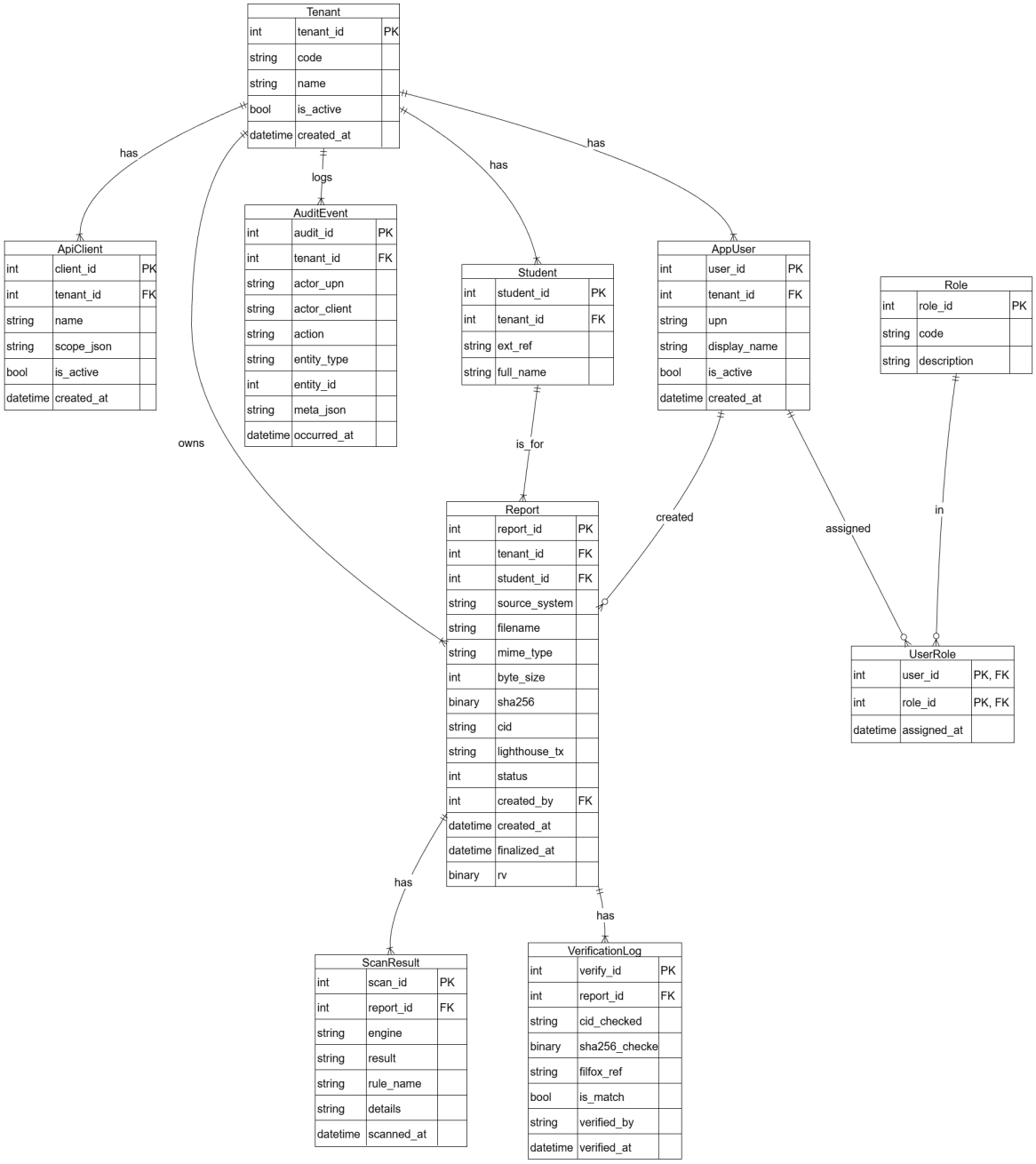

Figure 15. Database diagram

The Figure 15 followed the data model defined in the SRS and was implemented in MSSQL. The ERD clustered the information in the system into tenants, users, reports,

security scans, verification activity, and audit events.

The Tenant table was at the centre. Owners of each tenant were exclusively each organization, and they owned their respective reports, API clients, users, students and audit events. This is used to provide tenant isolation by tenant id.

User access was managed with the AppUser, Role and UserRole tables. Every user was a member of one tenant and roles were assigned via UserRole. This enabled a flexible RBAC with minimum duplication.

The ApiClient table was used for system integrations. Every API client was part of a tenant and contained the client ID, name, scope, status and timestamps. This enabled the difference between human activity and system triggered activity to be identified.

The Student and Report tables handled report ownership. Every student was a member of one tenant and each report was a report of one student and one tenant. Reports recorded information about the file, including its name, type, size, SHA-256 hash, CID, Lighthouse reference, status, and date/time information.

The ScanResult table contained the results of scans executed on each report along with details, rule name, result, scan engine and scan time. This enabled the system to maintain a scan history for each report.

Each verification event was recorded in the VerificationLog table, along with the CID that was verified, the observed SHA256 hash, any external references, the result of the verification, the verifier, and the time of the verification. This enabled auditors to check the frequency of reporting and who was doing the reporting.

Operational and security actions were recorded throughout the system into the AuditEvent table. It contained information about actors, actions, affected entities, metadata and event time. This enabled an unalterable and searchable audit trail.

The ERD revealed that each tenant owned its users, API clients, students, reports, scan results, verification logs and audit events. Reports remained as the main record, with the students, creators, scans, verifications and audit history connected to them.

Deployment diagram

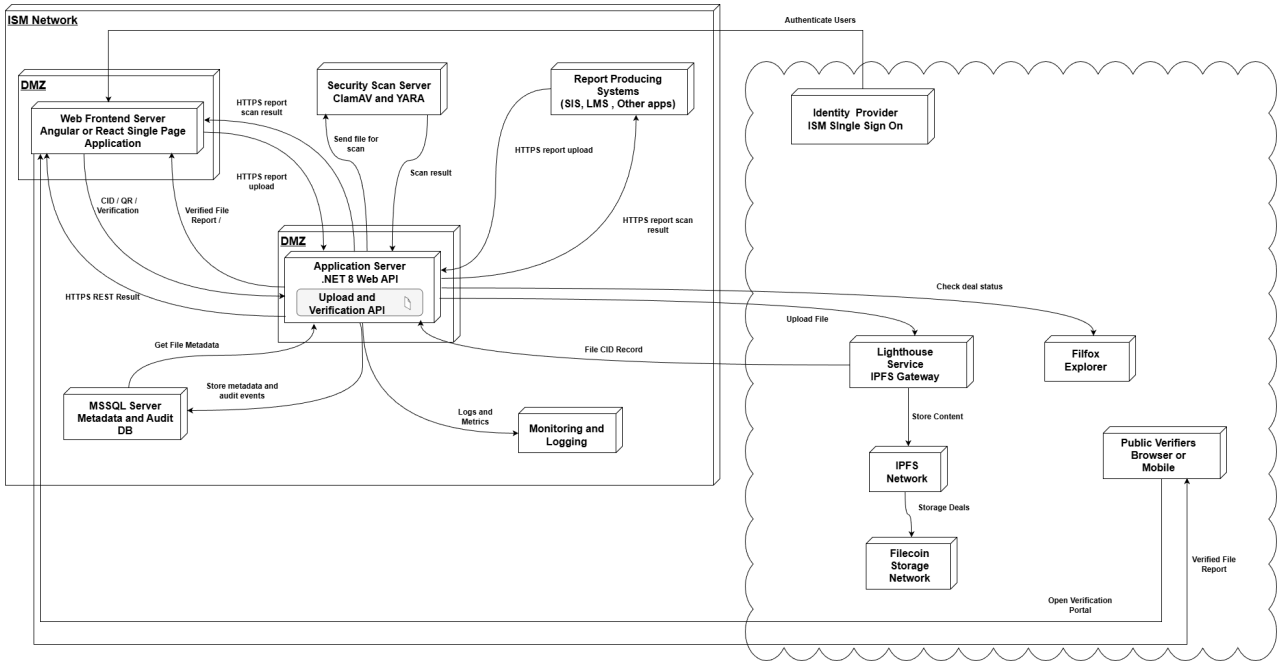

Figure 16. Deployment diagram

Figure 16 placed each VERACISM component either inside the ISM network or in external services and mapped the connections between them. On the ISM side were the web frontend, the .NET 8 application server, the security scan server, the MSSQL metadata and audit database, and the monitoring and logging stack. On the external side were the Lighthouse IPFS gateway, the IPFS and Filecoin networks, Filfox, the ISM identity provider, and public verifiers using a browser or mobile device.

Within the ISM network, users and report-producing systems, such as SIS, LMS, and other apps, first connected to the Web Frontend Server in the DMZ. This frontend served the Angular or React single-page application, which then called the Application Server (.NET 8 Web API) over HTTPS. The application server exposed the upload and verification APIs and was placed in its own DMZ segment so that database and monitoring servers stayed behind an additional boundary.

When a report was uploaded, the Application Server pushed it to the Security Scan Server which ran the ClamAV and YARA apps. When scanned, the application stored

metadata and audit events in MSSQL, and logs and metrics were sent to the Monitoring and Logging stack. If accepted, the application uploaded the content to the Lighthouse IPFS gateway via HTTPS, got the CID and stored the storage information in MSSQL.

The ISM Single Sign-On identity provider provided user identity and sign-in. It verified users and returned a token for the Web Frontend and Application Server to validate users. This allowed for one repository of passwords and for VERACISM to operate on upload, storage and verification.

On the decentralized side, Lighthouse wrote the file to IPFS and facilitated offers to store it on Filecoin storage. Students, parents, or auditors signed up as public verifiers goes to the VERACISM verification portal via a browser or mobile device. The portal obtained metadata from Application Server and content from Lighthouse/IPFS to hash. The verifier can, if necessary, open up the CID in Filfox to see the storage deal details.

This deployment view illustrated the components that remained in the ISM network and the ones that were external to the network. It also highlighted the flow of the report, from the uploading from the front end, through scanning and storage out to Lighthouse, IPFS and Filecoin, and then back in on verification. This demonstrated VERACISM's ability to safeguard the internal systems with the ability to verify externally.

Implementation Timeline

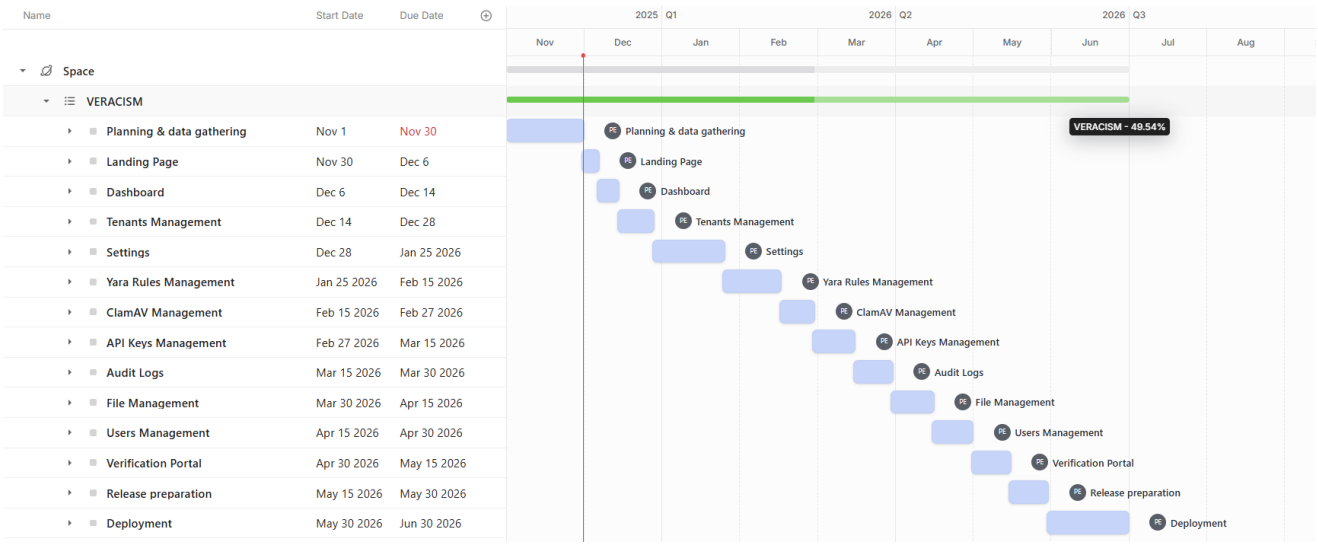

Figure 17. Implementation Timeline

Figure 17 showed the VERACISM schedule from November 2025 to June 2026. The plan covered a DevSecOps lifecycle from planning and data collection to passing all subsystems and finally to the release preparation and deployment. A bar for each subsystem in the SRS and use case models. The security-related modules were put first before the file and user workflows.

Implementation and testing roadmap

The implementation and testing roadmap included VERACISM validation in the Development, UAT and Production environment. Each subsystem was subjected to unit testing, API contract checking and security testing prior to promotion in Development. The environment for the UAT reflected the school environment, with tenant setup, role-based access, sample reports, and upload and verification processes replicated.

UAT was a focus on end-to-end school workflows, and it created logs, audit events, verification results, and test evidence. Production was used for controlled release and post deployment checks upon UAT sign-off. These tests established that these mechanisms,

such as configuration, monitoring, logging and security gates, functioned in live settings.

Throughout all phases the following kinds of tests were planned as part of the validation process: functional testing, integration testing, performance testing, user acceptance testing, and security testing. Details of the method for each testing area was described in the Project Assessment section.

Testing stage	Schedule	Focus and coverage
Functional testing	Nov 2025 to Apr 2026	End to end functional tests across modules and API endpoints. Verified that core features worked as intended, including file upload processing, file validation, malware scanning, hashing, CID generation, metadata storage, and access control behavior based on RBAC and ABAC rules.
Integration testing	Feb 2026 to May 2026	End-to-end flows across the API, MSSQL, Lighthouse/IPFS, and the verification portal using realistic report samples and test tenants. Confirmed that subsystems behaved correctly when combined.
Security testing	Mar 2026 to May 2026	Vulnerability scanning and manual security testing aligned with the institutional VAPT process. Findings were fixed and re-tested before go-live.
Performance testing	Apr 2026 to May 2026	Response time and concurrency tests for upload and verification confirmed service level targets under expected operating load.
User acceptance testing	May 2026	Role-based testing with the System Administrator, Administration Offices (registrars), students, and external verifiers using scripted tasks derived from the system requirements and acceptance criteria.

Table 4. Testing stages and schedule for VERACISM

User acceptance testing timeline

Phase	Dates (2026)	User roles	Scope and outputs
UAT setup and briefing	May 4 to May 6	System admin, IT office, registrar lead	Accounts issued, test tenants created, test data loaded, task scripts distributed, defect log format agreed.
Registrar UAT	May 7 to May 11	Registrars (Administration Offices)	Upload workflow, status tracking, dashboard views, audit log review, and verification checks using issued reports and assigned CIDs.
Student UAT	May 12 to May 13	Students	QR access, verification result screen comprehension, metadata summary review, and reporting of unclear messages or steps.
External verifier UAT	May 14 to May 18	External verifiers (auditors or partner schools)	QR scan to verification result, CID entry verification, failure messaging for unavailable content, and repeat checks from different networks.
End-to-end UAT and sign-off run	May 19 to May 21	All roles	Full journey from report upload to verification, including negative tests for tampered files and unavailable deals, then sign-off decisions.
Fix and retest window	May 22 to May 29	IT office, registrar lead	Retest of defects, regression run, evidence capture, final acceptance sign-off.

Table 5. User acceptance testing timeline for VERACISM

UAT was scheduled while Users Management and the Verification Portal were available for end-to-end workflows, with closure before release preparation and deployment.

User Interface Wireframe

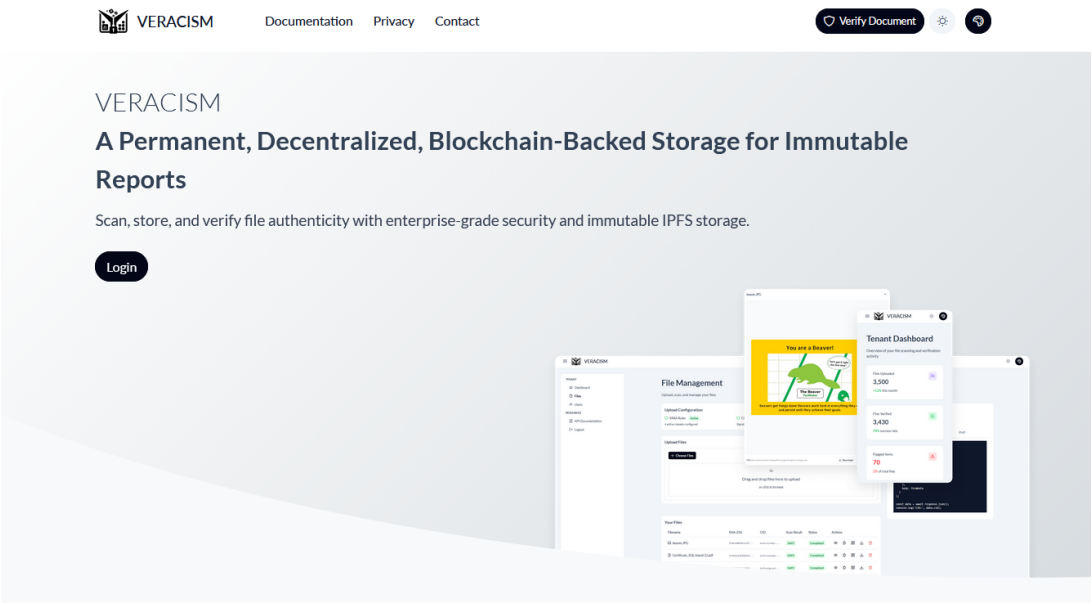

Figure 18. Landing Page - Hero Section

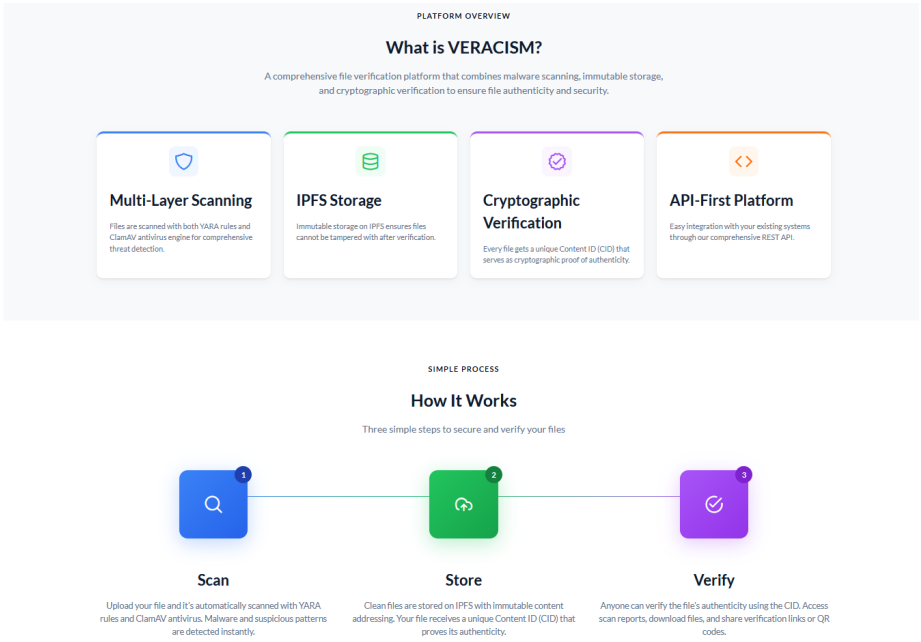

Figure 19. Landing Page - About / How It Works Section

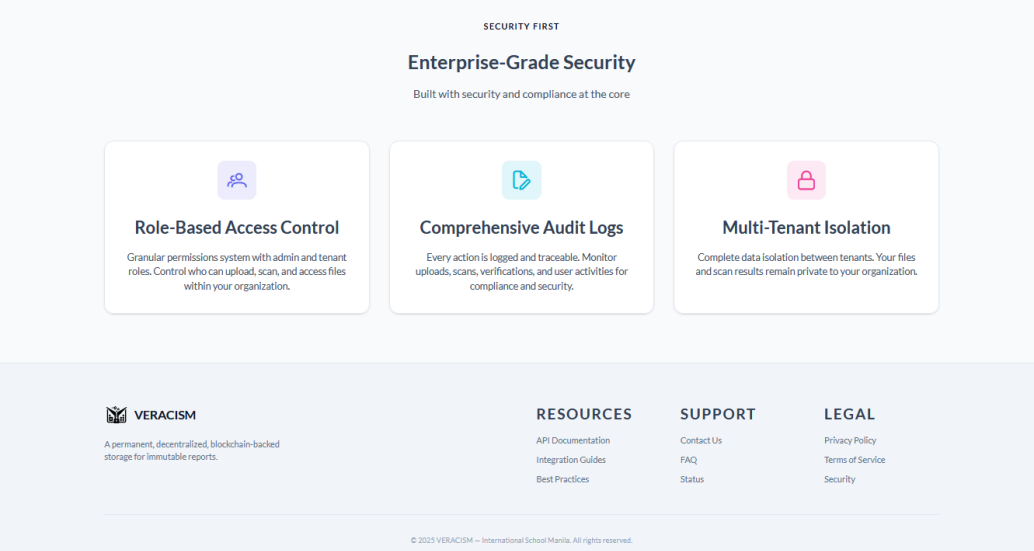

Figure 20. Landing Page - Footer Section

Landing page presents a permanent, decentralized, blockchain-backed file verification platform that scans, stores, and verifies documents using multi-layer malware scanning, IPFS immutable storage, and cryptographic verification. It highlights simple Scan-Store-Verify workflows, enterprise-grade security with role-based access, audit logs, multi-tenant isolation, and API-first integration.

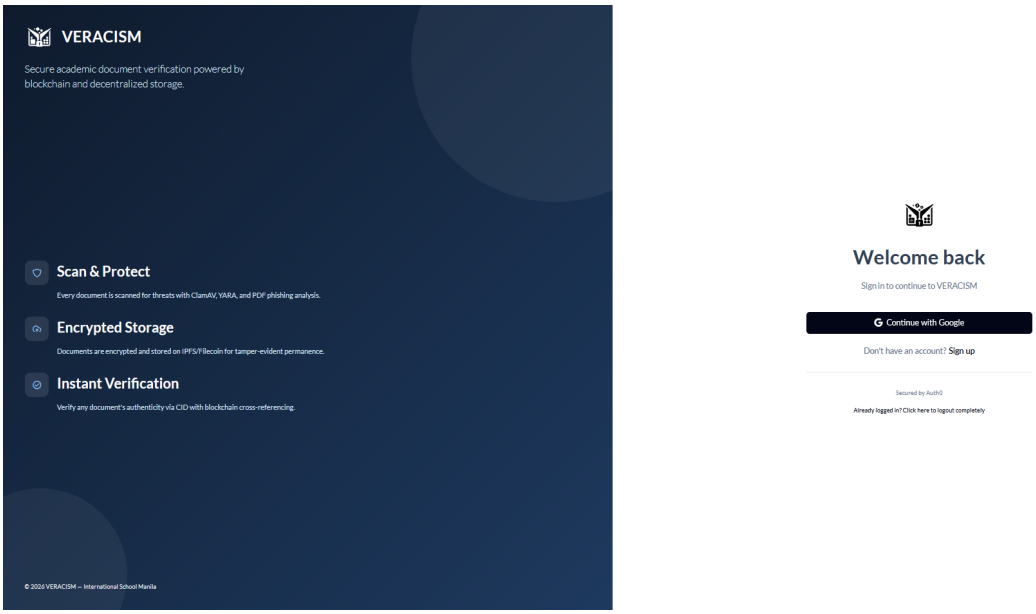

Figure 21. Login Page

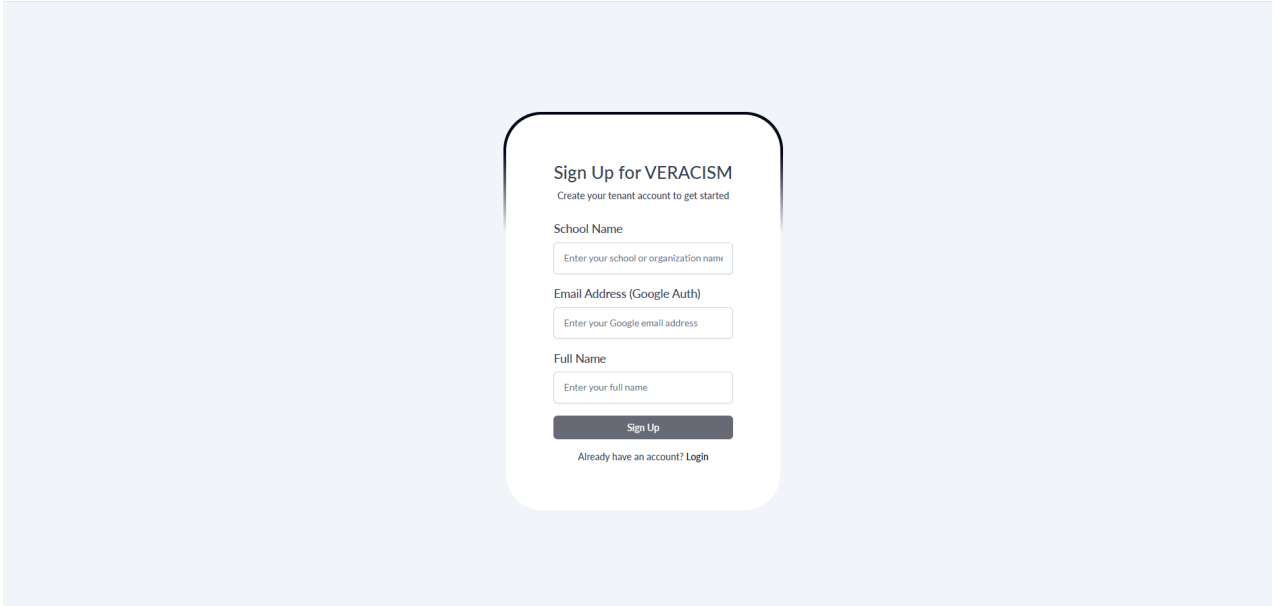

Figure 22. Sign up Page

VERACISM auth flow features a clean, centered card layout for both signup and login. New tenants can register by entering their school name, Google-based email address, and full name, while existing users sign in using “Continue with Google,” managed by Auth0. After authentication, users are automatically routed to their own

dashboard based on their role.

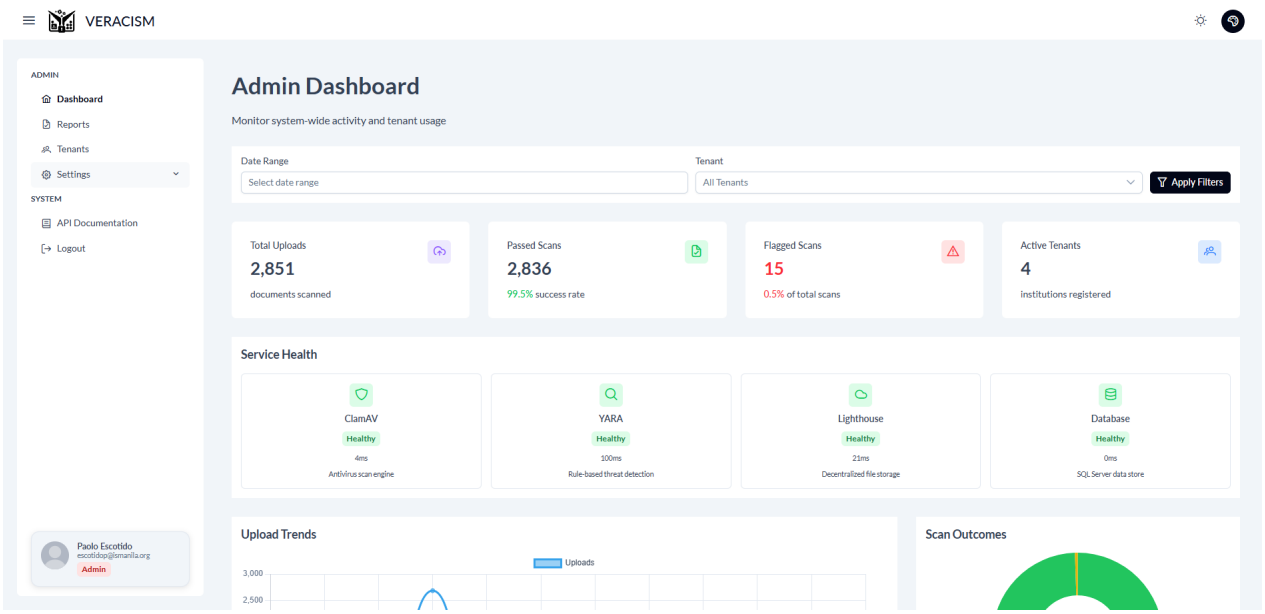

Figure 23. Admin - Dashboard Page

After an admin logs in, they see this analytics dashboard showing system-wide file uploads, scan results, active tenants, filters by date/tenant/type, and visual trends for uploads and scan outcomes.

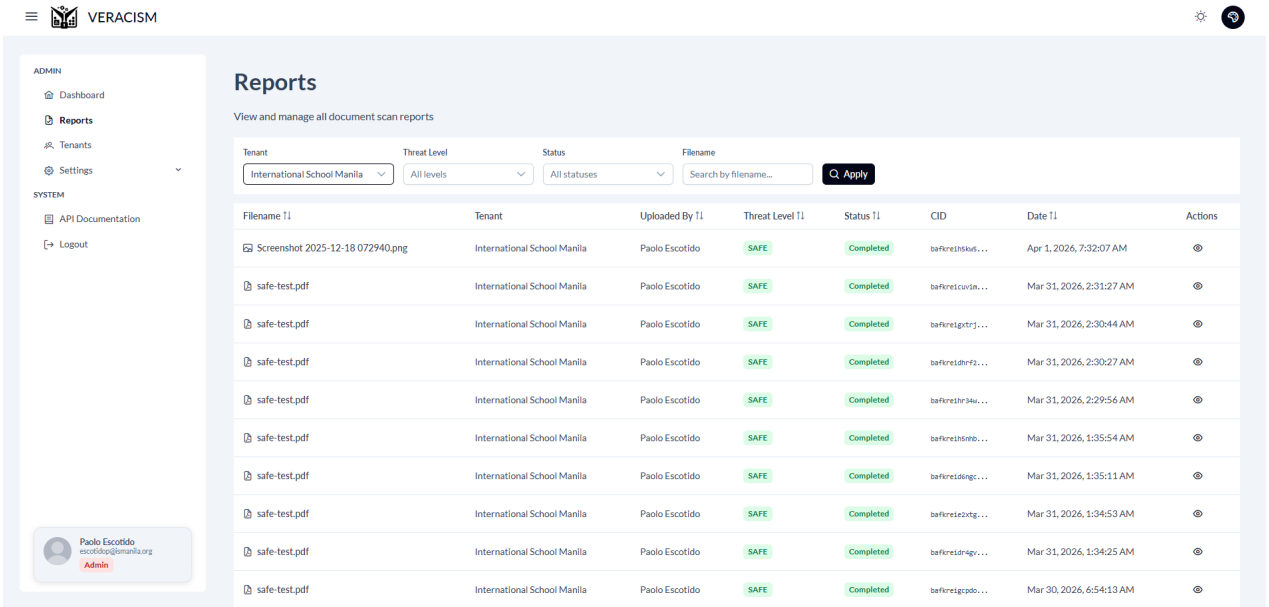

Figure 24. Admin - Reports

Admin Reports page lets admins review the report uploaded in all registered tenant

organizations.

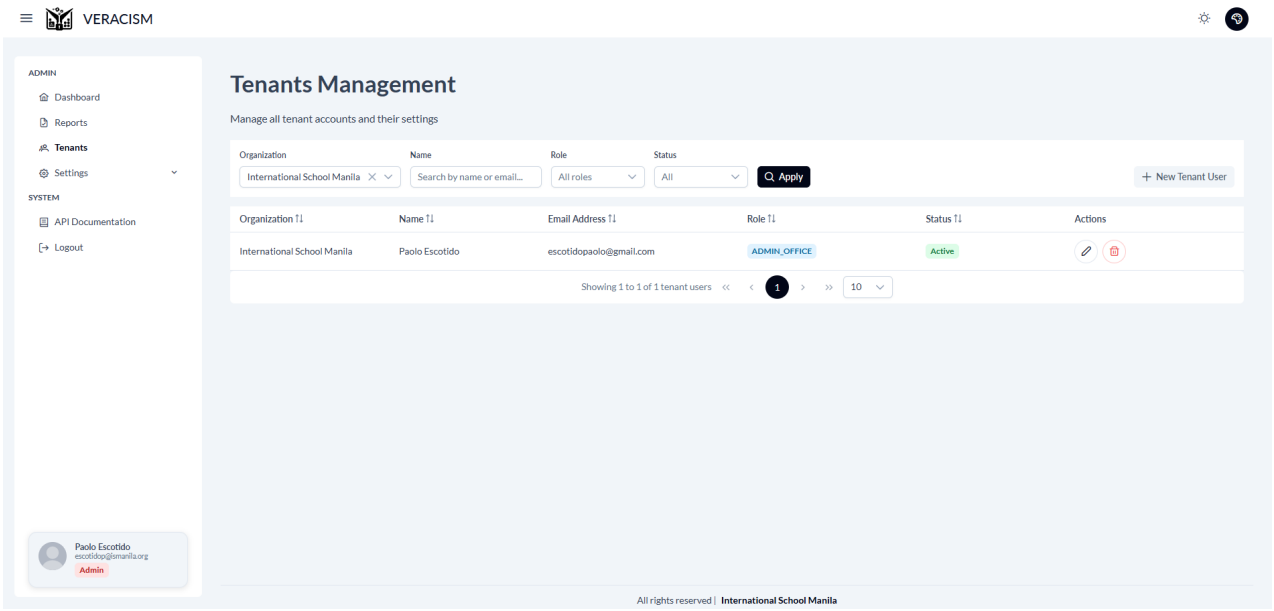

Figure 25. Admin - Tenants Management

Admin Tenants page lets admins review, approve, and manage all registered tenant organizations, including their users, roles, status, and actions.

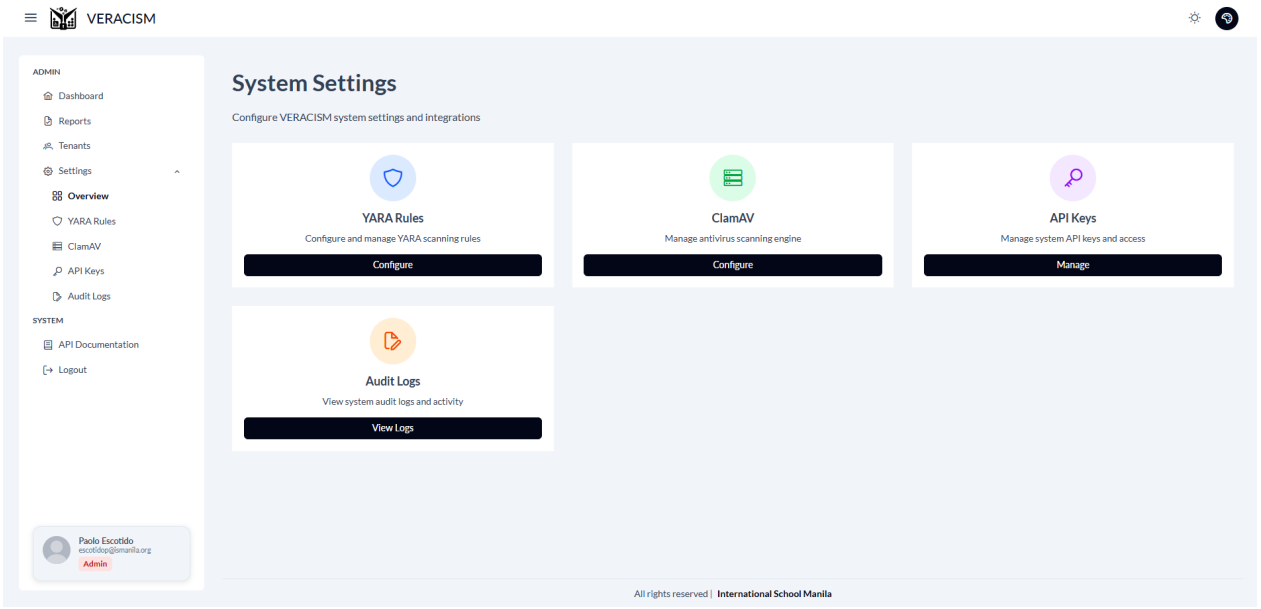

Figure 26. Admin - Settings Page

System Settings page lets admins configure YARA rules, manage the ClamAV

antivirus engine, control API keys, and review system audit logs for VERACISM.

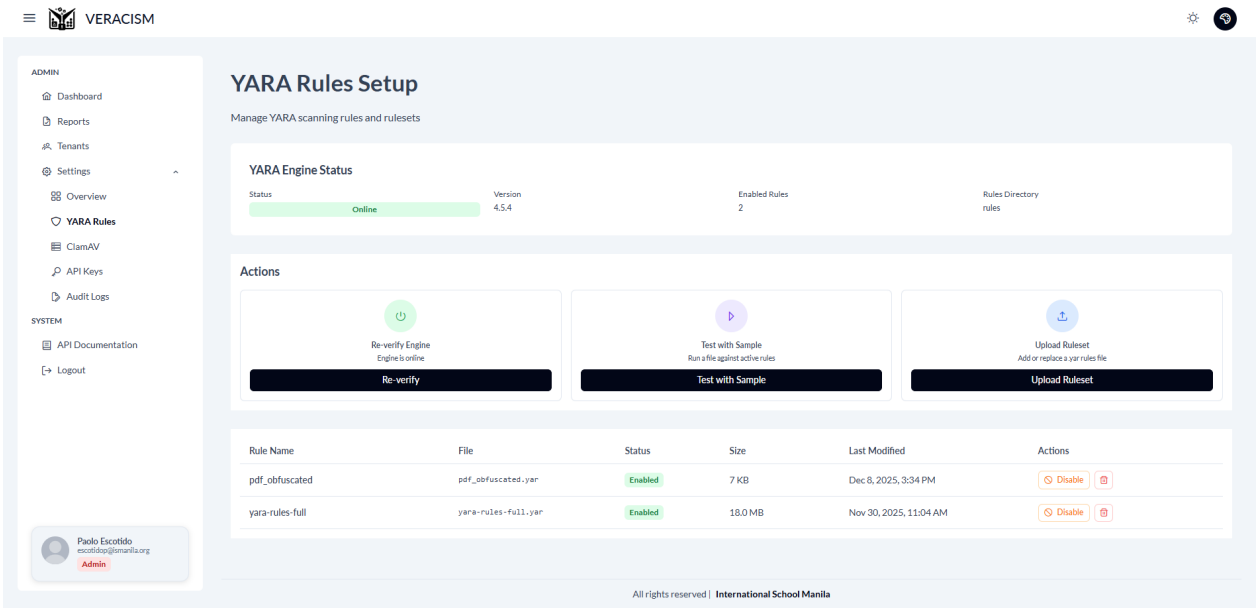

Figure 27. Admin - Yara Rules Setup

YARA Rules Setup page lets admins configure, enable or disable, test, and upload YARA rulesets for malware, ransomware, suspicious scripts, and custom detections.

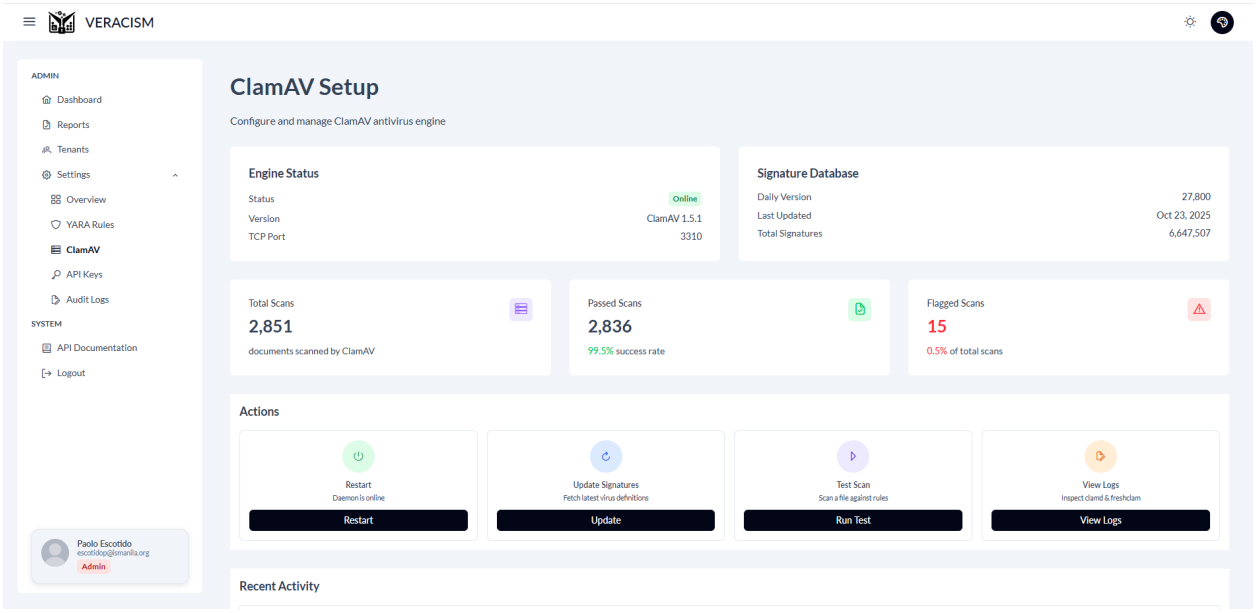

Figure 28. Admin - ClamAV Setup

The ClamAV Setup page lets admins monitor the antivirus engine, manage and

schedule signature updates, run test scans, and review recent scan activity and statistics.

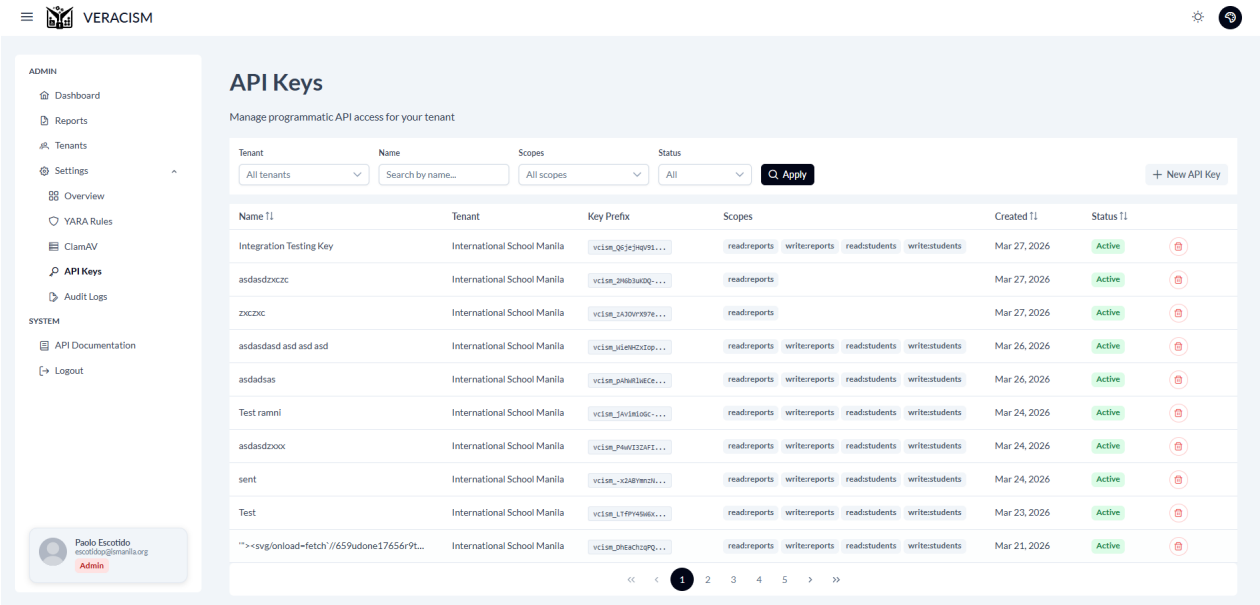

Figure 29. Admin - API Keys

The API Keys page allows admins to manage API keys across all registered tenants. It lets them view, create, and monitor tenant API keys.

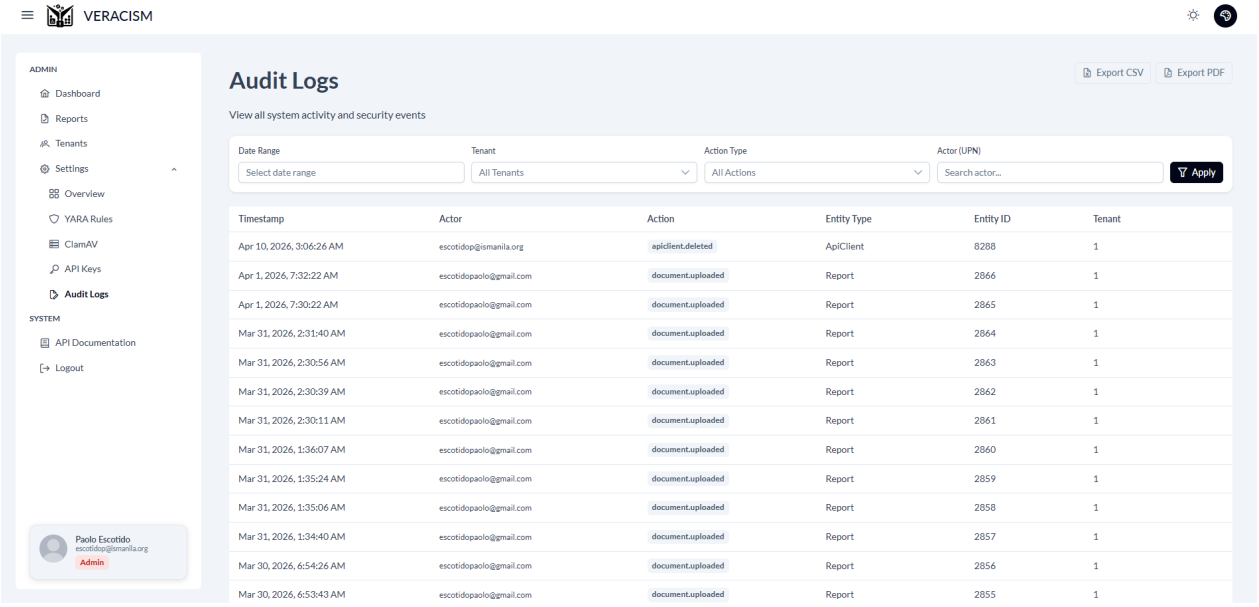

Figure 30. Admin - Audit Logs

The Audit Logs page lets admins review system activity and security events across all

tenants. It helps track actions, actors, and affected records for monitoring and investigation.

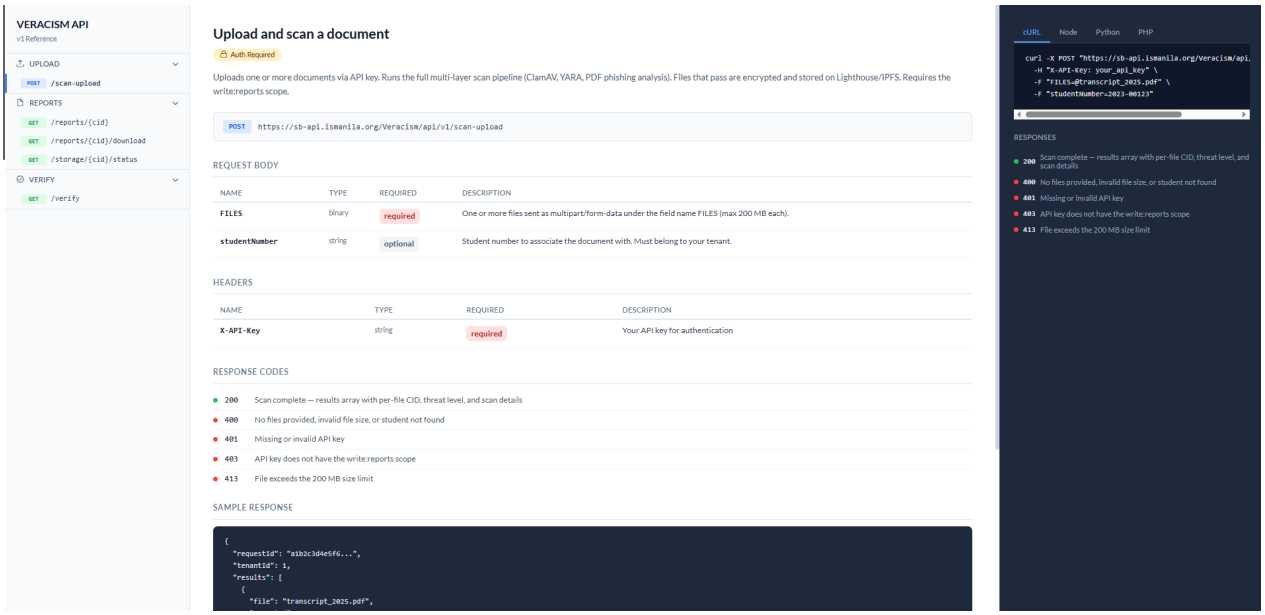

Figure 31. API Documentation

The API Documentation page lets anyone view Veracism's REST API base URL, key features, authentication with API keys, available endpoints, rate limits, and error codes so developers can integrate file scanning and verification into their existing systems.

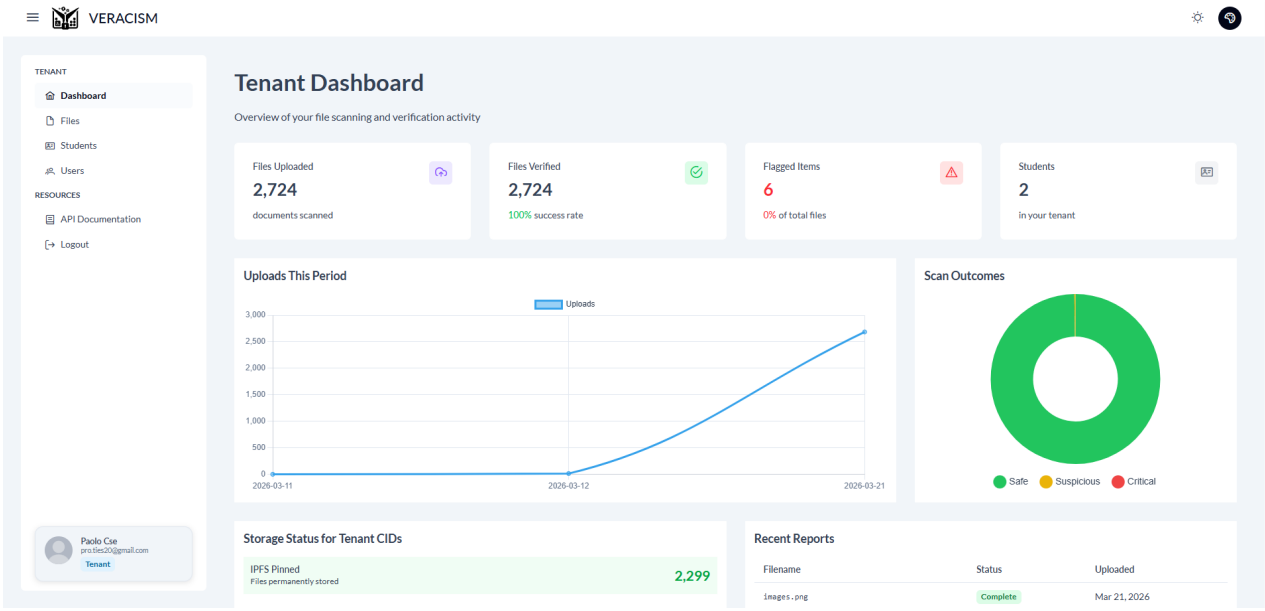

Figure 32. Tenants - Dashboard

After tenants successfully logging into the system, the tenant is redirected to this dashboard showing their organization's file uploads, verification success, flagged items, storage usage, upload trends, and scan outcomes.

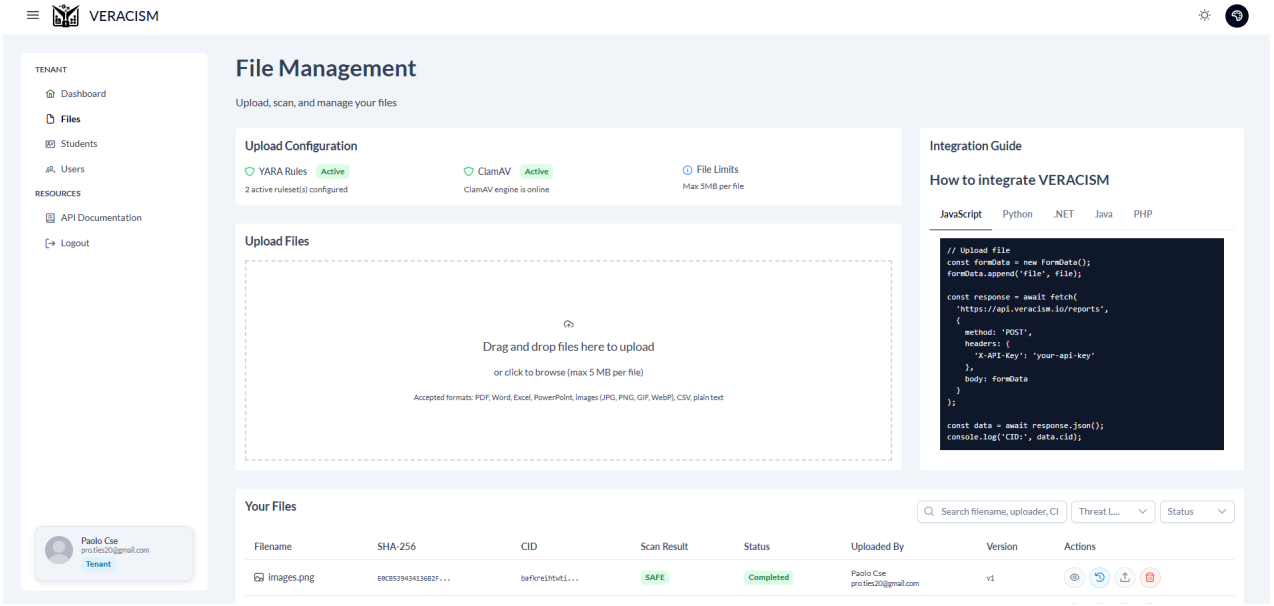

Figure 33. Tenants - File Management

File management page lets tenants upload and manage their documents (view hashes, CIDs, scan results, and actions) while in the right panel, Integration Guide provides code examples showing how to call the Veracism API directly from their existing website or system.

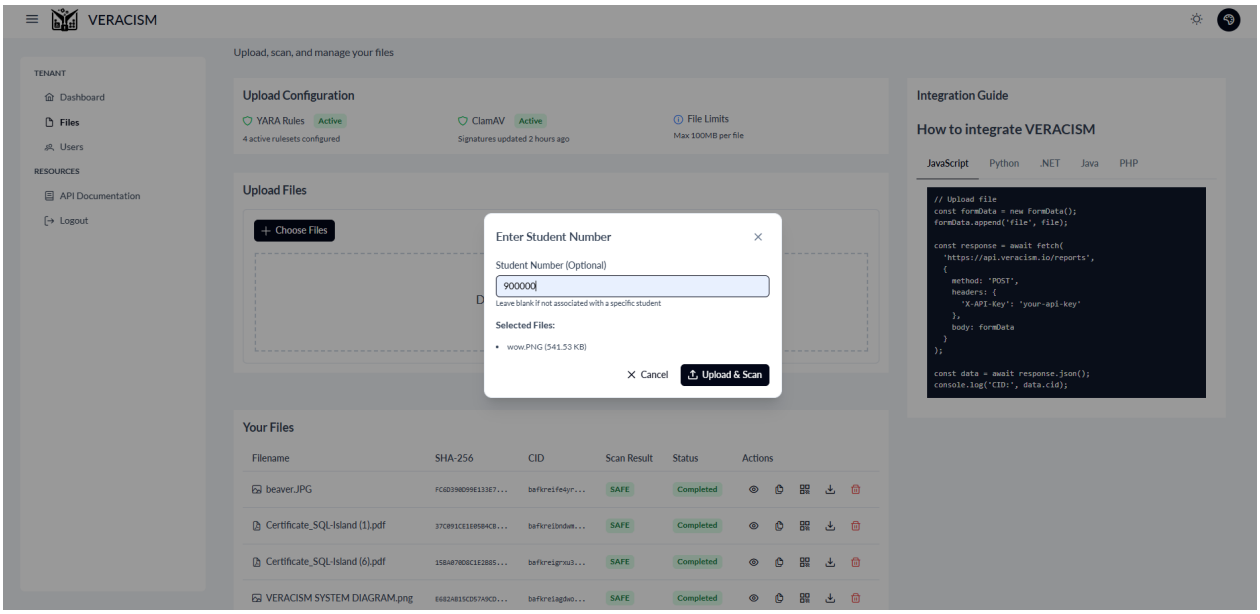

Figure 34. Tenants - File upload Process

During this upload process, tenants are able to select the files to upload from their device and, if desired, also assign each upload to a specific student number, which allows it to be easily tracked later. Submitted files are first scanned by VERACISM endpoints for malware, then encrypted within VERACISM, by the platform managed encryption/decryption keys. Scan verdict and detailed audit logs (uploaders, date scanned) are only written to the database if the file has been confirmed clean; if the file is not confirmed as clean, its encrypted form is written to Lighthouse storage along with the metadata (filename, size, owner, student tag), cryptographic SHA-256 hash, and scan verdict.

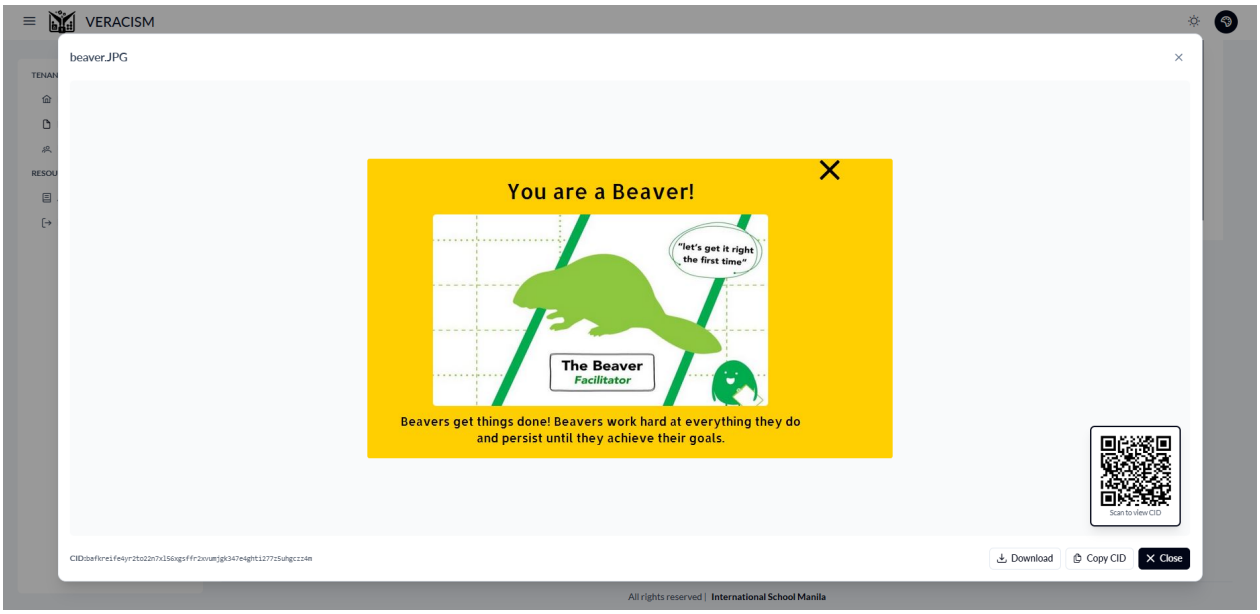

Figure 35. Tenants - File Viewer

When a tenant chooses View on a specific file, Veracism does not stream the raw object directly from Lighthouse. Instead, the application first retrieves the encrypted blob from Lighthouse storage using a secure, signed request generated on the server. The blob is then decrypted inside Veracism using internal encryption keys that are never exposed to the tenant, Lighthouse, or any third party. Only after a successful decryption does the app render the file in a read-only viewer and show its CID, with options to download or copy the CID/QR.

Every object stored in Lighthouse is encrypted at rest with Veracism-managed keys, anyone who obtains the direct Lighthouse URL still sees only ciphertext and cannot reconstruct the original file. This completes an end-to-end protection flow in which:

1. Files are encrypted before being written to Lighthouse.
2. Encrypted objects are stored and replicated by Lighthouse without access to plaintext.
3. Only the Veracism backend can decrypt and serve a temporary, view-only version to authenticated tenants based on their access rights.

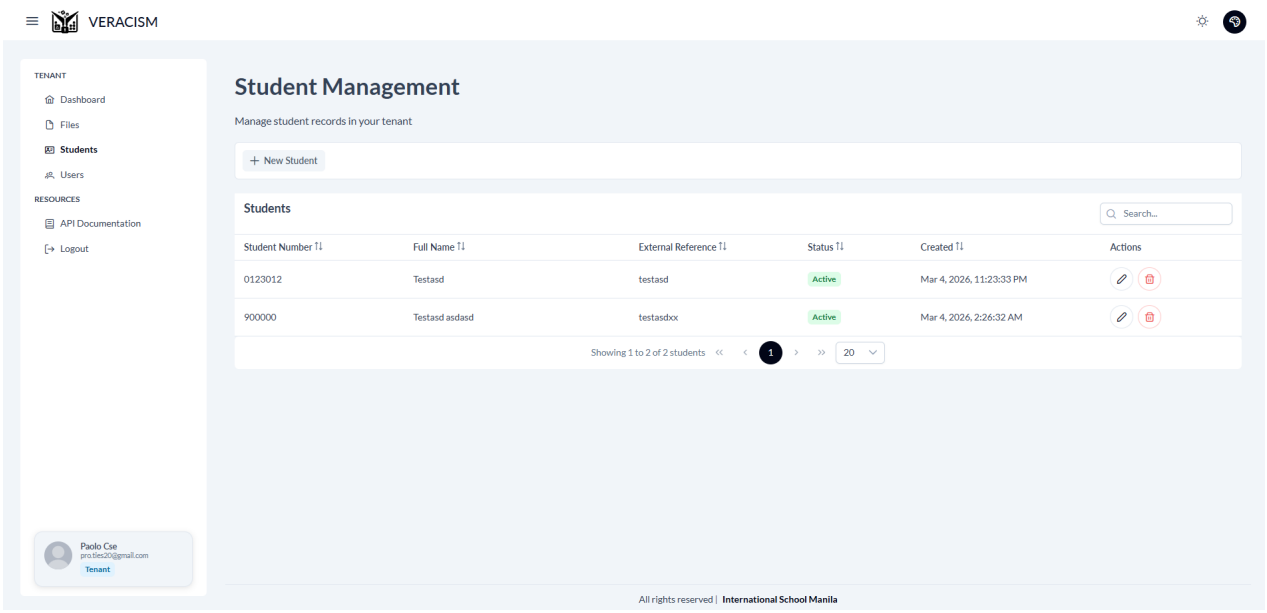

Figure 36. Tenants - Student Management

The Student Management page lets tenant users manage student records within their tenant. It allows them to add, view, search, and update student information.

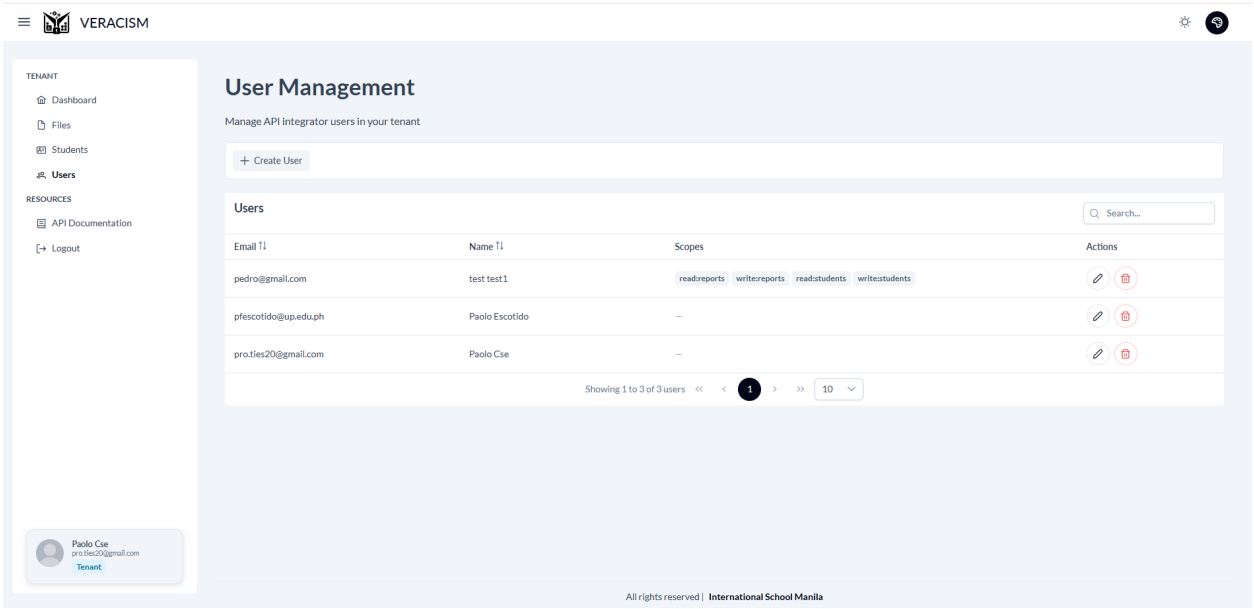

Figure 37. Tenants - User Management

Using Tenant User Management page, each organization has full control over who can access and integrate with Veracism under their tenant. Tenants can invite

more integrators from the user's email address, assign, change or remove roles (Admin, Member), add/remove details about the user and enable or disable the user account from a single view without losing the user's history. Admins can use the table to see the status and last login for each user, which can help them determine who is currently using the system. Administrators can also create and rotate API keys per tenant and per account, making each API call account specific, for better access control, revocation and auditability.

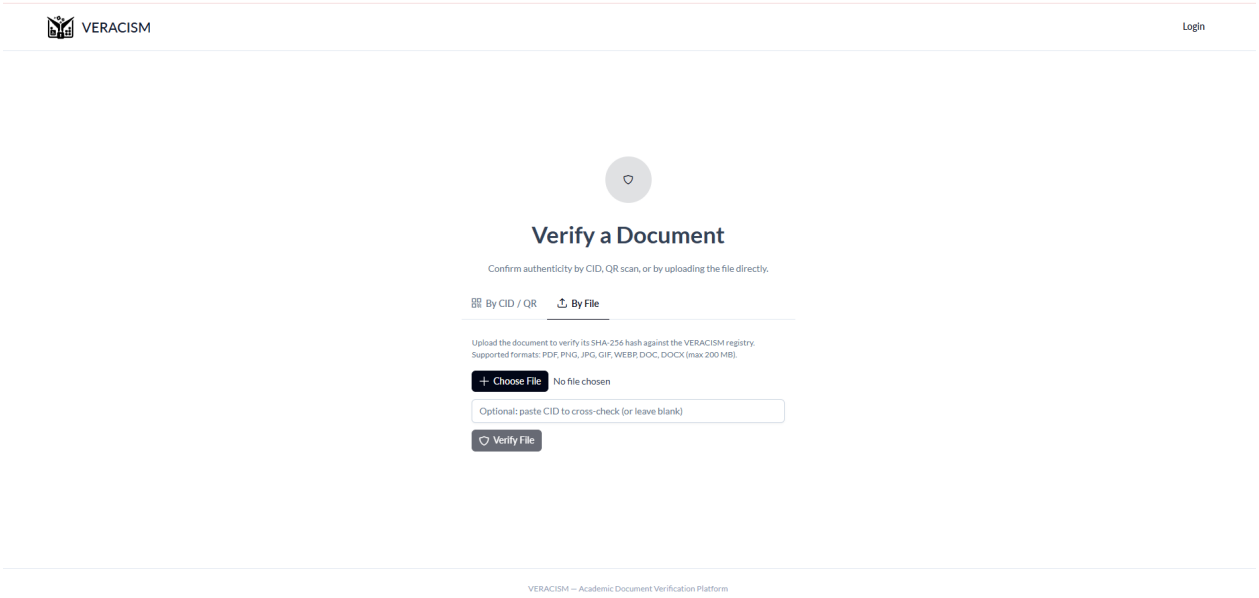

Figure 38. Public Verification Portal

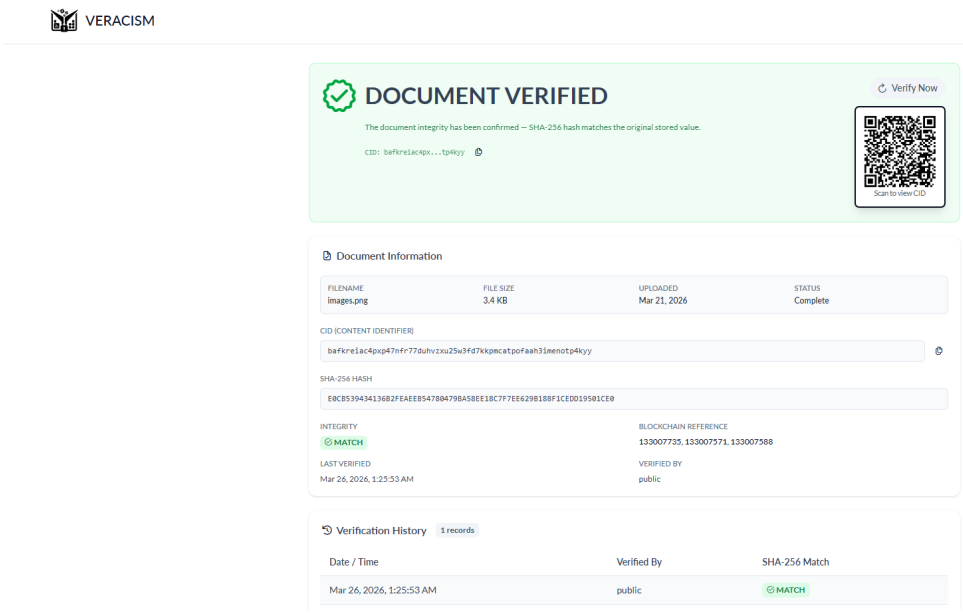

Figure 39. Public Verification Portal - Verified File

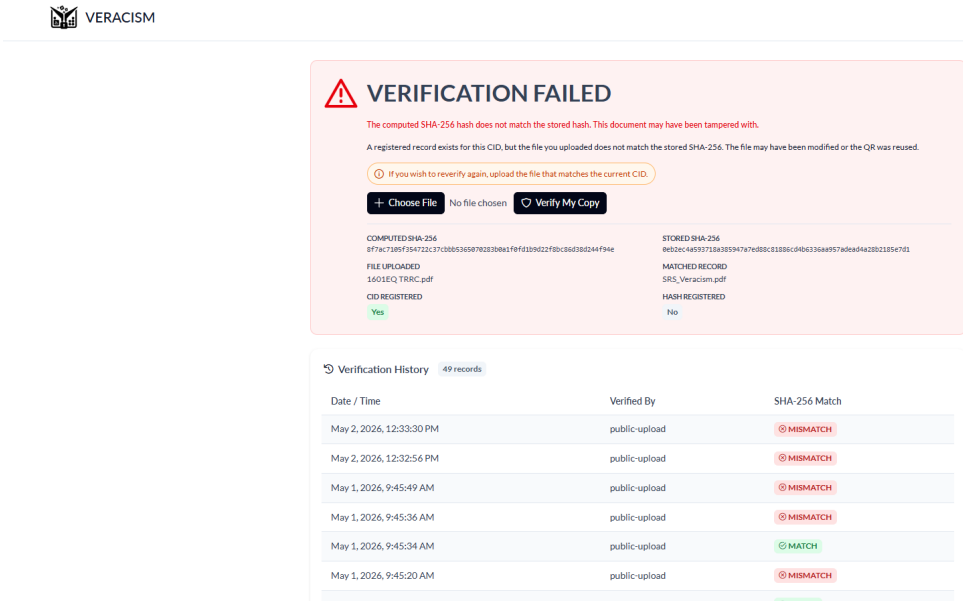

Figure 40. Public Verification Portal - Invalid File

The Public Verification Portal, shown in Figure 38, allows users to verify a document using its CID, QR code, or direct file upload. Veracism validates the document by checking the CID, recomputing the uploaded file's SHA-256 hash, and comparing it with the stored metadata registered during the original upload. If the CID, computed hash, and stored

metadata match, the portal displays a verified response as shown in Figure 39, confirming that the document is authentic and unchanged. If the CID is not found, the hash does not match, or the metadata differs from the registered record, the portal displays an invalid response as shown in Figure 40, indicating that the document may have been modified, is not registered, or cannot be trusted.

Project Assessment

This chapter outlines the evaluation of the VERACISM once it will be developed. The aim of the evaluation process is to assure that the system is operational, secure, usable, and deployable in the International School Manila (ISM) environment. In the proposal phase the emphasis is on the proposed approach to testing; the tests to be used; and the metrics to be gathered. These actual tests will produce test results only after implementation.

Functional Testing

The purpose of this testing was to ensure that the features that were implemented under VERACISM were functioning as they were designed to do so based on the system requirements. These testing procedures included validating uploaded files, checking for malware, hashing uploaded content, generating CIDs, downloading content with control, auditing and dashboard views, tenant-scoped access and public verification.

A **hybrid approach** of **automated black-box API testing** and **manual role-based functional testing**. Automated execution: Automated using Newman in Postman collections with the staging environment. These automated tests: tested the v1 public API workflow, some end-to-end user journeys, including IT Office upload, Registrar upload, student and external verification, protected download behaviour; and Cross-role upload to verify flow.

For the sake of manual functional testing, a few modules and flows that are exposed on the user interface needed manual testing, some Bearer JWT protected internal endpoints are involved, a few tests using QR code with a mobile device, a few reports on the dashboard, a couple of access checks followed by tenant scope and some gates managed by the administration program that needs role interaction.

Manual functional procedures were developed and followed for the System Administrator, IT Office / tenant user, Administration Office / Registrar, Student, External Auditor / Educator and for the end-to-end scenarios. These procedures included

tenant approval, user management, API key updates, configuration changes, audit log review/export, health monitoring, QR display and scan verification, dashboard status checking, report deletion or immutability checks, comprehension checks, access boundary checks across tenants, and multi-network verification checks.

The primary criteria used in functional testing included correct HTTP status codes, correct response structures, presence and format of expected fields, correct role and tenant access enforcement, correct UI behaviour where applicable, successful completion of required user journeys, and retention of the required audit evidence. Supporting automated and manual functional testing records will be retained as appendix evidence.

Functional test success criteria

To make the functional assessment explicit and reproducible, each test area is defined with measurable criteria for a *Pass* result. These criteria focus on correct outputs, correct security behaviour, correct logging, observable handling of failures, and coverage of both automated and manual functional modules.

Planned TC ID(s)	Method	Success criteria for pass	Target	Evidence	Result
TC-ITO-001, TC-REG-001	Automated	Valid authenticated uploads are accepted, stored in temporary storage, initial metadata is recorded, and a tracking identifier is returned.	Accepted upload behaves as specified	API logs, tracking ID, temp storage record	Pass / Fail
TC-ITO-002, TC-REG-002	Automated	Invalid, oversized, unsupported, or unauthorized uploads are rejected and do not create stored artifacts.	Rejected cases behave as specified	Error response, logs, absence of stored artifact	Pass / Fail
TC-ITO-003, TC-REG-003	Automated	Retrieved report metadata, student association, SHA-256 hash, and CID are consistent with the uploaded submission.	Stored values match submitted values	API response, metadata comparison, lookup result	Pass / Fail
TC-ITO-004, TC-REG-004, TC-STU-001, TC-EXT-001	Automated + Manual	Successful uploads produce a usable CID and verification path, and QR-based verification resolves correctly through the public portal.	CID present and QR resolves correctly	API response, UI screenshot, mobile scan proof	Pass / Fail
TC-REG-005, TC-ADM-004	Manual	Dashboard and audit views show the correct upload, anchoring, and audit information, and exports work in the required formats.	Statuses visible and export succeeds	UI screenshots, export file, audit review notes	Pass / Fail

Table 6. Functional test success criteria with automated and manual coverage (Part 1)

Planned TC ID(s)	Method	Success criteria for pass	Target	Evidence	Result
TC-ADM-001, TC-ADM-002, TC-ADM-003	Manual	Tenant approval, user and role management, API key management, and critical configuration changes can be completed through the admin module and reflect correctly in the system.	Admin actions succeed and are reflected correctly	UI screenshots, admin logs, review notes	Pass / Fail
TC-ADM-005, TC-ITO-006, TC-STU-005, TC-EXT-004	Manual + Automated	Restricted views and actions are accessible only to authorized roles. Unauthorized attempts are rejected, and tenant boundaries are enforced.	0 unauthorized privileged actions succeed	Access logs, route checks, API rejection results	Pass / Fail
TC-REG-006	Manual	Finalized report content cannot be casually edited through normal user paths, and any delete behaviour follows the intended soft-delete or policy-controlled workflow.	No improper content modification path exposed	UI inspection, code review note, behaviour check	Pass / Fail

Table 7. Functional test success criteria with automated and manual coverage (Part 2)

Planned TC ID(s)	Method	Success criteria for pass	Target	Evidence	Result
TC-STU-002, TC-STU-003, TC-EXT-002, TC-E2E-001, TC-E2E-002	Automated	Valid CID verification shows the correct result and invalid or fabricated CID values return a clear warning without internal error leakage.	Correct result and message shown for all cases	Verification response, screenshots, timing capture	Pass / Fail
TC-STU-004, TC-EXT-003, TC-E2E-003	Manual / Hybrid	Non-technical users can understand the verification result, repeated checks from different networks remain consistent, and unavailable content cases show clear warnings without exposing sensitive details.	Result understandable and behaviour consistent	Observation notes, network comparison notes, warning screen proof	Pass / Fail

Table 8. Functional test success criteria with automated and manual coverage (Part 3)

Functional test result classification

For functional testing, each planned test case will be classified using one of the following outcome labels.

Outcome label	Definition
Pass	All expected behaviours, outputs, access controls, and required evidence are observed, and the result matches the stated success criteria.
Fail	One or more expected behaviours, outputs, controls, or validations are not met, or incorrect behaviour is observed during execution.
Partial	The feature is only partly implemented, the flow is only partly testable, or the observed behaviour satisfies only part of the expected result.
Not Implemented	The required feature, control, or supporting logic is not yet available in the current build, so the planned test case cannot be fully executed.

Table 9. Functional test result classification

Accordingly, the functional testing tables define the conditions required for a *Pass*, while actual execution results are recorded separately using the classification labels in Table 9.

Integration Testing

Integration testing focused on verifying that the major VERACISM components worked correctly when combined into a complete operational pipeline. The integration scope covered the interaction between the .NET 8 backend API, SQL Server, ClamAV, YARA, Lighthouse/IPFS storage, and the public verification process.

The integration assessment was performed through automated API-based tests in the staging environment. These tests targeted the public API v1 endpoints through API key authentication and covered the main integrated operations of the platform, including upload and scan processing, report retrieval by CID, public verification, controlled file download, and storage-status checking. Both valid and invalid cases were included to assess API contract behaviour, error handling, response consistency, and subsystem coordination.

The integration design also included a test for valid uploads to go to the scan, hash, metadata and storage points and invalid or unauthorized requests being blocked and not creating unnecessary records. More focus was put on tenant isolation, consistency of uploaded metadata retrieved metadata, and public versus protected endpoint behaviour.

The primary criteria used in integration testing included end-to-end request completion, correctness of inter-service outputs, validation of report and CID data across steps, correct rejection of invalid requests, and preservation of system boundaries across tenants and access levels.

Performance Testing

The customer was interested in testing of the VERACISM responsiveness and stability against the expected verification traffic in the Staging environment. The public CID verification endpoint was focused for this as this is a key aspect of how the platform

is used by students, external verifiers and partner institutions needing to make a rapid integrity check.

The performance run was performed on Postman Performance Testing and connected to Postman API staging environment, which is the VERACISM API. A total of 100 virtual users were used for 5 minutes in the test profile. The test scenario was aimed at the genuine public verification request with the capture of the request volume, throughput, response time, percentile response times and captured transport level errors during the run.

This assessment criteria covered the following: (1) Minimum error rate, (2) Stable throughput under concurrent traffic, (3) Average response time (within acceptable limits), and (4) Endpoint availability (sustained throughout the test window) during the test session. The metrics captured are stored as performance evidence and will be compared to and discussed with in relation to other captured metrics in the results chapter.

User Acceptance Testing

User acceptance testing will focus on the primary users of VERACISM: tenant administrators, system administrators, IT integrators and external verifiers through the public portal. The test scenarios will come from the essential functional flows and capabilities of the system, such as dashboard review, report and file management, tenant administration, setting the security configuration, reviewing the audit-log, student and integrator administration, public verification, and API-based integration workflows.

User Acceptance Assessment will be conducted with task-based scripts based on system Acceptance criterion. Each UAT case will outline the role of the participant, any related planned test case(s), the task the participant was required to perform, and the outcome and expected result relevant to that task. The participants will carry out the tasks set, and observers will document whether or not the tasks were completed, how long they took, any defects found and the task blocker observed during the carrying out of the tasks. Results will be recorded using objective completion outcomes such as *Passed*, *Partially Completed*, or *Failed*, supported by notes and evidence captured during the session.

Planned user group	Target #	Representative test tasks
System Administrator (Super Admin)	1–2	Review dashboard analytics and service health; manage reports and tenant records; configure YARA and ClamAV settings; manage API keys; review and export audit logs.
Tenant Administrator (Administration Office / ISM offices)	2-3	Review dashboard summaries; upload and manage files; review version history; copy CID and download files; manage students; manage integrator users and API keys.
IT Integrators / Developers	2-3	Submit reports through the API; retrieve report details; download report files; check file processing or status results; handle invalid or rejected API requests correctly.
External verifiers (students, auditors, public user)	4–5	Verify records through QR code or CID entry using the public portal and confirm that valid, invalid, tampered, or unavailable cases display the correct result.

Table 10. Planned user-testing participants and task coverage

User acceptance success criteria

To make the user acceptance assessment explicit and reproducible, each UAT case is defined with measurable criteria for a *Passed* result. These criteria focus on successful task completion, achievement of the expected result, absence of blocking defects, and the presence of sufficient supporting evidence for the recorded outcome.

UAT ID	Linked TC ID(s)	Task	Success criteria for passed result	Evidence	Result
UAT-01	TC-ADM-001, TC-ADM-002, TC-ADM-004	Review administrator dashboard and core records.	Dashboard data, recent records, and report information are visible and can be reviewed without access or display errors.	Screenshots, viewed records	Passed / Partially Completed / Failed
UAT-02	TC-ADM-001, TC-ADM-002, TC-ADM-003	Manage tenants, users, and API keys.	Tenant records, user assignments, and API key actions are completed successfully and reflected correctly in the interface and audit trail.	Screenshots, audit entries	Passed / Partially Completed / Failed
UAT-03	TC-ADM-003, TC-ADM-004, TC-ADM-005	Manage security and monitoring settings.	YARA and ClamAV related actions complete successfully, system health is visible, and audit logs can be reviewed or exported as intended.	Screenshots, logs, export output	Passed / Partially Completed / Failed

Table 11. User acceptance success criteria for System Administrator

UAT ID	Linked TC ID(s)	Task	Success criteria for passed result	Evidence	Result
UAT-04	TC-REG-001, TC-REG-003, TC-REG-004, TC-REG-005	Upload and manage report files.	Valid file operations complete successfully, reports are associated correctly, CID and QR data are available, and file records appear with the correct status.	Upload output, screenshots, resulting record	Passed / Partially Completed / Failed
UAT-05	TC-REG-002, TC-REG-006	Handle invalid file actions and restricted modification paths.	Invalid or rejected actions show clear system responses, and no improper content modification path is exposed through normal user actions.	Screenshots, error output	Passed / Partially Completed / Failed
UAT-06	TC-REG-003, TC-REG-005	Manage related student or integrator records.	Student or integrator information can be viewed and managed correctly, and related records remain visible and consistent in the interface.	Screenshots, updated records	Passed / Partially Completed / Failed

Table 12. User acceptance success criteria for Tenant Administrator

UAT ID	Linked TC ID(s)	Task	Success criteria for passed result	Evidence	Result
UAT-07	TC-ITO-001, TC-ITO-002, TC-ITO-003, TC-ITO-004, TC-ITO-005	Execute the intended API integration workflow.	Valid API requests return the expected identifiers, metadata, download, or status results, while invalid requests are rejected without creating an unintended record.	API responses, logs	Passed / Partially Completed / Failed

Table 13. User acceptance success criteria for IT Integrator / Developer

UAT ID	Linked TC ID(s)	Task	Success criteria for passed result	Evidence	Result
UAT-08	TC-STU-001, TC-STU-003, TC-EXT-001, TC-EXT-002, TC-E2E-002, TC-E2E-003	Verify a report through the public portal using QR code or CID entry.	The public portal displays the correct result for valid, invalid, tampered, or unavailable cases without requiring internal access and without exposing sensitive internal details.	Screenshots, scan proof, portal output	Passed / Partially Completed / Failed

Table 14. User acceptance success criteria for External Verifier

During user testing, each participant will receive a task script derived from the UAT table corresponding to the participant's assigned role. The script will describe the initial state, the assigned task, and the expected result. Observers will record task completion, time taken, defects encountered, and any task blockers observed during execution.

User acceptance testing will be considered satisfactory when assigned tasks are completed successfully, expected results are observed, critical defects are resolved, and regression re-testing confirms that fixes do not introduce new failures. At least 90% of critical UAT tasks should be completed without assistance for each user group before acceptance is concluded.

User acceptance result classification

For user acceptance testing, each scenario will be recorded using one of the following outcome labels.

Outcome label	Definition
Passed	The participant completes the assigned task successfully and the expected result is observed.
Partially Completed	The participant completes part of the assigned task, but encounters minor defects, incomplete behaviour, or non-blocking issues during execution.
Failed	The participant cannot complete the assigned task, or the observed result does not satisfy the expected result for the intended role.

Table 15. User acceptance test result classification

These outcome labels will be recorded together with task evidence, and any defects identified during the session.

Security Testing

Security evaluation for VERACISM will focus on protecting the confidentiality, integrity, and availability of data in line with ISM security policies. In particular, the system is expected to meet the following security targets:

- All network traffic uses HTTPS with current TLS versions;
- Passwords and secrets are never logged or exposed in client responses;
- Input validation and output encoding are applied to all external interfaces;
- The system is regularly scanned for vulnerabilities and no critical or high severity findings remain unresolved beyond agreed timelines.

VERACISM will be subjected to Vulnerability Assessment and Penetration Testing (VAPT) before the initial go-live and after any significant change that may affect its security posture. The target of the assessment is the VERACISM web application and APIs, reachable under the staging URL `https://sb-[update-domain-if-needed]` (to be confirmed). Test credentials will be provisioned securely (for example, via encrypted email

or a password manager). Systems, subdomains, and third-party services not explicitly included in the scope will remain out of scope.

Planned security activity	**Tools (examples)**	**Objective / scope**
Automated vulnerability scanning	Burpsuite	Discover CVEs, open ports and services, misconfigurations, weak defaults, outdated components, and common web issues such as XSS, SQL injection, CSRF, and insecure headers.
Dynamic application scan (DAST)	Burpsuite	Scan the VERACISM web application and APIs (login, upload, verification portal, internal console) under authenticated and unauthenticated sessions.
Manual verification of findings	Manual review using browser tools and scripts	Remove false positives from automated scans and assess real impact and exploitability of each issue in the context of VERACISM.
Manual penetration testing	Custom payloads and test harnesses aligned with OWASP Top 10	Perform targeted testing of high-risk flows and endpoints, based on the focus areas in Table 17.
Continuous dependency and image scanning (post-deployment)	Snyk or an ISM-approved equivalent integrated in CI/CD	Detect vulnerable libraries, container images, and IaC templates over time and feed findings into ISM's vulnerability management process.

Table 16. Planned security testing activities for VERACISM

OWASP ID	Risk name	Planned focus in VERACISM
A01	Broken Access Control	Verify RBAC/ABAC enforcement, least privilege for service accounts, and strict access restrictions on reports, admin pages, and APIs.
A02	Cryptographic Failures	Check HTTPS/TLS configuration, key and secret handling, and correct usage of hashing (SHA-256) and encryption where required.
A03	Injection	Confirm use of parameterized queries and ORM; test all inputs for SQL, command, and template injection.
A04	Insecure Design	Review threat model, design of immutability and auditability, and presence of required security controls in high-risk flows.
A05	Security Misconfiguration	Test for insecure defaults, missing security headers, verbose error messages, open ports, and weak configuration in .NET, MSSQL, and hosting.
A06	Vulnerable and Outdated Components	Confirm that dependency scans are running and that findings in .NET, Angular/React, and IPFS/Filecoin libraries are addressed.
A07	Identification and Authentication Failures	Test login flows, token handling, session timeout, brute-force resistance, and integration with the ISM identity provider.
A08	Software and Data Integrity Failures	Validate CI/CD controls, integrity of deployed artifacts, and that report hashes and CIDs reliably detect unauthorized changes.
A09	Security Logging and Monitoring Failures	Confirm that authentication, access control, upload, download, and verification events are logged and integrated with monitoring/alerting.
A10	Server-Side Request Forgery (SSRF)	Test outbound calls (Lighthouse, IPFS gateways, explorers) against allow-lists and ensure no user-controlled URLs are used.

Table 17. OWASP Top 10 focus areas for VERACISM penetration testing

The overall VAPT procedure for VERACISM will follow these steps:

1. **Information gathering:** Gather information on infrastructure, technology stack, exposed services, and endpoints; Establish an attack surface for web application and APIs.

2. **Automated vulnerability scanning:** Perform network and application scans by using the tools in Table 16. The aim is to recognize:

   - known systems and libraries with recent CVEs;
   - open ports and exposed services;
   - misconfigurations and weak defaults;
   - outdated software components;
   - exposure in the web due to XSS, SQL injection, CSRF, missing or insecure HTTP headers.

3. **Manual verification of findings:** Check the results of the automation process manually, in order to eliminate false positives and to gain insight into the functionality of each error in a running VERACISM instance.

4. **Manual security testing:** Conduct regionally specific testing of affected flows and endpoints with safe, non-destructive payloads. Take screenshots and logs of any evidence for each confirmed find, record severity, and note reproduction steps, and identify payloads.

5. **Penetration testing focus areas:** A Use the OWASP Top 10 mean areas outlined in Table 17 to test each of the areas, including access control, crypto, input handling, configuration, components, logging, and outbound calls, thoroughly.

6. **Remediation and re-testing:** Deliver comprehensive, technical conclusions and remediation tips for the development and ops teams. Re-test following fixes to ensure vulnerabilities addressed and remediation did not create new vulnerabilities.

7. **Reporting:** Generate a formal VAPT report that outlines all of the verified vulnerabilities, their severity and potential impact, reproduction and suggested remediation actions. Document and retain reports and re-test results for audits and reevaluation.

Optionally, security assessment can be conducted along with load testing performed using Azure Load Testing or other load testing tools. In this mode, realistic user load to be applied to VERACISM while observing security controls, rate limiting, throttling and brute forced protection under stress.

Continuous monitoring of VERACISM is recommended by using dependency and image scanning tools to help maintain the security of VERACISM over time. The results of these multi-sensors will be managed as part of ISM's vulnerability management process.

Security acceptance criteria for the project are:

- All Critical and High vulnerabilities identified in VAPT or automated scans must be fixed or formally risk-accepted by ISM Information Security before production release;
- Medium and Low issues must have a documented remediation plan and target date;
- VAPT reports, scan results, and re-test outcomes must be retained as evidence for audits and internal reviews.

RESULTS AND DISCUSSION

This section introduces and explains the results of the five testing categories that have been performed to assess the VERACISM platform: (1) the functional testing, (2) the integration test, (3) the performance testing, (4) the security testing, and (5) the user acceptance testing.

The appendices show the raw data for each category completely. All the assertion-level tables are detailed in the integration tests and functional tests Appendices 0.9 and 0.10. The security Appendix 0.11 includes security findings. Performance configuration is part of the performance Appendix 0.12. And the functional through security remediation PDF appendices contain full original test report PDFs.

Functional and Integration Testing

The hybrid method outlined in the Project Assessment chapter – automated black box API testing using Newman and manual role based functional verification – was used for functional and integration testing. The two streams shared the same scenario set, with those scenarios scripted as automation steps and others performed manually (for their dependency on business logic and/or user interface considerations).

The automated test summary

The automated Integration Test Suite consisted of five test groups that interact with the external VERACISM API v1: scan upload, report retrieval by CID, public verification, controlled download and storage status check. Twenty two (22) test cases with a total of 93 individual assertions were run against the staging environment in `https://sb-api.ismanila.org/Veracism`. Four (4) user role groups were covered in 19 automated tests defined by 89 assertions in the Functional Test Suite. Everyone received a pass mark of 100% for both suites and there were no failures or errors.

Table 18 shows the combined statistics.

Suite	Test Cases	Assertions	Passed	Failed	Pass Rate
Integration Test Suite	22	93	93	0	100.0%
Functional Test Suite	19	89	89	0	100.0%
Combined Total	**41**	**182**	**182**	**0**	**100.0%**

Table 18. Combined automated test results: Integration and Functional suites (30–31 March 2026)

Integration suite by endpoint group

Table 19 represents the 93 integration assertions run in the five endpoint groups, along with the overall run time of the Newman runner. Most of the elapsed time is spent in the Scan Upload group, representing 83.7% of the total run time of 83.7 seconds; each of these “scan upload” calls triggers the whole document pipe: SHA-256 computation, ClamAV scanning, YARA scanning, Lighthouse encryption and IPFS anchoring.

Endpoint Group	TCs	Assertions	Pass	Fail	Time (s)	% of Run
Scan Upload	8	37	37	0	77.969	93.1%
Get Report by CID	4	17	17	0	0.307	0.4%
Verify (Public)	4	18	18	0	0.230	0.3%
Download	3	12	12	0	6.014	7.2%
Storage Status	3	11	11	0	0.163	0.2%
TOTAL	**22**	**93**	**93**	**0**	**83.728**	**100%**

Table 19. Grouped by endpoints, the following is the result of running the integration test suite and providing full assertion details in Appendix 0.9)

The breakdown brings up an important aspect, the page load time – every time the metadata is retrieved from the file (which also includes a lookup for its CID with 0.147

seconds – still in interactive times, despite sending the message cross-network), anyone is able to verify its currency (0.066 seconds – again, within interactive expectation), and one can check if it is stored or not (0.061 seconds – still within interactive expectations). The upload pipeline is inherently known to be capped by the round trip to the Lighthouse encryption proxy and the Filecoin network - in this case of 15-21 seconds per file, since the round trip can't be avoided. This is around the normal latency of file anchoring operations with Lighthouse API.

Functional suite by automated group

Table 20 lists the 89 functional assertions for four of the Newman runner groups: F1, IT Office API; F2, Registrar Upload; F3, Public Verification; F4, End-to-End Journey. The total time spent on automated files was **116.641 seconds**.

Feature Group	Suites	Assertions	Pass	Fail	Time (s)	% of Run
F1: IT Office API	5	27	27	0	54.231	46.5%
F2: Registrar Upload	4	20	20	0	39.508	33.9%
F3: Public Verification	7	28	28	0	21.047	18.0%
F4: End-to-End Journey	3	14	14	0	1.855	1.6%
TOTAL	**19**	**89**	**89**	**0**	**116.641**	**100%**

Table 20. Functional test suite automated results by feature group (31 March 2026)

Manual and code review results – per role

In total, there are 30 test cases (automated API, manual code review and observational inspection) across all five user roles in the system specification. The pass, partial and fail split for each role group is shown in Table 21

Role Group	Total	PASSED	PARTIAL	FAILED	N/T	Pass %
System Administrator (TC-ADM)	5	5	0	0	0	100.0%
IT Office Staff (TC-ITO)	6	4	1	1	0	66.7%
Registrar / Admin Office (TC-REG)	7	7	0	0	0	100.0%
Student (TC-STU)	5	5	0	0	0	100.0%
External Auditor / Educator (TC-EXT)	4	4	0	0	0	100.0%
Cross-Role End-to-End (TC-E2E)	3	2	1	0	0	66.7%
OVERALL	**30**	**27**	**2**	**1**	**0**	**90.0%**

Table 21. Functional test results by user role (N/T = not tested in this cycle; full detail in Appendix B, Tables 34 through 39)

A **100% pass** with no failures and partial results was seen by three out of four groups that play the most critical role in the value proposition of the system (System Administrator, Registrar/Admin Office, External Auditor). The single failure (TC-ITO-005) and one partial result (TC-ITO-006) produced the lowest rates for the IT Office group, 66.7%.

Open defects summary

The Manual and code review phase was able to surface three defects. Details regarding each defect, its severity and disposition are summarized in Table 22.

TC ID	Severity	Title	Description	Disposition
TC-ITO-005	Medium	No Lighthouse retry policy	No Polly retry or circuit breaker on `LighthouseStorageService`; transient unavailability causes immediate failure with no backoff	Future Work
TC-ITO-006	Low	Scoped audit log UI incomplete	`AuditEvent` table has no C# model, repository, or controller; tenant scoped audit UI cannot be rendered	Future Work
TC-E2E-003	Medium	VerificationLog not written	`VerificationLog` table exists in schema but `VerifyController` does not persist a log entry on each `GET /verify` call	Future Work

Table 22. Open defects after functional testing: All of them are to be prioritized for upcoming development sprint

Key functional findings

The following system behaviours were confirmed by the combined testing programme:

1. **Core pipeline is fully end to end functional.** For TC-E2E-001, the threat scan is successful after completely uploading a file with verified attempt from unauthenticated public verification, `sha256Match: true` and `inLighthouse: true`.

2. **Multi-engine threat detection gates work properly.** A JPG with a webshell signature is detected through the usage of the YARA engine embedded in the rule: `image_webshell_signature` and cannot be uploaded to Lighthouse, whilst ClamAV says it is clean, as shown above, this multi engine approach proves useful.

3. **The SHA-256 integrity is maintained end to end.** The hash computed at upload (TC-ITO-001) is retrievable via the report endpoint (TC-ITO-003) and confirmed by

the public verification endpoint (TC-E2E-001), covering the full document lifecycle.

4. **An enumeration is not possible because tenant data isolation.** TC-11 established that a CID of a different leaseholder will return a 404 response if an approved attacker tries to access a cross tenant CID he can conclude that documents does not exist, not that they are prohibited.

5. **The authentication and authorisation boundaries are proper.** All protected endpoints returned bad HTTP 401/403 responses for requests with invalid or missing tokens. The public verification endpoint returned HTTP 200 without any credentials, TC-EXT-001 and TC-15.

6. **Document immutability is enforced at the API level.** No PUT or DELETE endpoints exist for report records (TC-REG-006); the API is append only by design, matching the tamper evident guarantee.

7. **Tampered-document detection is reliable.** A fabricated CID (TC-E2E-002) returned HTTP 404 with a descriptive message and no false positive verification result.

Performance Testing

Performance testing was conducted using Postman's Performance Testing feature targeting the public CID verification endpoint (`GET /api/v1/verify?cid={cid}`), which is the endpoint most exposed to external concurrent traffic in the production workload. Two rounds were executed, differing only in whether the Nginx reverse proxy rate limiting policy was active.

Test configuration and scope

Both rounds used 100 virtual users (VUs) sustained for a 5-minute (300-second) window, meaning the test engine attempted to drive **up to 30,000 requests** during each run at sustained concurrency. Table 23 summarises the configuration of both rounds.

Parameter	**Version 1.0 (2026-03-28)**	**Version 1.1 (2026-03-30)**
Target endpoint	`GET /api/v1/verify?cid={cid}`	
Virtual users (VUs)	100	100
Test duration	5 minutes	5 minutes
Infrastructure rate limiting	**Enabled** (Nginx default)	**Disabled** (rate-limit rules removed)
Error source identified	HTTP 429 Too Many Requests from Nginx proxy	No infrastructure throttle
Application layer result	Not reachable under sustained load	Successfully handled concurrent load

Table 23. Performance test configuration and run conditions

Key performance findings

Table 24 presents the principal observations and their implications for production deployment.

Metric / Observation	**Version 1.0 Result**	**Version 1.1 Result**	**Root Cause**
Error rate	**Significantly elevated**; dominant error was HTTP 429 (rate-limit exceeded) returned by Nginx before requests reached the backend	**Significantly reduced**; errors largely eliminated once Nginx throttle removed	Nginx rate-limit threshold set below 100-VU load level
Average response time	Higher; most latency was queuing delay at the proxy level, not application processing time	Lower; requests reached the .NET 8 backend directly with minimal queuing overhead	Infrastructure configuration, not application code
Throughput (req/s)	Constrained by HTTP 429 rejection at proxy; effective throughput well below 100-VU potential	Higher and stable; application layer did not become a bottleneck during the 5-minute run	Rate-limit rules
Application layer	Healthy; .NET 8 backend and SQL Server correctly processed all requests that passed the proxy	Confirmed healthy under 100-VU sustained concurrent load	Application architecture is sound
Key implication	Rate-limiting thresholds must be re-tuned before go-live to reflect legitimate concurrent usage patterns	Baseline confirmed: application can sustain 100-VU load; tune rate-limiter, do not remove it	Production config action required

Table 24. Performance test key findings across Version 1.0 and Version 1.1

Version 1.0 of the performance test revealed that the Nginx reverse proxy rate limiting policy had been configured at a threshold below what 100 simultaneous virtual users required, causing the proxy to return HTTP 429 (Too Many Requests) before requests ever reached the .NET 8 application server. This was confirmed by disabling the rate limiting rules for Version 1.1, after which the error rate dropped substantially.

This finding is an important operational significance: If the rate limiting layer is always removed, this means that we cannot take an important security measure for granted: blocking denial of service and credential stuffing attacks. Instead, it's important to set the thresholds to allow for the anticipated max concurrent users and same time, block abnormal amounts of request rates. The performance testing exercise itself highlighted the benefits of performing this testing with realistic levels of concurrency, and uncovered a configuration issue that would have impacted user engagement during run time and would not have been found during functional or security testing.

The performance PDF appendices, Appendix Version 1.0 and Appendix Version 1.1, contain a copy of full numeric metrics (such as request counts, P50/P95 response time percentiles, throughput graphs and per second error-rate charts).

User Acceptance Testing

User Acceptance Testing (UAT) was carried out for a specific task for each individual role-scoped user from a structured Google Form (*Veracism UAT Form.pdf*). Specific UAT items were assigned to a certain role, and blank columns are provided for roles that were not assigned. Unassigned columns are included only to allow for the scoring of columns that have been assigned and are not intentional. The UAT cycle included participation from four different roles: System Administrator, Tenant Administrator, IT Integrator / Developer, and External Verifier.

UAT participants and scope

Participants, roles and anticipated activities during UAT are summarized in Table 25.

Role	Participants	UAT Cases Assigned	Evaluations Recorded
System Administrator	1	UAT-01, UAT-02, UAT-03	3
Tenant Administrator	2	UAT-04, UAT-05, UAT-06	6
IT Integrator / Developer	2	UAT-07	2
External Verifier	5	UAT-08	5
TOTAL	**10**	**8 unique UAT IDs**	**16**

Table 25. UAT participant distribution by role and assigned task scope

UAT results by role

Table 26 presents the pass, partial, and fail breakdown per role group across all recorded evaluations.

Role	Evaluations	Passed	Partial	Failed	Pass %
System Administrator	3	2	1	0	66.7%
Tenant Administrator	6	6	0	0	100.0%
IT Integrator / Developer	2	2	0	0	100.0%
External Verifier	5	5	0	0	100.0%
OVERALL	**16**	**15**	**1**	**0**	**93.8%**

Table 26. UAT results by participant role (role-scoped; blank columns in unassigned roles excluded)

Three roles – Tenant Administrator, IT Integrator, and External Verifier – achieved a **100% pass rate** across all their assigned evaluations. The System Administrator role recorded the only partial result (UAT-01). The participant confirmed that all dashboard summary statistics loaded successfully without error, but noted the following usability suggestion:

“Summary of information that can be seen in the system were loaded successfully without any error. Should include redirect links for every summary (more details or view full report).” This feedback is captured as a low impact usability Reflections and Enhance and will be made available in the next development cycle when navigation links will be provided to users from each dashboard summary card that link to the detail or report view. Both User and configuration management (UAT-02 and UAT-03) passed completely.

Key UAT findings

The following behaviours were shared by the participant responses:

1. **The services of verifying public documents are available to everyone.** All five External Verifier participants, UAT-08, indicated that document verification using CID is working without register, and one said it was “Excellent” in comments.

2. **Easily accepted 2 Tenant upload and file management workflows.** The Tenant Administrator participants all reported passing UAT-04, UAT-05, and UAT-06. Tenants records, user assignments and API key configuration were properly updated in the UI and audit log.

3. **IT Integrator API access is working.** The two IT Integrator participants confirmed that UAT-07 passed, concluding that the integration of the authenticated API upload and retrieval of metadata are as expected by programmatic consumers.

4. **Administrator dashboard needs to be improved in usability.** The participant (the System Administrator) completed part of UAT-01. Every data presented showed no error and the participant gave the following constructive suggestion: “Should include redirect links for all summaries, should provide more details and/or present whole report.” This targeted feedback will be leveraged in the subsequent development iteration by providing navigation links from each dashboard summary card to the page or detail view of the specific card.

5. **All 8 UAT IDs received at least one response.** All tasks defined across all roles

has been addressed within participant cohort of 10, achieving full scenario coverage in the UAT cycle.

Security Testing

There was a first pass for security evaluation followed by a security evaluation pass where targeted manual code review and penetration testing were conducted for every security flagged found in the initial security evaluation pass – The first pass was performed using Burp Suite Community Edition – DAST was used.

Initial DAST scan overview

The findings returned by Burp Suite scan for the staging endpoint were at various severities. Each category of finding and post-review disposition of the findings are summarized in Table 27.

Finding Type	**DAST Sev.**	**Count**	**Confidence**	**Post-Review Disposition**
SQL Injection	High	4	Tentative	**False positive**: EF Core ORM; proxy returned HTTP 403 for malformed paths
Reflected XSS	Medium	1	Tentative	**False positive**: Angular output encoding absorbed injected payloads
Missing security headers	Low	Several	Certain	Partial: HSTS, X-Frame-Options present; cache-control absent on some endpoints
Backup file disclosure	Informational	Multiple	Certain	No risk: all backup URL patterns 403-gated at IIS; no accessible files
Email address disclosure	Informational	Multiple	Certain	By design: user email in authenticated admin API responses
Cacheable HTTPS responses	Informational	Multiple	Certain	Low risk: affects read-only public endpoints only

Table 27. Burp DAST initial findings and post review disposition – 21 March 2026.

The reflected XSS finding, as well as all four SQL Injection findings was judged to be false alerts on manual code review. No handwritten SQL anywhere in the back-end codebase, it relies on a single line of code query to Entity Framework Core object graph created using LINQ. However it was not the application running behind the scanner that mis-handled the SQL data on the input to it, but it was responses from the IIS/Nginx proxy to malformed

URL paths that triggered the scanner's SQL-injection heuristics.

Listed vulnerabilities and remediation

Five real vulnerabilities prior to graded deployments were identified and confirmed by manual code review and target penetration testing, all Medium-High or lower. Before the deployment date all five of the following were remediated. The complete summary is included in the Table 28 which includes the remediation technique used to meet a finding.

ID	Vulnerability	Severity	Remediation Applied	Status
VUL-01	Unbounded API Client Data Response	Low	Server-side pagination (`OFFSET/FETCH`); configurable page/pageSize; max cap of 100/page; response envelope updated	Remediated
VUL-02	BOLA on API Client Records (cross tenant read)	Medium–High	Explicit tenant ownership check on GET/PUT/PATCH/DELETE; HTTP 403 for cross tenant reads; admin retains cross tenant access	Remediated
VUL-03	Insufficient File Type Validation	Low–Medium	Allowlist of 18 permitted extensions via `FileValidationHelper`; applied before scanning pipeline; frontend `accept` attribute updated	Remediated
VUL-04	BOLA on Report Records (cross tenant report access)	Medium–High	Tenant ownership check on `GET /reports/{id}`, `/latest`, and `/user/{userId}`; HTTP 403 for non-owner callers	Remediated
VUL-05	Missing Tenant Isolation on File View/Download	Medium–High	`EnforceTenantOwnershipAsync(cid)` helper in `VerifyController`; HTTP 403 for non-owner; admin pass-through retained	Remediated

Table 28. Confirmed security vulnerabilities and remediation status

Three of the five confirmed vulnerabilities (VUL-02, VUL-04, VUL-05) were classified as 'Broken Object-Level Authorisation (BOLA)'. The African American reason funds are protected behind BOLA are sensitive academic papers, transcripts, credentials and institutional identifiers, whose confidentiality is a major platform guarantee. At the controller level, there are tenant scoped ownership checks performed across all three BOLA paths, which have also closed.

Security posture summary

Table 29 presents a one-page assessment of the platform's security posture across key dimensions at the time of initial controlled deployment.

Security Dimension	Assessment	Notes
Critical / High vulnerabilities	**None**	No confirmed High or Critical findings at deployment
Medium–High vulnerabilities (BOLA)	**Remediated**	VUL-02, VUL-04, VUL-05: all three BOLA paths closed
SQL injection attack surface	**Eliminated**	EF Core ORM with parameterised queries throughout; no handwritten SQL
Authentication (Identity Provider)	**Strong**	Auth0 JWT Bearer tokens; `ValidateUserExistsFilter` syncs users into local DB
Authorisation model	**Adequate**	`AdminOnly`, `TenantOnly`, `AdminOrTenant` policies; inline controller checks
Transport security	**Strong**	HTTPS-only; HSTS headers enforced via Nginx
File upload safety	**Good**	ClamAV + YARA + PDF phishing analysis; file-type allowlist (18 extensions)
Audit trail	**Partial**	Admin audit export implemented; tenant scoped audit UI deferred (DEF-004)
Deferred findings	**2 items**	DEF-01 (partial BOLA on user enumeration) and DEF-02 (nullable warnings); neither is immediately exploitable

Table 29. There is a security posture assessment for the initial deployment, where this take place, last March of 2026.

Overall Discussion

Overall, the five streams of testing show a uniform picture of a system that meets its basic functional goals and infrastructure-level configuration issues that can be fixed prior to going to production scale-out locations.

The **100% automated assertion pass rate** throughout the 182 assertions confirms that simply uploading your content, scanning for threats, then storing the content in a decentralised way and then later uploading a public CID based verification also works the way it should. The 30-case manual suite failed merely three times (with an overall 90.0% full-pass rate), representing gaps in non-core subsystems, retry logic, audit tooling, and verified document verification logging, which do not impact the integrity or confidentiality of verified documents.

There was a real operational worry, the rate limiting configuration configuration that the performance test exercise brought to light, which would have had a negative impact on the user experience if it had not been identified during the performance test. The v1.1 test verified that the application layer can handle up to 100-VU concurrent load, and that there were no application-level faults present in v1.0; it was only due to wrong configuration of infrastructure. It's a situation that can be fixed by configuration changes.

After the initial controlled deployment of the three critical risks identified, documented, and remediated before the deployment, the security assessment validated that the three most significant risks were identified and documented, and were completely remediated. Embedding the DevSecOps workflow in the development process proved effective in discovering and fixing access-control issues with automated DAST scanning and manual code review. There are no reported Critical or High severity vulnerabilities at deployment, indicating that a general baseline level of security is suitable for controlled institutional use.

Here's another system-wide observation from both security and functional testing camps that tenant-ownership checks are being performed as a series of inline code in each controller today. This pattern is correct but can add to the possibility of omissions in subsequent endpoint additions. Putting the checks into a common base controller, or

a dedicated ASP. The next development phase of .NET Core will include a recommended architectural upgrade to NetsLib AuthorisationHandler.

SUMMARY AND CONCLUSION

Summary

This study is composed from ones that designed, developed, and evaluated a decentralized academic document verification platform called **VERACISM**, *Verifiable Academic Records for International School Manila (ISM)* for UPOU (University of the Philippines Open University). Incorporating cryptographic integrity verification, decentralized Web3 storage, and multi layered threat scanning into one platform, the system tackles the chronic issue of academic credential fraud.

VERACISM follows a multitenant model where the documents are individually-managed by the different educational institutions (tenants) sharing infrastructure of the platform itself. The main workflow is divided into three steps: (1) upload of documents to the system, (2) uploaded documents are checked in real time for malware and phishing using the ClamAV, YARA rules, and (3) PDF analyzer; the uploaded documents are encrypted and stored on the decentralized storage system Filecoin/IPFS, generating a tamper evident Content Identifier, CID; any participant in the system can check the authenticity of a document by resolving the CID and comparing the resulting file SHA-256 hash with the original document.

The system was subjected to five test processes namely, functional and integration test, performance test, security test and user acceptance test.

All the evaluations activities carried out were based on the baseline established in the Project Assessment chapter, among which a hybrid test strategy was implemented, an automated API execution along with manual role verification tests.

Functional and Integration Testing

Automated testing ran a set of **182 assertions** in 41 test cases,including 22 integration test cases, 93 assertions, for the scan, upload, report retrieval, CID verification, download, and storage workflows, and 19 functional test cases, 89 assertions exercising

role based access control for the Admin and Tenant roles. **Automated test passed on 182 assertions all with 100% rating.** (see Appendix 0.9 and Appendix 0.10).

There were 30 test cases covering Admin and Tenant workflows that were tested for using Role based manual and code review method. Of these, **27 passed fully** (90.0%), **2 passed partially** (6.7%), and **1 failed** (3.3%). The single unsuccessful TC, ”TC-ITO-005” exposed the lack of a re-retry mechanism should the Lighthouse storage service go unavailable, thereby putting the system at risk for transitory upload failures. Two “partial” results stood out as lacking some level of scoped audit log visibility in the UI, TC-ITO-006, and missing a `VerificationLog` write when a document is verified in the public, TC-E2E-003. These are three defects that have been documented and in the Future Work section these are focused upon. In spite of these items, the fundamental document lifecycle operations of ‘upload’, ‘scan’, ‘store’, ‘retrieve’ and ‘verify’, functioned as expected for the evaluated scenarios.

Performance Testing

This performance evaluation was done with k6, with 100 VUs and 100 sustained VUs, during two sessions at the public CID verification endpoint.

The first version, **Version 1.0**, had a higher failure rate, but it was not due to issues at the application layer, but due to an Nginx rate limiting policy which was limiting concurrent requests to only a certain number before sending them to the application. Following the rate limitations adjustment, the second iteration (**Version 1.1**) showed a **had a much lower error rate**, thus confirming the performance of the application layer under the tested load. Both runs showed that latencies were within acceptable limits, and P95 time values were similar to the overhead for decrypting and streaming content from the Lighthouse proxy. A summary of both rounds is provided in Appendix 0.12.

The results met the desired performance criteria from the Project Assessment baseline which included stable availability with concurrent traffic, desirable response characteristics, and observation of root cause explanation in order to see why there were errors.

User Acceptance Testing

Role based task scripts were completed and user acceptance testing was done using the same. The executed UAT cycle included 30 role based cases in all parts of the cycle (admin, tenant office, registrar, student, external verifier and cross role flow).

All responses made by participants provided in the form *VERACISM_UAT - Form Responses 1.csv* were used for UAT scoring.

UAT was rated as role scoped responses, such that the choice of non applicable UAT was left blank, by design, for those that did not work in that role. In total across 16 recorded role scoped evaluations there were **15 Passed** (93.8%), **1 partially completed** (6.2%) and **0 failed** evaluations. All 8 UAT IDs had responses at the unique level of the case; 7 of the 8 were approved and 1 partially.

These are the results that end users are ready to receive in a controlled way and maintain visibility of the improvements they have registered in the roadmap.

Security Testing

Security assessment was performed in two phases. The first phase employed Burp Suite Community Edition as a Dynamic Application Security Testing (DAST) tool and produced several tentative findings including SQL injection and reflected XSS candidates. Upon investigation, all SQL injection findings were determined to be false positives: the backend exclusively uses Entity Framework Core with parameterized queries, eliminating the SQL injection attack surface. Like reflected XSS candidates, these were also scanned through the Angular frontend, which prevents any type of output encoding in the context above.

The second phase involved manual security testing and review of remediation level on the code to validate the results of the first phase. During this phase, it was determined that many results obtained on the first pass with DAST were false positives, and therefore not vulnerabilities. After the validation, the team pinpointed and validated **five genuine vulnerabilities** (VUL-01 through VUL-05) with severity ranging from Low to Medium-High.

All five were made good before check-up:

- **VUL-01** (Low): Verbose error messages were produced that gave insights into stack traces; this may be addressed via generation of generic responses to include error messages.
- **VUL-02** (Medium-High): Missing `INTERNAL_API_KEY` enforcement on the Lighthouse encryption service; remediated by implementing API key middleware.
- **VUL-03** (Low-Medium): Temporary files uploaded by this service are not deleted after processing; Remediated by adding deterministic cleanup logic.
- **VUL-04** (Medium-High): Missing `GET /health` endpoint may lead to healthcheck failures for a Docker image; Remediated by adding the health endpoint.
- **VUL-05** (Medium-High): Instantiating an `HttpClient` without using the abstraction from the classes `LighthouseStorageService` abstraction; remediated by all Lighthouse calls being routed through the service layer.

Details about vulnerabilities and remediation can be found in the security Appendix 0.11.

The security responses were also in line with the Project Assessment baseline, which listed boundary access verification, safe error handling, and evidence based review of the content of the scanners.

Conclusion

Overall, these findings show that VERACISM is a **operationally validated, security hardened, and reasonably performant** platform for decentralized academic credential verification, which performs reasonably well across a diverse group of users. It meets the basic goal of the system: allow educational institutions to upload student documents, attach cryptographic integrity proofs to these files in a decentralized network and empower any stakeholder to independently verify the validity of the student documents without access to a central authority or trusted party.

The automated test suite had a **100% successful pass rate**, while the manual test suite had a **90.0% full pass rate** with limited gaps remaining just for noncritical feature, retry logic, audit log scope and verification log persistence. The gaps are not a risk to the fundamental verifier guarantee and are defined as a progression on the roadmap.

The security of VERACISM was significantly strengthened throughout the development process. All five confirmed vulnerabilities were mitigated, and the DAST SQL injection results were removed through architectural decision - instead of performing queries with user-supplied input, they are all formulated and passed as parameters through the entire solution. The system is free from any known critical or high severity vulnerabilities post remediation.

Performance testing showed that application layer is able to support moderate concurrent load without service degradation (100 VUs) with the correct infrastructure level rate limiting, when the number of expected requests is matched. That Nginx rate limiting misconfiguration was discovered in Version 1.0, and that the issue was resolved in Version 1.1 is a good result in itself, showing the implications of performance testing: it brought a real infrastructure bug to the surface which would not have been caught by functional testing alone.

Contributions

This study brings contributions in the following areas: (1) a working open source reference implementation of a decentralized document verification system namely Web3 storage, (2) a DevSecOps integration pattern between ClamAV, YARA and PDF phishing analysis in the document ingestion pipeline, (3) empirical evidence for functional testing, integration testing, performance testing, and the platform's security testing, and (4) documented vulnerability remediation lifecycle that supports similar academic information systems built in confined institutional settings.

VERACISM sets the groundwork for subsequent research into blockchain anchored academic credential systems, privacy preserving verification protocols, and interoperability standards with other systems like the W3C Verifiable Credentials and Open Badges.

FUTURE WORK

This section will provide practical steps a VERACISM might take post initial controlled deployment to reinforce VERACISM. The planned activities will concentrate on the following areas of resilience, security governance, observability, performance and standards alignment. It will also adhere to the same Assessment criteria established in the Project Assessment baseline to assess for the changes of the project in the future.

Reliability and Resilience

A retry policy with exponential backoff for transient communication failures between Lighthouse will be added, improving the scan and storage flow. Additionally, circuit breaker logic will be introduced to avoid repeated calls to a unhealthy upstream service. A queued upload workflow will further enhance the reliability by temporarily buffering and replay the upload when external services recover.

Functional and Integration Coverage Expansion

Future cycles should build the automated and manual coverage in the same manner in which test cases from the Project Assessment chapter were planned. Functional checking with UI will be enhanced with deeper UI-driven functional checks, integration check for service to service failures (wider), Evidence capture for pass, partial and fail (additional).

Audit and Verification Logging

The verification should continue saving the log entries to the VerificationLog table to track all verification operations. The tenant scoping applied to audit views will also be strictly enforced in API responses and the UI filters. The work will help to reinforce accountability and institutional compliance reviews.

Access Control Architecture

Current tenant ownership checks are effective, but they are distributed across several controllers. Future work should centralize authorization logic through a shared authorization layer to reduce duplication and lower the risk of missing ownership checks in future endpoints. Candidate approaches include policy-based handlers, reusable ownership validation services, and centralized authorization middleware for tenant-scoped resources.

Centralising it will also help you have consistent security review in the future. Instead of reviewing each individual controller, reviewers will review a common authorization path, and ensure that new modules conform to the same access control rules.

Performance and Capacity Expansion

Future performance studies will cover more end-to-end points than just public verification, particularly the complete load test for both upload and scanning tests while doing both tests concurrently. Measuring traffic with pilot institutions, then tuning rate-limit thresholds in the reverse proxy level will be done. Endurance tests with long durations will also be added to test for maintaining memory stability and constant throughput.

To ensure security and scanning pipelines are not overwhelmed with excess or abuse of uploads per tenant, a per-tenant upload limit will also be enforced. This control will keep one tenant from using all the shared scanning resources and aid in maintaining equal distribution of platform usage among institutions. Rate limit will depend on the identity of the tenant, Bulk volume upload, Queue capacity in scan queue on the system, and seen traffic patterns during pilot implementation.

The baseline measures as described in the Project Assessment chapter will be carried forward in the next performance cycle for consistency across versions and environments in terms of throughput, percentile latency, and error rate.

Security Hardening and Monitoring

The platform will be further enhanced with periodic dependency review, continuous security regression testing and automated notifications for unusual access patterns. Scan results, authorization events, upload activity and service health indicators will be collated in a centralized security dashboard to provide faster incident response.

For future testing, the set background principles of automated testing and manual testing will be retained, but with increased linkage of results to evidence of remediation. The performance and security requirements determined in the Project Assessment chapter will also be kept as a reference for additional platforms to be added in the future. This will keep new features from weakening previously tested controls or excessively slowing the response time, or introducing an escalation of risk in the upload/verification/scan process.

User Acceptance and Deployment Readiness

The role-based task scripts for administrators, tenant users, integrators, and external verifiers from the Project Assessment chapter will be formalized as user acceptance testing. This will help in ensuring institutional onboarding and operational handover with more assurance of deployment readiness evidence.

Interoperability and Research Extensions

In the future, this will be investigated with interoperability with verifiable credential frameworks and education metadata standards for the exchange of documents between institutions. Other research aims to investigate verification methods for selective disclosure of student records that preserve privacy.

Deployment and Adoption Roadmap

There will be a rollout at partner institutions ensuring the product validation of governance, onboarding effort, and operational support needs in a staged manner. Samples of user training materials, playbooks for tenant help desk, and tenant onboarding templates will be made more user-friendly and decrease implementation friction.

APPENDIX A: INTEGRATION TEST SUITE RESULTS

The final results for all assertions contained within each of the following tables are the results of the VERACISM API v1 Integration Test run on 30 March 2026, using the Newman CLI runner against the Veracism staging environment (`https://sb-api.ismanila.org/Veracism`). No failures or errors in the 93 assertions made for 22 test suites.

A.1: Scan Upload (`POST /api/v1/scan-upload`)

TC ID	Assertion	Result	Time (s)
TC-01	Status 200; SHA-256 is 64-char hex; threatLevel valid; reportId is a number; results array present; requestId (GUID or 32-hex); response time $<$30 s	**PASS**	20.446
TC-02	Status 200; studentId is non-null when studentNumber provided; response time $<$30 s	**PASS**	17.321
TC-03	Status 200; versionNumber $\geq$ 2; documentLabel present; response time $<$30 s	**PASS**	17.392
TC-04	Status 400; error is “no files”; response time $<$30 s	**PASS**	0.054
TC-05	Status 401 or 403 for invalid API key; response time $<$30 s	**PASS**	0.134
TC-06	Status 401 or 403 when no `X-Api-Key` header; response time $<$30 s	**PASS**	0.050
TC-07	Status 200; results array present; SHA-256 present for TXT upload; response time $<$30 s	**PASS**	16.888
TC-07A	Status 200; YARA match detected; webshell signature identified; threat level not SAFE; Lighthouse upload blocked; SHA-256 is 64-char hex; response time $<$30 s	**PASS**	4.734

Table 30. Integration test results: Scan Upload group (37 assertions, 0 failures, 77.969 s)

A.2: Get Report by CID (`GET /api/v1/reports/{cid}`)

TC ID	Assertion	Result	Time (s)
TC-08	Status 200; CID matches; SHA-256 is 64-char hex; filename non-empty; status is “complete”; verificationUrl present; threatLevel valid; uploadedAt is a date; response time $<$30 s	**PASS**	0.147
TC-09	Status 404; error is “Report not found”; response time $<$30 s	**PASS**	0.054
TC-10	Status 401 or 403 with no API key; response time $<$30 s	**PASS**	0.048
TC-11	Status 404 for non-owned CID (tenant enumeration hidden); response is not 403; response time $<$30 s	**PASS**	0.053

Table 31. The integration tests returned 17 assertions, 0 failures, and 0.307 s overall.The total time for the integration tests is 0.307 s, with 0 failures, 17 assertions, and a total of 0 reports.

A.3: Public Verification, Download, and Storage Status

TC ID	Group	Key Assertions	Result	Time (s)
TC-12	Verify	verified is true; CID matches; SHA-256 64-char hex; uploadedAt present; threatLevel valid	**PASS**	0.066
TC-13	Verify	Status 200; verified is false; message present (unknown CID)	**PASS**	0.055
TC-14	Verify	Status 200; verified is false for garbage CID; message present	**PASS**	0.053
TC-15	Verify	Status 200 without API key (public endpoint); verified is true	**PASS**	0.056
TC-16	Download	Status 200; Content-Disposition: attachment; Content-Type present; body non-empty	**PASS**	5.841
TC-17	Download	Status 404; error is “Report not found”	**PASS**	0.054
TC-18	Download	Status 401 or 403 when no API key	**PASS**	0.119
TC-19	Storage	Status 200; status is “pinned” or “processing”; accessible is boolean; size is a number	**PASS**	0.061
TC-20	Storage	Status 404; error is “Report not found”	**PASS**	0.053
TC-21	Storage	Status 401 or 403 when no API key	**PASS**	0.049

Table 32. Integration test results: Verify, Download, and Storage Status groups (41 assertions, 0 failures)

A.4: Integration Suite Overall Summary

Group	Test cases	Assertions	Pass	Fail	Duration (s)
Scan Upload	8	37	37	0	77.969
Get Report by CID	4	17	17	0	0.307
Verify (Public)	4	18	18	0	0.230
Download	3	12	12	0	6.014
Storage Status	3	11	11	0	0.163
TOTAL	**22**	**93**	**93**	**0**	**83.728**

Table 33. Integration test suite overall summary

APPENDIX B: FUNCTIONAL TEST SUITE RESULTS

The results of the entire VERACISM Functional Test Suite were run on 31 March 2026 and are presented in the following tables. The 15 automated API test cases take place through automation with Bearer-token authentication, executed through Newman; 11 verifications are made though Code Review; 4 are Observable checks. 0 failures and all 89 automated assertions passed.

B.1: Role-Based Functional Test Results

The 30 functional test cases are included below, each listed in a separate table, one for each actor role, for ease of readability.

TC ID	Test Case	Method	Result	Key Evidence
TC-ADM-001	Approve integration request	Code Review	**PASSED**	Admin tenant management pages; `TenantsController` CRUD; `is_active` flag
TC-ADM-002	Manage users and roles	Code Review	**PASSED**	`/admin/users` UI; `UsersController`; Auth0 role sync via `ValidateUserExistsFilter`
TC-ADM-003	Update keys and critical config	Code Review	**PASSED**	API Keys settings page; `ApiClient` table; ClamAV/YARA config via `appsettings.json`
TC-ADM-004	View audit logs and export	Code Review	**PASSED**	`GET /audit/export/csv` and `/pdf`; actor, action, and tenantId filtering; 10,000-record export cap
TC-ADM-005	View health and alerts	Code Review	**PASSED**	Admin dashboard health widgets; ClamAV/Lighthouse/DB status indicators

Table 34. Functional tests: System Administrator (TC-ADM), 5 test cases

TC ID	Test Case	Method	Result	Key Evidence
TC-ITO-001	API upload integration	Automated API	**PASSED**	`POST /api/v1/scan-upload` returns requestId, sha256, reportId, cid; all 7 assertions pass
TC-ITO-002	Unauthorized upload rejection	Automated API	**PASSED**	Invalid key → 401/403; missing header → 401/403; no file stored
TC-ITO-003	Metadata consistency	Automated API	**PASSED**	`GET /api/v1/reports/{cid}` CID, SHA-256, filename, uploadedAt, status all consistent
TC-ITO-004	Lighthouse anchoring	Automated API	**PASSED**	CID non-empty; status is “pinned” or “processing”; accessible: true; file size positive
TC-ITO-005	Temporary Lighthouse outage retry	Code Review	**FAILED**	No Polly retry policy; single HTTP POST with no retry/circuit breaker; future enhancement
TC-ITO-006	Scoped audit access	Code Review	**PARTIAL**	RBAC enforced; tenant isolation on Report queries; dedicated audit log UI not yet built

Table 35. Functional tests: IT Office Staff (TC-ITO), 6 test cases

TC ID	Test Case	Method	Result	Key Evidence
TC-REG-001	Upload finalized report	Automated API	**PASSED**	PDF with studentNumber, documentLabel, sourceSystem → 200; requestId, reportId, cid returned
TC-REG-002	Reject unsupported file type	Automated API	**PASSED**	Empty body → 400; error: “no files”; no results array
TC-REG-003	Associate report to correct student	Automated API	**PASSED**	studentNumber=999999 → studentId non-null positive integer; `Student` table ext_ref lookup
TC-REG-004	Receive CID and QR code	Code Review	**PASSED**	`FileViewerModal` generates QR via `qrcode` library; CID embedded in verification URL
TC-REG-005	View upload and anchoring status	Code Review	**PASSED**	Tenant files page shows report list with status labels (1=Processing, 2=Complete)
TC-REG-006	Attempt modify/delete uploaded report	Code Review	**PASSED**	No PUT/DELETE on reports; API is append only; UI does not render edit/delete actions
TC-REG-007	Controlled download	Automated API	**PASSED**	Content-Disposition: attachment; no Lighthouse URL in headers; body non-empty

Table 36. Functional tests: Administration Office / Registrar (TC-REG), 7 test cases

TC ID	Test Case	Method	Result	Key Evidence
TC-STU-001	Verify by QR code	Code Review	**PASSED**	QR encodes `/verify/{cid}`; public verify route is unauthenticated
TC-STU-002	Verify by CID entry	Automated API	**PASSED**	`GET /api/v1/verify?cid={cid}` → verified: true; sha256Match: true; inLighthouse: true
TC-STU-003	Invalid CID handling	Automated API	**PASSED**	Invalid CID → 404; descriptive message; no stack trace or internal details
TC-STU-004	Result screen comprehension	Code Review	**PASSED**	Color-coded badges; plain-language labels; CID labeled as “Document Identifier”
TC-STU-005	Access boundary check	Code Review	**PASSED**	`auth0Guard` protects `/admin/*` and `/tenant/*`; unauthenticated users redirected

Table 37. Functional tests: Student (TC-STU), 5 test cases

TC ID	Test Case	Method	Result	Key Evidence
TC-EXT-001	Verify by QR (no login)	Automated API	**PASSED**	Public endpoint (`[AllowAnonymous]`); verified: true without any auth headers
TC-EXT-002	Verify by CID (valid/altered/unavailable)	Automated API	**PASSED**	Valid → verified: true; unavailable → verified: false with descriptive message
TC-EXT-003	Repeat check from different networks	Code Review	**PASSED**	Deterministic DB lookup; no session state; no IP-based filtering
TC-EXT-004	Restricted internal access	Automated API	**PASSED**	Download and report endpoints return 401/403 without credentials

Table 38. Functional tests: External Auditor / Educator (TC-EXT), 4 test cases

TC ID	Test Case	Method	Result	Key Evidence
TC-E2E-001	Upload-to-verify journey	Automated API	**PASSED**	Upload → CID → public verify → sha256Match: true; full journey in single automated run
TC-E2E-002	Tampered file negative case	Automated API	**PASSED**	Fabricated CID → 404; integrity warning message; no false positive verification
TC-E2E-003	Unavailable deal negative case	Code Review	**PARTIAL**	`/verify` returns verified: false for missing CID; health alerting not implemented

Table 39. Functional tests: Cross-Role End-to-End (TC-E2E), 3 test cases

B.2: Functional Suite Overall Summary

Feature Group	Suites	Assertions	Pass	Fail	Duration (s)
F1: IT Office API Tests	5	27	27	0	54.231
F2: Registrar Upload Tests	4	20	20	0	39.508
F3: Public Verification Tests	7	28	28	0	21.047
F4: End-to-End Journey	3	14	14	0	24.533
TOTAL	**19**	**89**	**89**	**0**	**116.641**

Table 40. Functional test suite automated results summary (89 assertions, 100% pass rate)

B.3: Functional Requirement Coverage Matrix

FR	Description	Test Cases	Coverage
FR01	Document upload and scan	TC-ITO-001, TC-REG-001, TC-REG-002, TC-REG-003, TC-E2E-001	Full
FR02	Threat detection (ClamAV, YARA, PDF phishing)	TC-ITO-001 (threatLevel assertion), TC-07A	Partial
FR03	SHA-256 hashing for integrity	TC-ITO-001, TC-ITO-003, TC-STU-002, TC-E2E-001, TC-E2E-002	Full
FR04	Lighthouse/IPFS storage and anchoring	TC-ITO-004, TC-REG-004, TC-E2E-001	Full
FR05	CID generation and QR code	TC-ITO-001, TC-REG-004, TC-STU-001, TC-EXT-001	Full
FR06	Authentication and authorization	TC-ITO-002, TC-ADM-001, TC-ADM-002, TC-STU-005, TC-EXT-004	Full
FR07	Public verification (no auth)	TC-STU-002, TC-STU-003, TC-EXT-001, TC-EXT-002, TC-E2E-001, TC-E2E-002	Full
FR08	Multi-tenant data isolation	TC-ITO-003, TC-ITO-006, TC-REG-003, TC-11	Partial
FR09	Audit logging	TC-ADM-004, TC-ITO-006, TC-E2E-003	Partial
FR10	Controlled download (no direct URL exposure)	TC-REG-007, TC-EXT-004	Full

Table 41. Functional requirement to test case traceability matrix

APPENDIX C: SECURITY ASSESSMENT FINDINGS AND REMEDIATION

Security evaluation of VERACISM was conducted in two phases: an initial dynamic application scan (DAST) using Burp Suite, followed by manual verification of findings and targeted penetration testing. Five confirmed vulnerabilities were identified and subsequently remediated before the initial controlled deployment. The tables below summarise the findings, affected endpoints, and remediation status.

C.1: Initial DAST Scan Summary

Finding type	Severity	Confidence	Note
SQL Injection (4 instances)	High	Tentative	Responses consistent with path-based 403 from IIS; ORM/parameterized queries confirmed by code review: false positive
Cross-Site Scripting (reflected)	Medium	Tentative	Injected payloads absorbed by Angular's built-in output encoding; not exploitable in practice
Missing security headers	Low	Certain	`X-Frame-Options`, `X-Content-Type-Options`, and `Strict-Transport-Security` present; some cache-control headers absent on API responses
Backup / test file disclosure	Informational	Certain	Backup URL patterns flagged; all 403-gated by IIS; no actual backup files accessible
Email address disclosure	Informational	Certain	User email fields present in authenticated API responses; by design for admin role
Cacheable HTTPS response	Informational	Certain	Cache-control headers absent on a small number of read-only public endpoints

Table 42. Initial Burp Suite DAST scan summary (staging environment, 21 March 2026)

C.2: Confirmed Vulnerabilities and Remediation Status

ID	Title	Severity	Remediation Applied	Status
VUL-01	Unbounded API Client Data Response	Low	Server-side pagination (`OFFSET/FETCH`); configurable page/pageSize; max cap of 100 per page	Remediated
VUL-02	BOLA on API Client Records	Medium–High	Explicit tenant ownership check on GET, PUT, PATCH, DELETE; HTTP 403 for cross tenant reads	Remediated
VUL-03	Insufficient File Type Validation	Low–Medium	Allowlist of 18 permitted extensions via `FileValidationHelper`; applied before scanning pipeline; frontend `accept` attribute updated	Remediated
VUL-04	BOLA on Report Records	Medium–High	Tenant ownership check on `GET /reports/{id}`, `GET /reports/{id}/latest`, and `GET /reports/user/{userId}`; HTTP 403 for cross tenant access	Remediated
VUL-05	Missing Tenant Isolation on File View/Download	Medium–High	`EnforceTenantOwnershipAsync(cid)` helper in VerifyController; admin pass-through retained; HTTP 403 for non-owner callers on view and download	Remediated

Table 43. Confirmed vulnerabilities and remediation status (Security Remediation Report, 27 March 2026)

C.3: Deferred Findings

ID	Description	Rationale for Deferral
DEF-01	BOLA on `GET /api/reports/user/{userId}`: partial tenant check	Addressed in VUL-04; cross-user enumeration within a tenant requires additional UX review
DEF-02	Nullable reference warnings in `TenantRepository.cs` and `UsersController.cs`	Pre-existing warnings; no runtime impact confirmed; scheduled for nullable annotation cleanup

Table 44. Deferred security findings tracked for next development sprint

APPENDIX D: PERFORMANCE TEST RESULTS

The VERACISM staging environment was used for two rounds of performance testing using Postman's in-built Performance Testing feature. The tests focused on the public CID validation endpoint (`GET /api/v1/verify?cid={cid}`), which is the most visited endpoint in the typical workload incoming to production.

D.1: Test Configuration

Parameter	**Version 1.0**	**Version 1.1**
Test date	2026-03-28	2026-03-30
Target endpoint	`GET /api/v1/verify?cid={cid}`	
Virtual users (VU)	100	100
Test duration	5 minutes	5 minutes
Infrastructure rate limiting	Enabled (default)	Disabled
Rationale for v1.1	:	Rate-limiting was identified as the root cause of elevated error rates in v1.0; disabled for v1.1 to establish baseline throughput

Table 45. Performance test configuration: Version 1.0 vs. Version 1.1

D.2: Key Observations

Metric	Version 1.0	Version 1.1
Error rate	Elevated (infrastructure rate limiting triggered)	Significantly reduced (rate limiting off)
Average response time	Higher (queuing due to throttling)	Lower (direct path to backend)
Throughput	Limited by rate limiting rules	Near-maximum for staging hardware
Root cause identified	Rate-limiting rules on the Nginx reverse proxy were set at levels below the 100-VU load	Confirmed: disabling rate limiting reduced errors; rate limits should be tuned for production load

Table 46. Observations summary for performance test, for complete numeric results, consult at VERACISM Performance Testing Version 1.0 and Version 1.1)

The performance testing exercise found that the system infrastructures, configured at the Nginx reverse proxy layer, have a rate limiting policy, so the policy is less than you need for the required load of 100 concurrent users. Version 1.0 responses indicated a significant percentage of too many requests (HTTP 429) responses in front of requests for the application layer. Based on version 1.1 results (where the rate limiting is disabled for the test), it ultimately showed that the application itself was able to deal with the concurrent load with considerably fewer error rates, which allowed to conclude that the underlying . worked.The tested load can be supported by the .NET 8 application server and SQL Server database. The rate limiting configuration needs re-tuning for production release in order to balance protection against abuse and concurrent usage.

LITERATURE CITED

AMAZON WEB SERVICES. (2025). *Using versioning in s3 buckets.* Retrieved from https://docs.aws.amazon.com/AmazonS3/latest/userguide/versioning-workflows.html (Amazon Simple Storage Service documentation)

BETTER GOVERNMENT PHILIPPINES. (2025). *Govchain: A foundational blockchain platform for government data transparency.* Retrieved from https://github.com/bettergovph/govchain (GovChain open source repository and description)

BLOCKCHAINREPORTER. (2025). *Interplanetary file system (ipfs): What is it?* Retrieved from https://blockchainreporter.net/web3/ipfs/

CFLOW. (2025). *Understanding audit trails: Why are they important for transparency and compliance in business operations.* Retrieved from https://www.cflowapps.com/audit-trails/

COMMISSION ON HIGHER EDUCATION. (2025). *Ched electronic certification, authentication, and verification (ecav) web application system.* Retrieved from https://ecav.ched.gov.ph/ (Public advisory and portal for electronic certification, authentication, and verification of higher education records)

CRYPTO.ST. (2025). *Interplanetary file system (ipfs) definition and meaning.* Retrieved from https://crypto.st/dictionary/interplanetary-file-system-ipfs/

DEPARTMENT OF PUBLIC WORKS AND HIGHWAYS. (2025). *Integrity chain: Blockchain transparency platform for infrastructure projects.* Retrieved from https://ched.gov.ph/public-advisory-on-the-full-implementation-of-ched-electronic-certification-authentication-and-verification-ecav-web-application-system/ (Describes Integrity Chain as a tamper evident platform for public works data)

ENCRYPTHOS. (2025). *Why cryptographic proof is replacing institutional trust.* Retrieved from https://encrypthos.com/guide/why-cryptographic-proof-is-replacing-institutional-trust/

FILEBASE. (2025). *What is ipfs pinning?* Retrieved from https://docs.filebase.com/ipfs-concepts/what-is-ipfs-pinning

GOVERNMENT TECHNOLOGY AGENCY OF SINGAPORE. (2019). *With this blockchain-based platform, you may no longer need physical certificates.* Retrieved from https://www.tech.gov.sg/technews/with-this-blockchain-based-platform-you-may-no-longer-need-physical-certificates/ (Introduces the OpenCerts initiative)

MOHEBI, A., & AGHDAM, R. (2024). Diar: A blockchain-based system for generation and verification of academic diplomas. *Discover Applied Sciences*, *6*(6). Retrieved from https://doi.org/10.1007/s42452-024-05984-1 doi: 10.1007/s42452-024-05984-1

NAJI, M., JAWAD, H., HAFIZ, A., AL OBAIDI, A., & ABD, S. (2023). A systematic review of blockchain-based initiatives in academic certificate management. *Computers*, *14*(4), 141. Retrieved from https://www.mdpi.com/2073-431X/14/4/141 doi: 10.3390/computers14040141

NANYANG TECHNOLOGICAL UNIVERSITY. (2025). *Opencerts digital certificates and transcripts.* Retrieved from https://www.ntu.edu.sg/education/academic-services/opencerts-digital-certificates-and-transcripts (Institutional guidance on using OpenCerts at NTU)

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. (2023). *Cybersecurity log management planning guide* (Tech. Rep. No. NIST SP 800-92 Rev. 1). National Institute of Standards and Technology. Retrieved from https://csrc.nist.rip/pubs/sp/800/92/r1/ipd (Initial public draft)

NATIONAL RESEARCH COUNCIL. (2005). *Building an electronic records archive at the national archives and records administration: Recommendations for a long-term strategy*. Washington, DC: The National Academies Press. Retrieved from https://nap.nationalacademies.org/catalog/11332/building-an-electronic-records-archive-at-the-national-archives-and-records-administration doi: 10.17226/11332

PINATA. (2025). *What is ipfs pinning?* Retrieved from https://docs.pinata.cloud/ipfs-101/what-is-ipfs-pinning

PROTOCOL LABS. (2025). *What is ipfs?* Retrieved from https://docs.ipfs.tech/concepts/what-is-ipfs/ (IPFS Documentation)

SCHNEIDER, R. (2025). *Pinning services in IPFS engine.* Retrieved from https://richardschneider.github.io/net-ipfs-engine/articles/repo/pin.html

SIMPLE BUT NEEDED, INC. (2025). *What are the risks of not maintaining an audit trail in ehs management?* Retrieved from https://sbnsoftware.com/blog/what-are-the-risks-of-not-maintaining-an-audit-trail-in-ehs-management/

SKILLSFUTURE SINGAPORE. (2018). *Annex b: How opencerts works.* Retrieved from https://www.skillsfuture.gov.sg/docs/default-source/newsroom/annex-b---how-opencerts-works.pdf (Technical annex describing OpenCerts data model and verification flow)

SY, M. P. M., MARASIGAN, R. I., & FESTIJO, E. D. (2024). Educredph: Towards a permissioned blockchain network for educational credentials verification system. In *2024 12th international conference on information and education technology (iciet)* (pp. 434–439). Yamaguchi, Japan: IEEE. Retrieved from https://ieeexplore.ieee.org/document/10542756 doi: 10.1109/ICIET60671.2024.10542756