Memorial University of Newfoundland

Faculty of Engineering and Applied Science

Course: ENGI 9837 – Software Design and Specification

Team Members: Priyanka Saha, Gerard, Sadia Alam, Anil, Ronit

Date: November 1, 2025

Minimum Viable Product (MVP) Report

1. Overview

Our MVP demonstrates the core trust workflow that certifies and verifies files against the blockchain. The objective was to validate the backend trust logic before adding the user-interface layer, PyPI SDK release, and CI/CD automation.

2. Features Implemented in MVP

Full Design Module	**MVP Subset Implemented**	**Description / Purpose**	**Status**
Module 1 – Smart Contract Layer	Deployed minimal EVM contract on Tenderly Sepolia	Provides store(), retrieve(), and get_total_records() for on-chain trust verification	Completed
Module 2 – Python SDK Backend	Core hashing & blockchain interaction functions	Handles local file hashing, wallet key management, gas estimation, and transaction signing via web3.py	Completed
Module 3 – Storage Layer	MongoDB mid-layer integration	Enables fast record lookup while waiting for blockchain confirmation	Completed
Module 4 – FastAPI Interface	Upload and verification endpoint testing	Facilitates quick validation of file certification workflow	Completed
Module 5 – Frontend UI / Portal	Keep in view (KIV)	React interface and public portal to be developed after MVP	Phase 2

Module 6 – CI/CD Pipeline & Packaging	Keep in view (KIV)	CI/CD pipeline and packaging setup not started during MVP phase; planned for future iteration.	Phase 2

3. Alignment Between Design Specifications and MVP

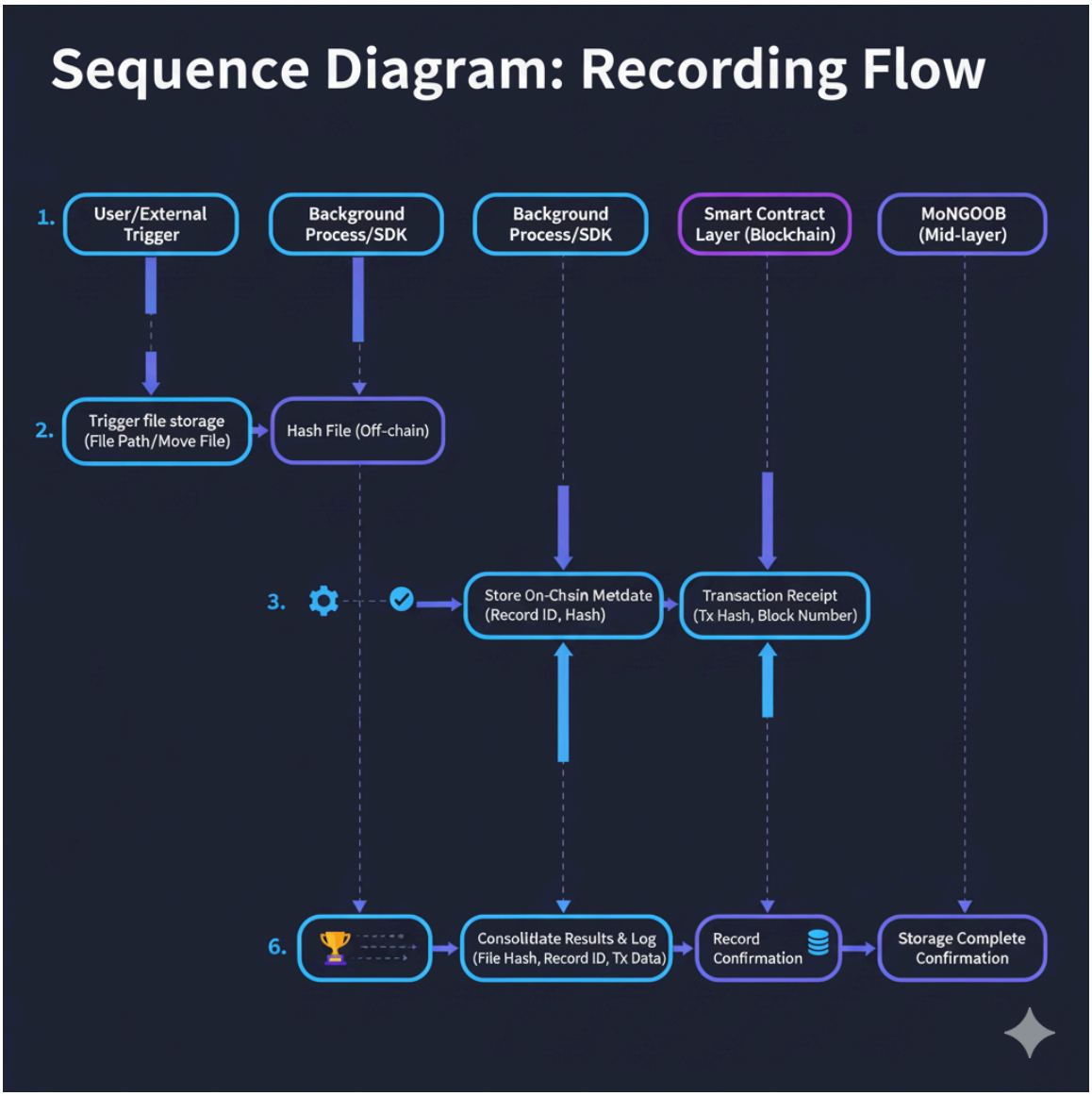
Design Specs Deliverable	**Description (as per Design Specs)**	**MVP Status**
Core Trust Layer (Smart Contract)	Implements recordHash() and checkHashExists() for immutable verification.	Completed – Deployed on Tenderly Sepolia testnet; interface renamed to store(), retrieve(), and get_total_records() for a clear description of process.
Python SDK (Backend Engine)	Handles local hashing, gas estimation, wallet management, and contract interaction.	Completed – Functional and tested with FastAPI integration.
React Component Library	UI components for certification and integrity verification.	KIV – Scheduled for Phase 2 development.
Public Verification Portal	External web portal for third-party file verification.	KIV – Scheduled for Phase 2.
CI/CD Pipeline	Automated build, test, and deployment in GitLab.	KIV – Not started in MVP phase.

4. Process flow

Due to high blockchain transaction gas cost, each file is logged with a record ID instead of direct message digest. We introduced MongoDB as a mid-layer database to keep track of all recorded files with their hash as well as the record ID.

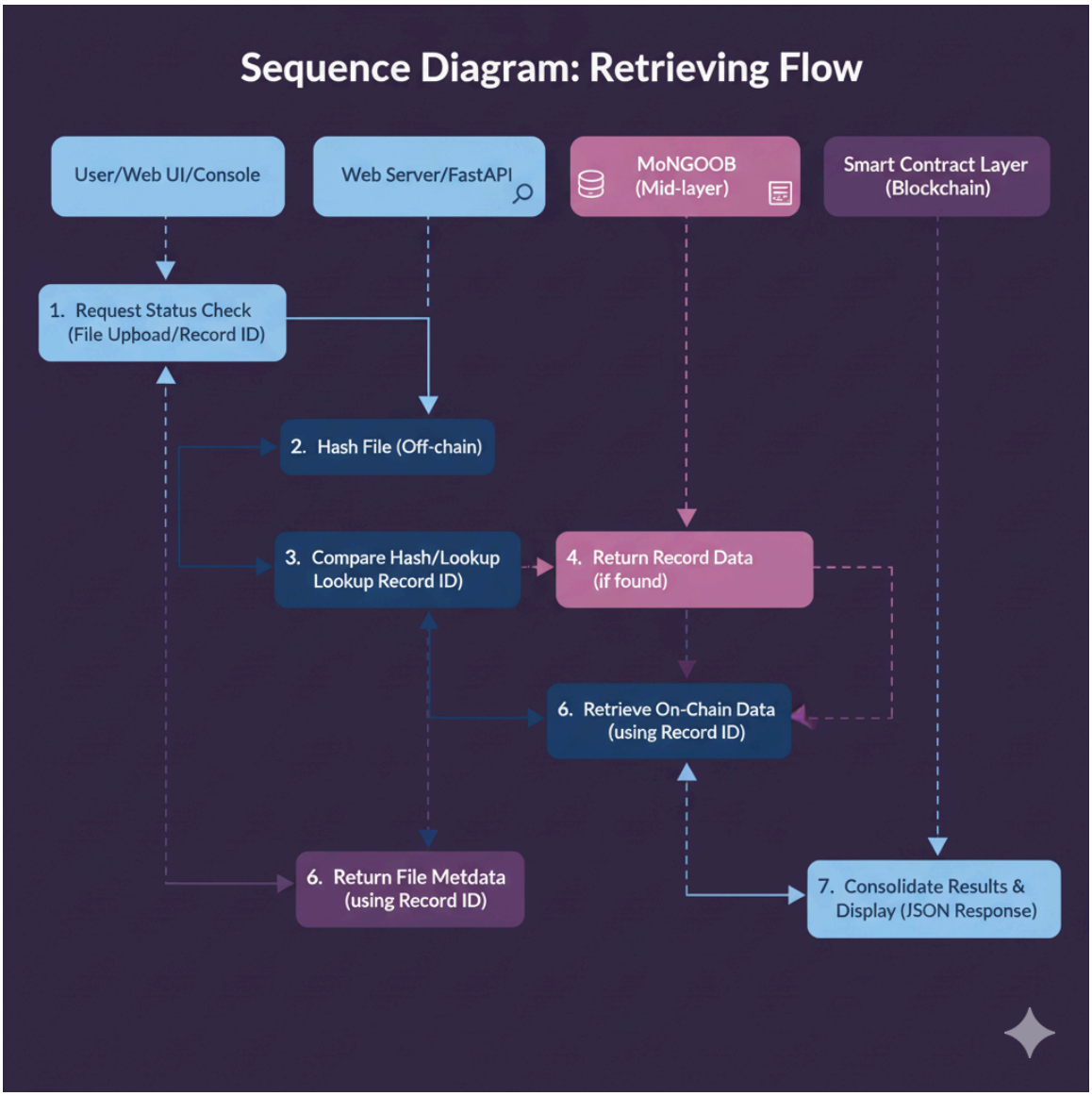
4.1 Recording flow

1. Storing file(s) can be triggered by direct function calls on the file path or moving them to the target directory which is then picked up by the background process.
2. The file is hashed off-chain for message digest.
3. File is stored on-chain with transaction hash, block number, record ID and timestamp
4. Results are consolidated and logged on MongoDB.



4.2 Retrieving flow

1. File can be uploaded directly on the web server (provided by fastapi docs) to check for recording status or direct function calls with record ID.
2. The file is hashed off-chain for message digest and compared with MongoDB records.
3. The File is retrieved with record ID (if found) on-chain with file hash, block number, timestamp
4. Results are consolidated and displayed as response JSON on web UI or console (with direct function calls).



5. Test Evidence and Results

Test Case / Scenario	Observed Output	Result
Smart Contract Deployment	Contract deployed on Tenderly Sepolia; verified transaction hash	Pass
store_record Execution	File hash successfully written to blockchain (log ID = #24)	Pass
retrieve_record Execution	Returns correct hash and timestamp from blockchain	Pass
MongoDB Storage Validation	Record ID and hash match with on-chain entry	Pass
FastAPI Upload Verification	JSON response shows verification_status: true	Pass

6. Test Evidence Screenshots

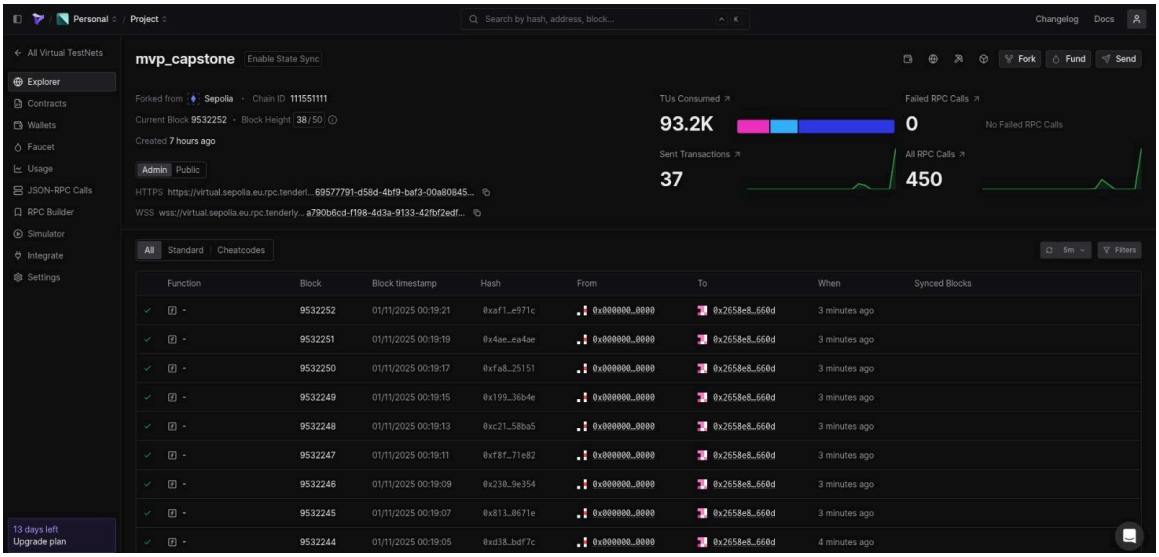


Figure A1. Smart Contract on-chain transactions – Tenderly Sepolia

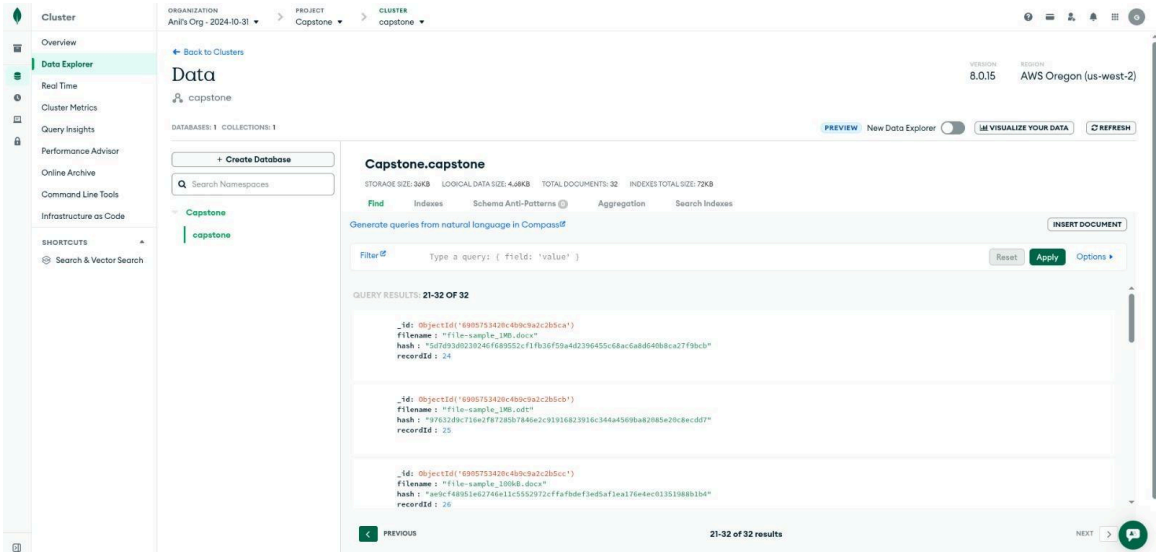


Figure A2. MongoDB Record Snapshot

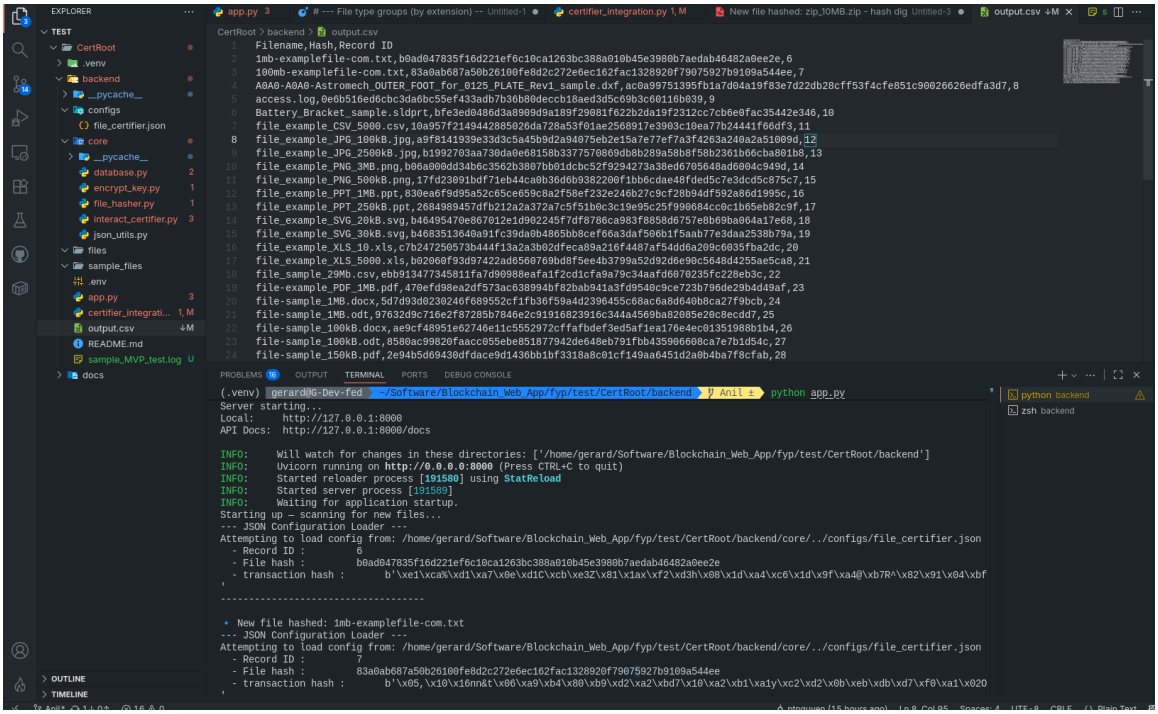


Figure A3. Python SDK Execution Logs

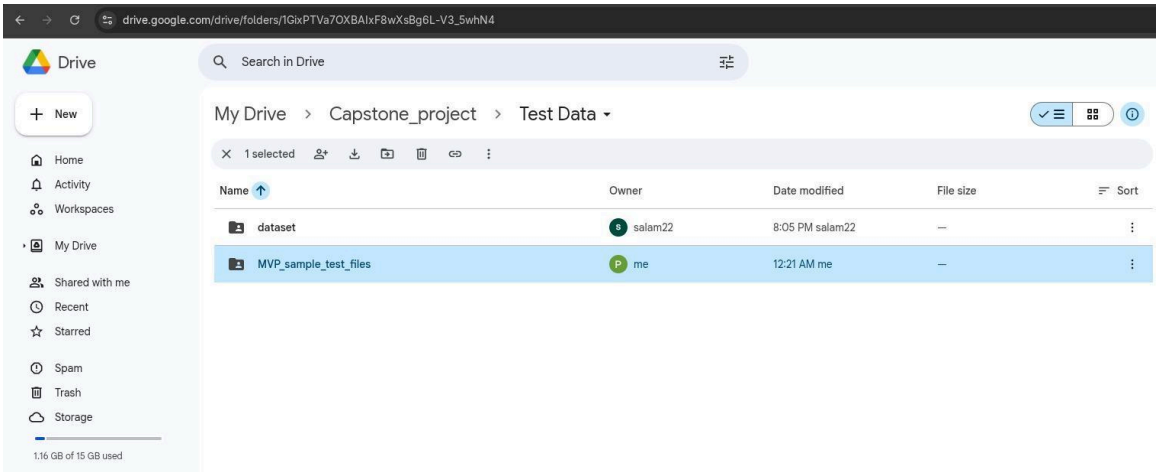


Figure A4. MVP sample dataset

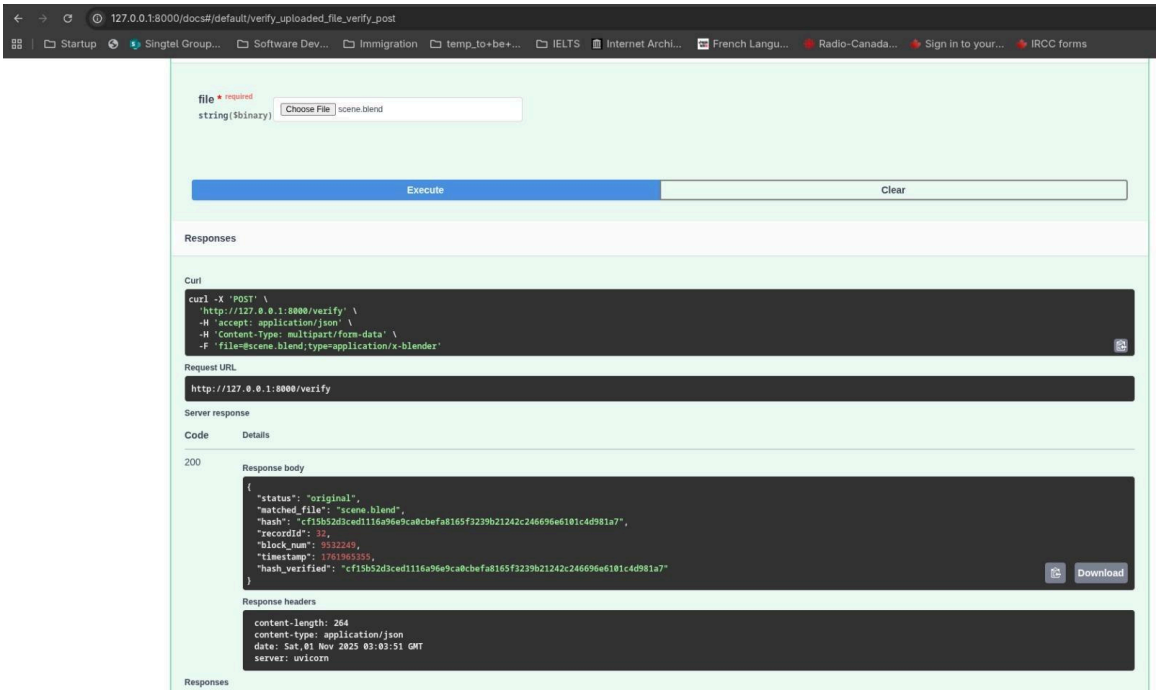


Figure A5. FastAPI Upload Verification (retrieving)

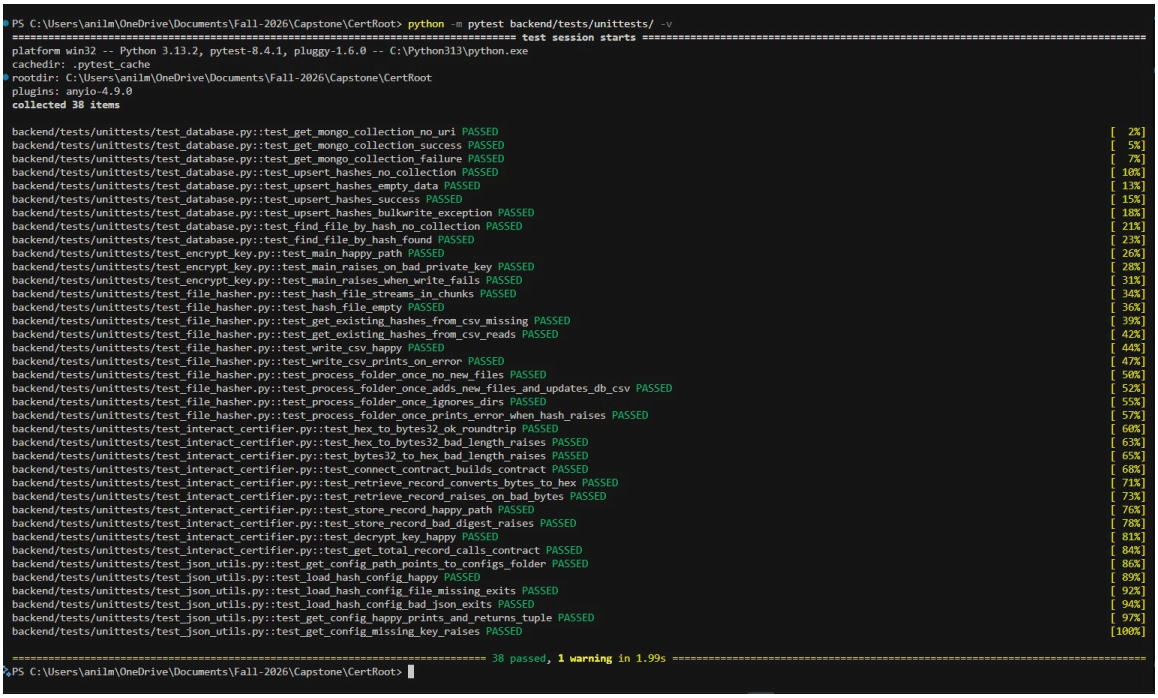


Figure A6. Unit Test log

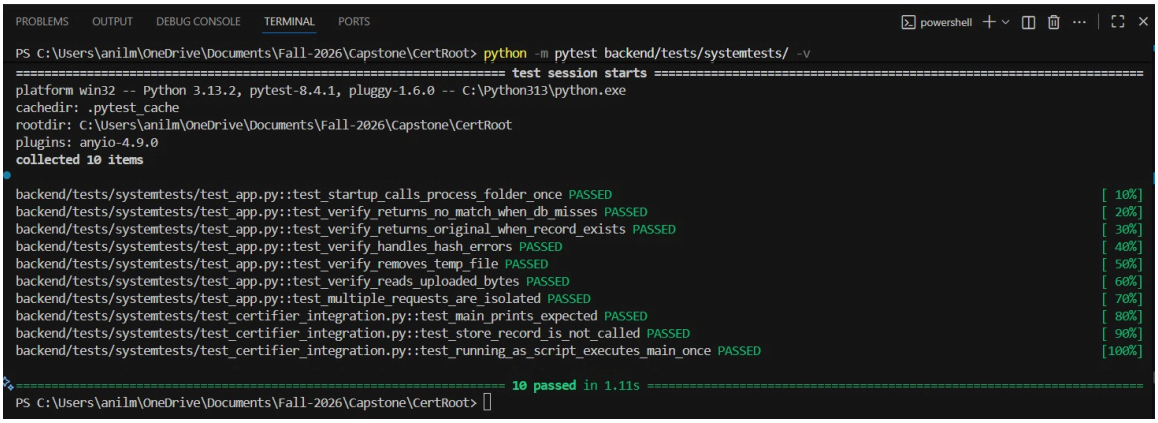


Figure A7. System Test log

7. Challenges and Solutions

• High gas cost and slow confirmation on testnet → Introduced MongoDB cache layer.
• Gas estimation and network timeouts → to add retry and structured exception handling (Phase 2).
• Limited testnet block quota (50) → Optimized contract interactions.
• Integration complexity → Used shared ABI, consistent branching, and FastAPI tests.

8. Tasks planned for phase 2

The following components are planned for the next development phase and were intentionally deferred from the MVP:

• CI/CD pipeline setup in GitLab (full automation including deployment and artifact publishing)
• PyPI publishing of the Python SDK for public access
• Front-end React component library release for UI verification
• Public verification portal for user access and transparency

9. Team Contributions

Team Member	Primary Responsibilities	Key Deliverables / Contributions
Anil Manyam	Backend development, system testing, File hashing validation, and API testing	Built a whole backend FastAPI logic for upload verification, validated end-to-end flow with system testing.
Gerard Nguyen	Smart contract logic, deployment, and integration with the main backend.	Developed and deployed EVM contract on Tenderly Sepolia, validated ABI and contract calls.
Ronit	Test case building and file management.	Built Unit test cases for all the functionalities across the code base.
Priyanka Saha	Exception handling and retry mechanisms, as well as documentation.	Documenting the progress of the project and research on deployment options with Flow Design
Sadia Alam	MongoDB management and logging and Data collection across various sources.	Collected different formats of data from various sources across the web.

All members contributed to architecture discussions, integration testing, and debugging sessions during MVP development.