Ledger – Phase 1 Full Implementation Plan (CS50 Final Project)

1. Project Overview & Objectives

Ledger is a cybersecurity-focused audit system designed as the CS50 final project and the first stage of a larger secure framework. Its purpose is to record, protect, and verify system or security events using a tamper-evident hash chain. The project demonstrates key skills in systems programming, data persistence, and security validation by combining C, Flask, and SQL technologies.

Primary objectives:
- Record system events in an immutable sequence.
- Ensure verifiable integrity of each stored entry.
- Provide a user-friendly interface for event insertion and verification.
- Detect and report any modification to existing records.

2. Detailed Architecture

The Ledger system is composed of three main layers working together:

1. C Core: Handles cryptographic hashing and chain verification. It receives serialized event data and returns computed hash values.
2. Flask Application: Acts as the interface layer between the user and the core logic. It manages HTTP requests, input validation, and responses.
3. SQL Database (SQLite for Phase 1): Stores all verified events persistently in a single append-only table.

The data flow is linear: user input → Flask validation → C core hashing → database storage → optional chain verification.

3. Development Milestones & Workflow

The implementation will follow a structured workflow:
1. Initialize the development environment and database schema.
2. Develop and test the C core hashing and verification functions.
3. Build Flask endpoints for adding, listing, and verifying events.
4. Integrate the C core with Flask using ctypes.
5. Test full workflow with simulated events and tampering scenarios.
6. Finalize user interface and prepare documentation for CS50 submission.

4. Environment Setup Guide

Development tools and configuration:
- Operating system: Linux, macOS, or Windows with WSL.
- Compiler: GCC (for building C code) with secure flags.
- Python version: 3.10 or newer.

- Flask for web interface, SQLite for local data.
- GitHub repository for version control and submission.

Setup steps:
1. Create project folders for C core, Flask app, database, tests, and docs.
2. Initialize Git repository.
3. Configure virtual environment and install dependencies.
4. Prepare sample database file for development.

5. Interface Specification

Interaction between Flask and the C core:

- Flask passes serialized event data (timestamp, actor, action, details, prev_hash) to the C library.
- C library computes new hash using cryptographic function and returns it.
- Flask receives the result, stores the event with its hash in the database, and confirms success.

Available Flask routes:
- /add: adds a new event.
- /events: lists all stored events.
- /verify: runs full chain verification and returns result.

This separation ensures that cryptographic operations remain isolated and safe while Flask handles user interactions.

6. Testing Plan

Testing ensures correctness and resilience of the system.
Test categories:
- Unit testing: validate hash generation and chain verification in C.
- Integration testing: confirm communication between Flask, C, and the database.
- Tamper detection testing: manually alter one record and confirm detection.
- Persistence testing: restart the system and verify data continuity.
- Boundary testing: handle empty input, long strings, or missing fields safely.

7. Risk Assessment Matrix

Identified risks and mitigation strategies:
- Integration risk (Flask ↔ C): mitigate by testing the C library independently first.
- Data loss or corruption: mitigate with regular database backups.
- Deadline pressure: mitigate through incremental weekly milestones.
- Environment issues: document dependency installation steps clearly.
- Demonstration failure: prepare pre-verified backup copy of the database for the CS50 demo.

8. CS50 Submission Checklist

For CS50 completion, the project must include:
- A fully working local Flask app demonstrating Ledger functions.
- README explaining objectives, structure, and usage.
- Demonstration video (less than 5 minutes) showing event creation, verification, and tamper detection.
- Properly commented code in both C and Python.
- Project report files (design specification, roadmap, implementation plan).

9. Transition Plan (Phase 2)

After successful CS50 submission, Ledger will transition into Phase 2 — Security & Cloud Deployment. Planned upgrades:
- Migration from SQLite to PostgreSQL for scalability.
- Deployment on Azure App Service under the coreventra.com domain.
- Implementation of HTTPS, JWT authentication, and user roles.
- Integration of post-quantum cryptography algorithms (Dilithium, Kyber) in the C layer.
- Connection with Coreventra ecosystem modules for unified audit logging.

10. Summary & References

This implementation plan defines a complete blueprint for building the Ledger Phase 1 prototype. It ensures structured progress from setup to CS50 submission and prepares the project for real-world extension in later phases. By combining C performance, Flask accessibility, and SQL persistence, Ledger demonstrates practical cybersecurity engineering principles. References include CS50 documentation, Flask official guide, and cryptographic best practices from NIST standards.