Ledger – Phase 1 Execution Phase (CS50 Final Project)

1. Purpose

This document provides a structured execution guide to move from planning and design to an operational prototype of the Ledger Phase 1 project. It defines concrete development, testing, and submission steps required to complete the CS50 final project successfully.

2. Pre-Development Setup

Before coding, ensure the environment and repository are properly prepared:
- Create the project folder structure: c_core, flask_app, database, tests, docs.
- Initialize a Git repository for version control.
- Install dependencies: Python ≥ 3.10, Flask, SQLite.
- Verify that the C compiler (GCC) is installed and accessible from terminal.
- Create a Python virtual environment to isolate dependencies.
- Prepare an empty SQLite database file for local testing.

3. Core Development Steps

The implementation follows sequential construction of the system's three layers:

Step 1 – C Core Implementation:
• Build functions to compute and verify hashes.
• Test using sample data before integration.

Step 2 – Flask API Development:
• Create endpoints: /add, /events, /verify.
• Implement input validation and structured responses.

Step 3 – Integration:
• Connect Flask to the C core using ctypes or cffi.
• Integrate with SQLite for storage.

Step 4 – Verification Functionality:
• Implement logic to recompute hashes for chain validation.
• Ensure system flags tampering correctly.

Step 5 – Dashboard Interface:
• Build a simple web page with three sections: Add Event, Event List, Verify Chain.

4. Testing Activities

Testing validates the system's logic and ensures correct tamper detection:
- Unit Testing: check that the C core functions return consistent hash values.
- Integration Testing: confirm Flask and C exchange data correctly.

- Tamper Simulation: manually edit a record to verify detection.
- Persistence Test: restart the application to ensure chain continuity.
- Verification Test: run ‘Verify Chain’ on normal and altered data.

5. Validation Criteria

The system is valid when:
- Each event is correctly added and displayed.
- Verification confirms integrity on unaltered data.
- Tampering triggers an immediate mismatch message.
- No direct modification or deletion is possible through the app.
- All documentation matches implemented functionality.

6. Documentation Tasks

During development, maintain clear records:
- Update README with setup and usage instructions.
- Keep a short technical log of errors and fixes.
- Capture screenshots or output logs for demonstration.
- Store all specification and plan files inside the /docs folder.
- Maintain consistent commit messages describing progress.

7. Demonstration Preparation

For CS50 presentation:
- Record a video showing event creation, viewing, and verification.
- Include a tamper simulation and show detection.
- Narrate system workflow briefly.
- Keep video between 3 and 5 minutes.
- Ensure all outputs are visible and readable.

8. Submission Process

Once testing and demonstration are complete:
- Push final code and documentation to GitHub.
- Include README, demo video link, and specification files.
- Follow CS50 submission form instructions precisely.
- Retain local backups of repository and database.
- After submission, tag the repository as ‘v1.0 – CS50 Final Project’.

9. Post-Submission Actions

After completing CS50 submission:
- Begin Phase 2 preparation: migrate database to PostgreSQL.
- Deploy Flask app on Azure under the coreventra.com domain.
- Introduce authentication and encryption modules.
- Continue version control with a new branch ‘Phase2-Azure’.
- Document all changes for integration with Coreventra systems.

10. Summary

This execution plan transforms the Ledger design into a tangible, verifiable prototype. Following these steps ensures a systematic approach—preparation, coding, testing, documentation, and delivery. Successful execution will result in a fully working, CS50-compliant project that clearly demonstrates secure data integrity principles.