Ledger – Phase 1 Detailed Design Specification

Step 1 – Design Specification

Purpose: Define the foundation of Ledger, your CS50 final project. Ledger is an immutable cybersecurity audit system designed to record and verify security events using a cryptographic hash chain. It combines three components:
- C core: performs secure hash computation and verification.
- Flask: handles user interaction and API logic.
- SQL (SQLite or PostgreSQL): stores the audit data persistently.

The system demonstrates data integrity, immutability, and tamper detection — key principles in cybersecurity auditing.

Step 2 – Event Structure

Each event represents a single action within the system. Every record is cryptographically linked to the previous one, forming an unbreakable chain of trust. The event fields are:
- ID: sequential identifier.
- Timestamp: automatically generated at event creation.
- Actor: entity performing the action.
- Action: operation performed.
- Details: contextual metadata.
- Prev Hash: hash of the previous record.
- Hash: new hash calculated from this record's data and Prev Hash.

This model ensures that if any past event changes, all subsequent hashes become invalid.

Step 3 – Functional Workflow

The functional process describes how users and the system interact with the ledger:

1. Initialization: On startup, Ledger loads the existing database or creates a genesis record.
2. Add Event: User inputs actor, action, and details. Flask sends these to the C core, which calculates a new hash and returns it. The record is saved with its Prev Hash and new Hash.
3. View Ledger: User lists all events in chronological order.
4. Verify Chain: User runs integrity verification. The system recalculates all hashes to confirm consistency.
5. Tamper Demonstration: For CS50, manual alteration of a record shows that verification detects changes.
6. Shutdown: Ledger closes gracefully, maintaining its state for the next run.

Step 4 – Security Model

Ledger enforces integrity and immutability:
- Append-only logic: data cannot be updated or deleted.

- Input validation: Flask ensures valid fields before submission.
- Isolation: Only Flask accesses the database; C handles computation only.
- Verification ensures any modification is immediately visible.
- Each verification result is also logged as a new event.

The prototype prioritizes traceability and proof of tampering, not confidentiality (added later in Phase 2).

Step 5 – Data Structure & Storage Design

Ledger stores events in a single table 'events' using SQLite (local) or PostgreSQL (future cloud use). Each entry represents a complete audit record with all event fields. Data access is append-only. Events can be read and verified but never modified or deleted.

Structure:
ID | Timestamp | Actor | Action | Details | Prev Hash | Hash | Verified

Benefits:
- Fast appends.
- Sequential verification.
- Simple export and portability for CS50 submission.

Step 6 – Verification Mechanism

Verification ensures the chain remains unaltered:
1. Retrieve all events in order.
2. Recalculate each event's hash from its content and stored Prev Hash.
3. Compare with the saved Hash value.
4. If mismatch occurs, flag the event as tampered.
5. Report overall result: 'Ledger verified successfully' or 'Chain broken at event #N'.

This method guarantees cryptographic integrity: even one changed byte causes a failure.

Step 7 – User Interface & Interaction Design

The Flask dashboard provides visual clarity for graders:

Sections:
- Add Event: input actor, action, details; automatic timestamp.
- Event Log: display chronological table of all events.
- Verify Chain: one button triggers integrity check.
- Status Area: shows total events, last verification, and chain validity (green/red indicator).

This design allows full demonstration of the ledger's functions with no technical complexity for the user.

Step 8 – Deliverables & Evaluation Checklist

Deliverables for CS50 submission:
- Local working prototype (no internet dependency).
- Short demo video showing valid chain and tamper detection.
- README explaining design, workflow, and verification logic.
- Project report (this document) describing architecture and security model.

Evaluation Criteria:
- Correctness of functionality (hash chain works).
- Design clarity and documentation quality.
- Security reasoning (immutability proof).
- Presentation (clean, understandable demonstration).

Step 9 – Phase 1 Wrap-Up & Transition Plan

Once Phase 1 is completed and submitted as your CS50 final project:
- You will migrate from SQLite to PostgreSQL.
- Deploy the Flask app to Azure App Service.
- Configure Coreventra subdomains for dashboard and API.
- Add HTTPS, JWT, and user authentication.
- Integrate post-quantum security (Dilithium/Kyber) in the C core.
- Connect Ledger to Coreventra and Network Lab modules for unified audit logging.

At this stage, Ledger will evolve from a local academic prototype into a professional-grade audit infrastructure.