Project Vision Document

Project Title: Hybrid Backup Service with Decentralized Sovereignty

Team Name: Team Mysten

Team Members:
- Neil Roy (neilroy@ucsb.edu)
- Kevin Lee (kjlee@ucsb.edu)
- Edwin Medrano Villela (eemedranovillela@ucsb.edu)
- Awin Zhang (awin@ucsb.edu)
- Suhrit Padakanti (suhrit@ucsb.edu)

Team Lead: Neil Roy
Team Scribe: Kevin Lee

Industry Partners:
Alberto Sonnino (alberto@mystenlabs.com)
Deepak Maram (deepak@mystenlabs.com)

Company Overview:
Mysten Labs is a Web3 infrastructure company focused on building Sui, a high-performance Layer-1 blockchain designed for scalability and mass adoption. Its technology emphasizes parallel transaction execution, scalable storage, and an object-centric data model, enabling developers to create next-generation decentralized applications across gaming, finance, and social platforms.

Problem Statement:
Traditional cloud backup services often force users into a trade-off between usability, security, and long-term accessibility. Even when encrypted storage is offered, users remain dependent on a single centralized provider. This creates risks:

- **Lock-in:** Providers may shut down, increase prices, or alter policies (e.g., reduce encryption guarantees), leaving users with limited options.
- **Regulatory pressure:** Centralized providers may be compelled to compromise security measures.
- **Data loss:** If the central service fails, users may permanently lose access to backups.

Project Overview:
This project aims to combine the convenience and performance of a centralized backup service with the resilience and sovereignty afforded by decentralized storage. By integrating Walrus (https://www.walrus.xyz) as a secondary backend, we enable users to retain control over their encrypted data, even if the centralized provider changes its policies or ceases operations.

Technical Approach:
The architecture introduces a hybrid storage model:

- **Centralized Layer:**
  - Provides fast-access caching to ensure a smooth user experience.
  - Handles bandwidth-heavy synchronization with Walrus, shielding end users from large data overhead (up to 4–5x bandwidth amplification).
  - Abstracts the complexity of blockchain interactions, managing WAL/SUI payments while offering common subscription models (credit card, etc.).
  - Automates blob lifecycle management (renewals, expirations) so users are not burdened with low-level storage concerns.
- **Decentralized Layer (Walrus):**
  - Stores encrypted user data as independently retrievable blobs.
  - Ensures long-term persistence, service portability, and user sovereignty.
  - Acts as the failsafe: if the centralized service fails or policies change, users (or new providers) can directly access data blobs

Expected Benefits:

- Full user control → Users can retrieve their encrypted data from Walrus at any time, independently of any service lock
- Provider flexibility → Rather than spending unnecessary resources on re-uploads, new providers can service using existing data stored in Walrus
- Usability without compromise → End-to-end encryption, high availability, and a smooth user experience are not compromised by the decentralized storage system via a centralized cache
- Sustainable ecosystem adoption → By handling operational challenges (payments, renewals, bandwidth optimization), the service makes decentralized storage viable for mainstream backup users.

Technologies:

- **Languages:** TypeScript (Node.js), Python (limited usage for quick scripts)
- **SDKs:** Walrus TypeScript SDK (@mysten/walrus)
- **Payments:** Suiet (@suiet/wallet-kit)
- **Infra:** Docker (reproducibility), GitHub (CI/CD)

Timeline:

Week 1 — Research & Setup

- Learn about decentralized storage and Walrus architecture
- Explore use cases and security properties (i.e. confidentiality, integrity, availability, sovereignty)
- Install and test the Walrus CL
- Store and retrieve test blobs

Week 2 — Basic Walrus Integration

- Build simple scripts or a minimal app to automate:
  - Blob upload and retrieval with Walrus
- Validate different file types and sizes
- Explore WAL/SUI payment

Week 3 — Encryption Layer

- Choose an encryption scheme (i.e. AES-GCM)
- Implement client-side encryption:
  - Encrypt uploads and decrypt downloads
- Ensure encryption is transparent to storage logic

Week 4 — Caching Layer

- Build a caching service on a centralized server:
  - Acts as a proxy between the client and Walrus
  - Caches uploaded/downloaded blobs
- Add the ability to optionally handle encryption in the caching layer (if not on the client)

Week 5 — Performance

- Log upload/retrieval performance for comparison
- [Challenge] Handle lazy upload (buffer locally, upload later)

Week 6 — Client Interface

- Implement client UX:
  - CLI (e.g. upload, download, status) *OR* simple web app (React/Vite or minimal Flask backend)
- Add config for selecting encryption and caching mode

Week 7 — Resilience & Direct Recovery

- Implement fallback: if cache is down, client retries via Walrus
- Validate user sovereignty: client can recover all blobs independently

Week 8 — Testing & Final Presentation

- Prepare a demo scenario:
  - Upload → encrypted → cache → Walrus
  - Kill cache → retrieve from Walrus
- Write the final report/documentation and record the demo

Process Model - Agile Methodology

- **Individuals and interactions** over processes and tools.
- **Working software** over comprehensive documentation.
- **Customer collaboration** over contract negotiation.
- **Responding to change** over following a plan

Conclusion:

This project aims to mitigate the drawbacks of centralized backup storage and decentralized sovereignty by implementing a hybrid storage model that enables full user sovereignty with Walrus while maintaining service portability to prevent costly reuploads. Ultimately, this project delivers a new backup solution that combines the simplicity and performance of mainstream backup services while embedding decentralized resilience at its core.