👁️ Project Monocle: System Design Document

Version: 1.0

Date: January 27, 2026

Status: Initial Draft

---

1. Executive Summary

Monocle is a cross-platform digital ecosystem designed to integrate real-time social interaction with a gamified community infrastructure. It moves beyond traditional messaging by incorporating a task-based economy, "sector-specific" community tools, and a dual-currency financial model.

---

2. Software Requirement Specification (SRS)

2.1 Functional Requirements (FR)

- **FR-1: Cross-Platform Accessibility:** The system must provide a unified user experience across Web, Android, iOS, Linux, and macOS.
- **FR-2: Sector-Based Communities:** Support for "Limitless Communities" with role-based access control and custom toolsets for different professions.
- **FR-3: Gamification Engine:** A server-side logic handler for Quests, Missions, and Events that updates user state in real-time.
- **FR-4: Dual-Token Economy:** * **Shards (Soft):** Earned via engagement; spent on social features.
    - **Orbs (Hard):** Purchased via IAP; spent on community registration and premium assets.
- **FR-5: Interactive Layer:** Integrated mini-games and activities accessible directly within the communication interface.

2.2 Non-Functional Requirements (NFR)

- **NFR-1: Scalability:** The backend must handle high-concurrency WebSocket connections using Go's goroutines.
- **NFR-2: Data Integrity:** Financial transactions must follow ACID properties (Atomicity, Consistency, Isolation, Durability).
- **NFR-3: Low Latency:** Messaging and state updates must have a P99 latency of <250ms.

---

3. System Architecture

3.1 Tech Stack

- **Frontend: Flutter** (Single codebase for Mobile, Desktop, and Web).
- **Backend: Golang** (High-performance microservices).
- **BaaS/Database: Supabase** (PostgreSQL for relational data, Auth, and Real-time subscriptions).
- **Communication: WebSockets** for real-time bidirectional data flow.

3.2 Data Model (Relational Schema)

The system relies on a relational model to ensure the economy remains balanced.

- **Users Table:** UID, Username, Bio, Profession_Tag.
- **Wallets Table:** User_ID, Shard_Balance, Orb_Balance.
- **Ledger Table:** Transaction_ID, User_ID, Amount, Type (Credit/Debit), Timestamp.
- **Communities Table:** ID, Owner_ID, Sector_Type, Member_Count.

---

4. Software Development Process

4.1 Methodology: Agile-DevOps

Monocle will utilize an **Agile** development cycle with 2-week Sprints, managed through GitHub Issues and Projects.

4.2 CI/CD Pipeline

1. **Version Control:** Git (GitHub).
2. **Continuous Integration:** GitHub Actions to run Go `go test` and Flutter `flutter test`.

3. **Continuous Deployment:** Automated containerization via Docker for the Go backend; Flutter Web hosted on global CDN.

🗺️ Monocle Development Roadmap (v1.1)

Phase 1: Infrastructure & Core Identity (Weeks 1-4)

Goal: Establish the "Walking Skeleton" of the application.

- **Backend (Go):** Initialize Go module, setup Supabase Auth integration, and create the middleware for JWT validation.
- **Frontend (Flutter):** Project initialization, theme configuration (using the new logo colors), and Auth flow (Login/Signup/Password Recovery).
- **DevOps:** Setup GitHub Actions for automated linting and unit testing.
- **Deliverable:** A functional cross-platform app where users can create accounts and log in.

Phase 2: Communication & Real-time Engine (Weeks 5-8)

Goal: Implement the "Limitless Communities" pillar.

- **Backend (Go):** Build the WebSocket hub using `goroutines` to handle concurrent message broadcasting.
- **Database (Supabase):** Design the `messages`, `channels`, and `communities` tables.
- **Frontend (Flutter):** Implement the Chat UI, real-time message state management (using BLoC or Riverpod), and "Sector" discovery page.
- **Deliverable:** Real-time messaging capability within different community sectors.

Phase 3: The Economic Ledger & Wallets (Weeks 9-12)

Goal: Deploy the digital ecosystem's financial layer.

- **Backend (Go):** Create the **Transaction Service**. All currency movements (Shards/Orbs) must be processed through Go to ensure server-side validation.
- **Database (Supabase):** Implement Postgres Functions and Triggers to maintain a strictly immutable `transaction_ledger`.
- **Frontend (Flutter):** Build the "Wallet" dashboard, transaction history view, and "Orb" purchase integration.

- **Deliverable:** A secure, functional in-app economy where users can hold and transfer balances.

Phase 4: Gamification & Engagement Layer (Weeks 13-16)

Goal: Build the "Quest" and "Mission" engine.

- **Backend (Go):** Develop the **Mission Logic Controller** (e.g., if message_count > 50, reward 10 Shards).
- **Frontend (Flutter):** UI for Quests, progress bars, and "Claim Reward" animations.
- **Mini-games:** Integration of the first basic mini-games (e.g., social betting or trivia) using the in-app currency as entry fees.
- **Deliverable:** A fully gamified ecosystem that rewards user activity.

Phase 5: Optimization & Scaling (Weeks 17+)

Goal: Hardening the system for a public launch.

- **Performance:** Load testing the Go server for 10k+ concurrent WebSocket connections.
- **Desktop/Web:** Fine-tuning the Flutter UI for mouse/keyboard interactions and larger screens.
- **Security:** Final audit of the currency logic to prevent exploits.

🛠️ Technical Debt & Maintenance

- **Documentation:** Generating API docs using Swagger/OpenAPI for the Go backend.
- **Monitoring:** Integrating Prometheus/Grafana to track server health.