Comprehensive Technical Specification: Self-Evolving Software Network (SESN)

1. Executive Summary

The Self-Evolving Software Network (SESN) is a dynamic, modular, AI-driven architecture designed to autonomously translate user requirements into entirely new software capabilities. This architecture ensures that the software is not only highly adaptable but also continuously self-develops based on changing user needs. All critical structural changes operate under a mandatory **Human-in-the-Loop (HITL)** supervision model to ensure security and stability.

2. The Three-Layer SESN Architectural Framework

The SESN architecture rigorously separates the responsibilities for data management, application logic, and the user interface.

Layer	Primary Function	Core Responsibility
Layer 1 (Data Core)	Standardized access to raw data	Basic APIs (Core Data Gates)
Layer 2 (Application Logic)	Defining application schema, permissions, and access gateway	Virtual APIs (AI-designed Gateways)
Layer 3 (User Interface)	User interaction and presentation	Microclients (AI-generated Front-End Units)

3. Data Core Layer: Basic Data Gates

This layer is the persistent storage core of the SESN, leveraging a flexible **NoSQL** database to manage the evolving schema.

3.1. NoSQL Nature and Base Primitives

- **Schema Flexibility:** The NoSQL nature allows the network to add new data structures at runtime without requiring complex physical migrations.
- **Base Primitives:** The NoSQL data core consists of a standardized set of fundamental data structures (e.g., string, number, date, reference, and lists) which the AI combines to construct Virtual Structures.

3.2. Basic Data Gates (BDGs)

All interactions at the core data level **must** pass through these four standardized gates without exception. This provides a single control point for **data consistency**, **atomic operations**, and **auditable state**.

Gate	Function	Key Logic Requirements
CREATE/UPDATE (CU)	Manages write operations (creation and modification)	Ensures operation atomicity and basic validation against Base Primitives.
READ (R)	Retrieves data documents	Supports standard filtering, pagination, and guarantees Read-After-Write consistency.
DELETE (D)	Manages deletion	Enforces a soft-deletion policy by setting an is_active: false flag to preserve records for auditing.
HISTORY (H)	Tracks changes	Records an immutable log of all CU/D operations, enabling auditing and rollback capabilities.

4. Interface and Application Logic Layer: Virtual APIs

This layer is where the AI translates user requirements into an operational application schema. The Virtual API is the most critical layer for responsibility decoupling within the SESN architecture.

4.1. Virtual Structure Definition and Role

The **Virtual API (Virtual Structure) absolutely does not store any data itself.** It serves solely as an **abstract, permissioned gateway.**

Primary functions of a Virtual API:

1. **Define Application Schema:** Uses Base Primitives to construct an application-specific schema (e.g., a complex task structure designed by the AI).

2. **Determine Permissions:** Explicitly embeds Access Control rules for every operation (CU, R, D), specifying which users or roles can use this access path.
3. **Traffic Direction:** Translates high-level Microclient requests into low-level calls directed at the Basic Data Gates.

4.2. Relationship Between Virtual API and Basic API

- **Indirect Access Path:** The Virtual API acts purely as a proxy, routing requests to the appropriate Basic Data Gates (CU/R/D/H). **Microclients must never consume Basic APIs directly.**
- **Reusability:** Any single Basic API (e.g., READ) is a core function that can be concurrently consumed by **multiple Virtual APIs**. This promotes development efficiency and ensures core data uniformity.

5. Client Layer and User Interaction

5.1. Microclient Generation and Coupling

A **Microclient** is a lightweight, front-end application unit entirely generated by the AI based on a specific Virtual API schema.

The Single-Focus Connection Rule:

- Each Microclient is **only** coupled to **one published Virtual Structure**.
- Microclients derive their rendering logic and functionality directly from the schema and permissions of their paired Virtual API.

5.2. Microclient Management Layer (Host Application)

This layer is a persistent shell responsible for providing a unified user experience. Its functions include:

- Managing navigation and the unified user environment (profiles, notifications).
- Managing the lifecycle and dynamic presentation of all deployed Microclients.
- Enabling the user to seamlessly switch between features, ensuring the user perceives SESN as a single, cohesive application.

6. Data Integrity and Human Supervision

6.1. Data Safety Against Virtual API Deletion (Critical Point)

Deleting a published Virtual API **in no way jeopardizes the integrity of the physical stored data.**

Technical Rationale:

- **Separation of Concerns:** The Virtual API is merely a **permissioned schema definition layer.**

- **Physical Data Preservation:** Physical data resides in the NoSQL collection, guarded by the Basic APIs.
- **Operational Outcome:** Deletion only removes the Virtual API's **definition, permissions, and access gateway**. Consequently, the connected Microclients become **inoperable** because their authorized access path no longer exists, but the underlying data remains intact and accessible via other active Virtual APIs or a newly deployed replacement.

6.2. The HITL Publication Process

To ensure stability and security, no new structure can connect to the live data network without final human approval:

1. **AI Design:** The AI creates the Virtual Structure and the Microclient code.
2. **Staging:** The structure and code are placed in a staging environment.
3. **Admin Review (HITL):** A System Administrator rigorously reviews and approves the embedded data schema and integrated permission rules.
4. **Publication:** Upon Admin approval, the structure is published, creating the necessary indexes and gateways, and thus enabling Microclients to interact with the underlying real data.