System Design Document

AI-Powered Personal SSOT

(Single Source of Truth)

Version: 2.0
Date: 2025-11-11
Author: Jorge Luis Contreras Herrera (Artificial Intelligence Developments)

1.0 Executive Summary

This document outlines the architecture for a multi-user, AI-Powered Single Source of Truth (SSOT). This system is designed to manage all tasks, projects, and knowledge for a creative developer. It integrates a personal "deep work" workflow with external stakeholder inputs (e.g., spouse, clients) into a single, unified pipeline.

The system's core function is to act as an "AI Operations Manager" that autonomously files, analyzes, and guards the user's workflow, protecting the "Create" (Deep Work) state and enforcing professional best practices (like PR-based development) automatically.

2.0 Core Philosophies & Methodologies

This SSOT is a hybrid system that blends several world-class methodologies, adapted to a developer-creator's workflow.

- **GTD (Getting Things Done):** Used for its rigorous, logistical **Clean** (Clarify) step ("Is this actionable?") and its **On Hold** (Someday/Maybe) concept.
- **Original 3-Path Logic:** All Items are triaged into one of three states:
    1. **Workflow (Kanban):** Actionable tasks.
    2. **Library (Searchable):** Non-actionable knowledge.
    3. **On Hold (Someday/Maybe):** Actionable-but-not-now tasks.
- **Kanban:** The visual UI/UX for managing the **Workflow**, visualizing flow, state, swimlanes, and enforcing the WIP=1 rule.
- **Deep Work (Cal Newport):** The core philosophy for the **Create** step. The system is architected to protect this state of cognitive focus.
- **PR-Based Workflow:** The "Done" trigger can optionally, not by default set to: Tasks are only completed via a git push that merges a Pull Request, not a simple commit. Default is manually moved to Done in the Kanban board.

3.0 User Personas & Clients

- **Persona 1: The Creator**
    - **Client:** Full-featured Desktop Web Application, and Mobile App with minimal functionalities.

    - **Permissions:** Full CRUD (Create, Read, Update, Delete).
    - **Goal:** Maximize productivity and manage concurrent projects.
- **Persona 2: The Stakeholders (e.g., Wife)**
    - **Client:** Lightweight Mobile App (PWA).
    - **Permissions:**
        - Create: Can create new Items which land *only* in the Creator's Inbox.
        - Read: Can read the status of any Item she created.
    - **Goal:** Assign tasks and get passive, real-time status updates. Input via text or voice.

4.0 The 5-Step Information Flow (Creator Workflow)

This is the primary workflow for the "Creator" persona.

1. **CAPTURE (Inbox):**
    - **Job:** Get ideas out of your head.
    - **Method:** Mobile-first, messy text input, or voice.
    - **System Action:** A new Item is created with status: 'Inbox'. Stakeholder tasks also land here with created_by_user_id: '[Stakeholder_ID]'.
2. **CLEAN (Daily Ritual):**
    - **Job:** Process the Inbox to zero. This is a *logistical* gate.
    - **Method:** For each Item in the Inbox:
        - **A. Clarify (GTD):** Ask "Is this actionable?"
            - **If NO (Info):** Update Item.status = 'Library'.
            - **If YES (Task):** Proceed to...
        - **B. Route:** Ask "Is this for *now*?"
            - **If NO (On-Hold):** Update Item.status = 'On Hold'.
            - **If YES (To Do):** Update Item.status = 'To Do'.
        - **C. Comment (Optional):** Write additional information or instructions for AI Filer.
        - **AI Filer (Highest Autonomy):** Parses natural language to auto-file tasks. Parses this to *autonomously* set swimlane: 'Expedite', priority: 'High', and labels: ['Job 1 (Income)', 'Project']. Uses comments or instructions, if available (e.g., "urgent bug").
    - **System Action:** The Item is saved. It now appears in the correct list (Workflow, Library, or Someday). The **"AI Librarian"** begins a background analysis. **AI Librarian** Attaches passive conflict/dependency/relation analysis.
3. **TRIAGE (Manual Prioritization):**
    - **Job:** Strategically select the *one* task for the next Deep Work session.
    - **Method: Expertise Decision.** The Creator reviews the Kanban board (all swimlanes and priorities).
    - **System Action:** User pulls *one* Item from a To Do column into the Create column. This is the explicit start signal.
4. **CREATE (Deep Work):**
    - **Job:** Execute the chosen task with zero distraction.
    - **Method:** The system enforces the **WIP=1** rule. The user works on the one task.

    - **System Action:** The Item.status is set to 'Create'. The AI Guardrail prevents any other Item from being moved to this column.
5. **Publish (Done):**
    - **Job:** The task is complete and verified.
    - **Optional Method (Automated):** For code tasks, triggered by **GitHub Webhooks**:
        1. **PR Opened:** Webhook moves Item from Create $\rightarrow$ **In Review**. This frees the Create (WIP=1) slot.
        2. **PR Merged:** Webhook moves Item from In Review $\rightarrow$ **Done**.
    - **Default Method (Manual):** The user manually drags the Item from Create $\rightarrow$ Done.
    - **System Action:** When status is set to Done, this change is reflected in the Stakeholder's app in real-time.

6.0 System Architecture & Tech Stack

This is a multi-client, API-driven, real-time application.

- **Clients:**
    - **Creator App:** Web Application (React/Next.js) with Kanban UI.
    - **Stakeholder App:** Progressive Web App (PWA).
- **Backend / Platform:**
    - **Recommendation:** Cloud Database.
    - **Why:** Provides PostgreSQL (structured data), Realtime (status sync), Auth (multi-user), and Serverless Functions (AI prompts & webhooks).
- **Database:**
    - **PostgreSQL** (Managed).
- **Integrations:**
    - **AI Model:** Best available API.
    - **VCS:** GitHub Webhooks.

7.0 Core Data Schema (PostgreSQL)

- **Users Table**
    - id (uuid, pk): ID.
    - email (text, unique): User's email.
    - role (enum: 'Creator', 'Stakeholder'): Defines permissions.
- **Items Table (The Atomic Unit)**
    - id (uuid, pk): Primary key.
    - human_id (text, unique): Short, human-readable ID for commit messages.
    - title (text): The title of the item.
    - raw_instructions (text): The original captured text.
    - type (enum: 'Task', 'Project', 'Info'): Set during Clean (Clarify).
    - status (enum: 'Inbox', 'To Do', 'Create', 'In Review', 'Blocked', 'Done', 'Library', 'On Hold', 'Cold Storage'): The primary state.
    - priority (enum: 'High', 'Medium', 'Low'): Set by AI Filer or user.

    - swimlane (enum: 'Expedite', 'Project', 'Habit', 'Home'): Set by AI Filer or user.
    - labels (text[]): An array of text labels (e.g., ['Job 1 (Income)', 'Latina']).
    - created_by_user_id (uuid, fk to Users): Tracks creator.
    - project_id (uuid, fk to Items): Self-referencing link to an Item of type: 'Project'.
    - created_at (timestampz)
- **AI_Analysis Table (for AI Librarian)**
    - id (uuid, pk)
    - item_id (uuid, fk to Items): The Item this suggestion is for.
    - analysis_type (enum: 'Conflict', 'Dependency', 'Redundancy', 'Related', 'Suggestion'):
    - analysis_text (text): The AI's finding.
    - is_read (boolean, default: false): Toggles the passive icon on the UI.
- **Rules & Messages Tables (for AI Guardrail)**
    - Stores the "Rules-as-Data" (e.g., WIP_Limit = 1) and the corresponding UI messages (e.g., "Your 'Create' slot is full.").
    - Superuser has dashboard to adjust all Rules.

8.0 AI Assistant Functionality

Function 1: AI Filer (Instant)

- **Trigger:** User finishes the Clean step on an Item.
- **Action:** A serverless function parses the Item.raw_instructions to infer swimlane, priority, and labels.
- **System Prompt:**

> "You are the 'Filer' AI for a project management SSOT. Your job is to parse a user's instruction and output a JSON object with the *inferred* swimlane, priority, and labels.
> **Mapping Rules:**
> - Swimlanes:
>   * Expedite: Urgent, external interrupts (e.g., "owner texted," "bug fix," "Wife task," "ASAP").
>   * Project: Standard project or deep work (e.g., "new feature," "architecture").
>   * Habit: Recurring personal tasks (e.g., "Data Science study").
>   * Home: Domestic tasks (e.g., "Buy milk").
> - Priorities:
>   * High: For Expedite or Home swimlanes.
>   * Medium: Default for Project tasks.
>   * Low: For Habit tasks.
> - Labels:
>   * If projects are mentioned, add Job 1 (Income).
>   * If relevant to AI Authority building, add Job 2 (Authority).

Input: (User Instructions as text)

Output: (Must be *only* JSON) {"swimlane": "...", "priority": "...", "labels": ["..."]}"

- **Result:** The Item is instantly saved to the DB with this data and appears on the Kanban board.

Function 2: AI Librarian (Background)

- **Trigger:** A new Item is successfully saved by the "AI Filer."
- **Action:** A background function (async) takes the new Item and compares it against *all other Items* in the database.
- **System Prompt:**

> "You are an AI Project Analyst. A new Item has been added. Analyze this New_Item (JSON) against the Corpus (an array of all other related Items).
> **Your Job:**
> 1. Identify Conflicts.
> 2. Identify Dependencies.
> 3. Identify Relations
> 4. Identify Redundancies.

Input: {"New_Item": {...}, "Corpus": [...]}
Output: (A JSON array of findings) [{"type": "Conflict", "text": "This conflicts with..."}, ...]"

- **Result:** The AI's findings are saved to the AI_Analysis table. This triggers a passive icon to appear on the Item's card.

8.0 Kanban Rules Guardrail (Real-time)

- **Trigger:** User action violates a rule (e.g., dragging a second task to Create).
- **Action: App Logic**, not an AI prompt. The app queries the Rules table and displays the corresponding Message (e.g., "You can't do that. Your 'Create' (WIP) slot is full.").
- **Result:** The user is coached in real-time, based on Kanban rules, which you can change in the database.

Professional Recommendation

Your familiarity with AWS is an advantage, as it means you can use its powerful, managed "primitive" services (like a database) without being locked into its higher-level, proprietary "platform" services.

The best architecture is:

1. The Database: PostgreSQL (via AWS RDS)

This is the most critical, "future-proof" choice.

- **Why PostgreSQL?** It is the most powerful, reliable, and extensible open-source relational database in the world. It has native support for JSON (for your Items), full-text search (for your Library), and vector embeddings (for your AI features, via pgvector).
- **Why AWS RDS?** You use **Amazon RDS (Relational Database Service)** to run your PostgreSQL instance. This gives you the best of both worlds:
    1. **No Vendor Lock-in:** You are using open-source PostgreSQL. If AWS triples its prices, you can take your database dump and move it to Google Cloud, Azure, or a private server tomorrow with zero changes to your application code.
    2. **Reliability:** You get all the "hassle" of AWS that you're used to (and that clients pay for): automated backups, high availability, read replicas, and security, all managed by Amazon.
- **Contrast with Supabase:** Supabase is *also* PostgreSQL, but you are locked into *their* platform, their tooling, and their pricing. By using AWS RDS, you use the *same core technology* but retain 100% architectural freedom.

2. The "BaaS" Replacement: Docker + Kubernetes

Since you are not using a BaaS, you must build the five components I mentioned (API, Auth, Real-time, Functions, etc.) yourself. The "future-proof" way to do this is with containers.

- **Technology: Docker** (to containerize your application) and **Kubernetes** (to manage and orchestrate your containers).
- **AWS Service:** You would run this on **Amazon EKS (Elastic Kubernetes Service)**.
- **The "Future-Proof" Advantage:** This is the ultimate "vendor-agnostic" architecture. You are building an application that runs *on* AWS, not *in* AWS. If you ever want to leave, you don't re-architect your app; you just point your Kubernetes configuration to a different cloud provider (like Google Kubernetes Engine or Azure Kubernetes Service).

Summary: The Trade-off

Criterion	**BaaS (Supabase)**	**Vendor-Agnostic (AWS+K8s)**
Speed to V1	**(Winner)** 1-2 weeks	3-6 months
Future-Proofing	Poor (High vendor lock-in)	**(Winner)** Excellent (Total freedom)
Reliability	Good (Managed by one team)	Excellent (Managed by you, built on AWS)
Complexity	Low	Extremely High

Given your goal of long-term reliability, the AWS RDS (PostgreSQL) + EKS (Kubernetes) stack is the technically superior, "future-proof" architecture. It is just a *dramatically* slower and more complex build.