**Core Integration Strategy**

The **Life Game Schema** revolves around user-driven gamified systems (users, characters, habits, quests), while the **SBS schema** revolves around system orchestration and automation pipelines (systems, routines, logs, orchestrator stages).  
They can be merged under a **shared meta-framework**: every “system” from SBS can be owned or driven by a “character” or “guild” in the Life Game, allowing both automation and gamification layers.

**Unified Database Overview**

| **Category** | **Key Tables** | **Integration Description** |
| --- | --- | --- |
| Accounts | users | Still the primary authentication entity for both the Life Game and SBS systems. |
| Systems & Automation | systems, system\_steps, routines, system\_templates, system\_logs | Extended with pointers to characters.id or guilds.id for ownership tracking across the Life Game. |
| Game Layer | characters, skills, habits, projects, tasks, events, guilds | Serve as conceptual “instances” of the SBS systems, e.g., a Habit could be implemented as an SBS routine. |
| Logs & Telemetry | system\_logs, systems\_log, events | Merge into a single event/audit system that tracks both game events and automation triggers. |
| Shared Metadata | JSONB metadata columns | Used across both schemas to extend automation or quest logic dynamically. |

**Example Integrated Table Modifications**

**1. Modified systems Table**

sql

**ALTER** **TABLE** systems

**ADD** **COLUMN** owner\_type **TEXT** **CHECK** (owner\_type IN ('user', 'character', 'guild')) **DEFAULT** 'user',

**ADD** **COLUMN** owner\_id **INT**,

**ADD** **FOREIGN** **KEY** (owner\_id) **REFERENCES** users(id);

This allows a **system** (SBS automation) to belong to a user, character, or guild.

**2. Link SBS Routines to Life Game Habits**

sql

**ALTER** **TABLE** routines

**ADD** **COLUMN** habit\_id **INT** **REFERENCES** habits(id),

**ADD** **COLUMN** trigger\_type **TEXT** **DEFAULT** 'scheduled' **CHECK** (trigger\_type IN ('manual', 'scheduled', 'event')),

**ADD** **COLUMN** active **BOOLEAN** **DEFAULT** TRUE;

Now, completing a habit can trigger an SBS routine or vice versa.

**3. Merge Log Systems**

Combine system\_logs and systems\_log into one standardized auditing system:

sql

**CREATE** **TABLE** unified\_logs (

id **SERIAL** **PRIMARY** **KEY**,

**timestamp** **TIMESTAMP** **DEFAULT** NOW(),

source **TEXT**,

system\_id **INT** **REFERENCES** systems(id),

character\_id **INT** **REFERENCES** characters(id),

user\_id **INT** **REFERENCES** users(id),

**action** **TEXT**,

detail JSONB,

outcome **TEXT**,

severity **TEXT** **DEFAULT** 'info'

);

This captures both **SBS workflow events** and **Life Game events** uniformly.

**New Cross-System Relationships**

* Every **Habit** can have a linked **Routine** (SBS automation).
* Every **Project (Quest)** can map to a **System Template**, making quests reusable automation blueprints.
* **Guilds** can own systems that execute routine automation for shared challenges.
* The **Events** table can ingest SBS webhook notifications using pg\_notify.

**Unified Database Relationship Diagram (Summary)**

* **users** — 1→1 — **characters**
* **characters** — 1→\* — **systems**
* **systems** — 1→\* — **system\_steps**, **routines**, **system\_logs**
* **routines** — optional —→ **habits**
* **projects** — optional —→ **system\_templates**
* **events** — tracks both **in-game** and **system** updates

**Suggested Integration Workflow**

1. **Unify Accounts**: use the same users authentication base for SBS and Life Game.
2. **Bridge Automation and Gameplay**: link SBS routines to habits and SBS systems to project templates/quests.
3. **Shared Logging**: maintain one consistent audit-log model (unified\_logs).
4. **Cross-Trigger Mechanism**:
   * Life Game events can trigger SBS workflows (via pg\_notify).
   * SBS results can write back to Life Game tables (e.g., XP gain, habit reward).
5. **Visualization Layer**: use n8n or Supabase dashboards to monitor both personal progress (Life Game) and automation health (SBS).

-- USERS table (Life Game)

CREATE TABLE users (

id SERIAL PRIMARY KEY,

email VARCHAR(255) UNIQUE NOT NULL,

username VARCHAR(80) UNIQUE NOT NULL,

avatar VARCHAR(255),

join\_date TIMESTAMP DEFAULT NOW(),

password\_hash VARCHAR(255),

theme VARCHAR(40) DEFAULT 'default'

);

-- CHARACTERS table (Life Game)

CREATE TABLE characters (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

class VARCHAR(32),

bio TEXT,

goals TEXT,

level INTEGER DEFAULT 1,

xp INTEGER DEFAULT 0,

hp INTEGER DEFAULT 100,

coins INTEGER DEFAULT 100,

prestige\_level INTEGER DEFAULT 0,

title VARCHAR(120),

last\_login TIMESTAMP

);

-- SYSTEMS table (SBS with owner integration)

CREATE TABLE systems (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

category TEXT,

purpose TEXT,

inputs TEXT,

outputs TEXT,

update\_frequency TEXT,

current\_stage TEXT DEFAULT 'define',

metadata JSONB DEFAULT '{}',

created\_at TIMESTAMP DEFAULT NOW(),

owner\_type TEXT CHECK (owner\_type IN ('user', 'character', 'guild')) DEFAULT 'user',

owner\_id INTEGER,

CONSTRAINT fk\_owner\_user FOREIGN KEY (owner\_id) REFERENCES users(id) ON DELETE SET NULL,

CONSTRAINT fk\_owner\_character FOREIGN KEY (owner\_id) REFERENCES characters(id) ON DELETE SET NULL

);

-- SYSTEM STEPS table (SBS)

CREATE TABLE system\_steps (

id SERIAL PRIMARY KEY,

system\_id INT REFERENCES systems(id) ON DELETE CASCADE,

step TEXT NOT NULL CHECK (step IN ('define', 'design', 'build', 'automate', 'review')),

status TEXT DEFAULT 'pending' CHECK (status IN ('pending', 'complete', 'blocked')),

notes TEXT,

metadata JSONB DEFAULT '{}',

updated\_at TIMESTAMP DEFAULT NOW()

);

-- ROUTINES table (SBS with link to habits)

CREATE TABLE routines (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

system\_id INT REFERENCES systems(id) ON DELETE CASCADE,

day\_of\_week TEXT,

description TEXT,

status TEXT DEFAULT 'active' CHECK (status IN ('active', 'paused', 'archived')),

metadata JSONB DEFAULT '{}',

habit\_id INTEGER REFERENCES habits(id) ON DELETE SET NULL,

trigger\_type TEXT DEFAULT 'scheduled' CHECK (trigger\_type IN ('manual', 'scheduled', 'event')),

active BOOLEAN DEFAULT TRUE

);

-- HABITS table (Life Game)

CREATE TABLE habits (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id) ON DELETE CASCADE,

skill\_id INTEGER REFERENCES skills(id) ON DELETE SET NULL,

name VARCHAR(100),

type VARCHAR(10) CHECK (type IN ('good', 'bad')),

xp\_value INTEGER DEFAULT 0,

hp\_value INTEGER DEFAULT 0,

streak INTEGER DEFAULT 0,

last\_completed DATE,

template\_id INTEGER REFERENCES habit\_templates(id) ON DELETE SET NULL

);

-- SKILLS table (Life Game)

CREATE TABLE skills (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id) ON DELETE CASCADE,

name VARCHAR(64),

xp INTEGER DEFAULT 0,

level INTEGER DEFAULT 1,

unlocked BOOLEAN DEFAULT FALSE

);

-- HABIT\_TEMPLATES table (Life Game)

CREATE TABLE habit\_templates (

id SERIAL PRIMARY KEY,

name VARCHAR(80),

skill\_name VARCHAR(64),

description TEXT

);

-- SYSTEM\_TEMPLATES table (SBS)

CREATE TABLE system\_templates (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

category TEXT,

description TEXT,

default\_inputs JSONB,

default\_outputs JSONB,

schema\_ref TEXT

);

-- SYSTEM\_LOGS table (SBS)

CREATE TABLE system\_logs (

id SERIAL PRIMARY KEY,

system\_id INT REFERENCES systems(id) ON DELETE CASCADE,

event TEXT NOT NULL,

details JSONB,

created\_at TIMESTAMP DEFAULT NOW()

);

-- UNIFIED\_LOGS table (Merged logs from SBS and Life Game for event auditing)

CREATE TABLE unified\_logs (

id SERIAL PRIMARY KEY,

timestamp TIMESTAMP DEFAULT NOW(),

source TEXT,

system\_id INT REFERENCES systems(id) ON DELETE SET NULL,

character\_id INT REFERENCES characters(id) ON DELETE SET NULL,

user\_id INT REFERENCES users(id) ON DELETE SET NULL,

action TEXT,

detail JSONB,

outcome TEXT,

severity TEXT DEFAULT 'info'

);

-- GUILDS table (Life Game)

CREATE TABLE guilds (

id SERIAL PRIMARY KEY,

name VARCHAR(100) UNIQUE,

description TEXT,

leader\_id INTEGER REFERENCES users(id),

xp\_pool INTEGER DEFAULT 0,

created TIMESTAMP DEFAULT NOW()

);

-- ROUTINE\_IN\_GUILDS: Allow routines to be owned or used by guilds

ALTER TABLE routines

ADD COLUMN guild\_id INTEGER REFERENCES guilds(id) ON DELETE SET NULL;

-- PROJECTS table (Life Game quests)

CREATE TABLE projects (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id) ON DELETE CASCADE,

area\_id INTEGER REFERENCES areas(id),

title VARCHAR(120),

description TEXT,

total\_xp INTEGER DEFAULT 0,

coin\_reward INTEGER DEFAULT 0,

difficulty VARCHAR(32),

deadline DATE,

completed BOOLEAN DEFAULT FALSE,

system\_template\_id INTEGER REFERENCES system\_templates(id) ON DELETE SET NULL -- Link quests to system templates

);

-- AREAS table (Life Game)

CREATE TABLE areas (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

name VARCHAR(50),

description TEXT

);

-- TASKS table (Life Game subquests)

CREATE TABLE tasks (

id SERIAL PRIMARY KEY,

project\_id INTEGER REFERENCES projects(id) ON DELETE CASCADE,

title VARCHAR(120),

completed BOOLEAN DEFAULT FALSE,

xp INTEGER DEFAULT 0,

coins INTEGER DEFAULT 0,

difficulty VARCHAR(32)

);

-- Additional entities such as inventory, items, events, messages, etc. can be linked similarly with references to the core users, characters, and systems as needed.

-- INDEXES for performance

CREATE INDEX idx\_systems\_owner ON systems(owner\_type, owner\_id);

CREATE INDEX idx\_system\_steps\_system\_id ON system\_steps(system\_id);

CREATE INDEX idx\_routines\_system\_id ON routines(system\_id);

CREATE INDEX idx\_routines\_habit\_id ON routines(habit\_id);

CREATE INDEX idx\_projects\_character\_id ON projects(character\_id);

CREATE INDEX idx\_unified\_logs\_timestamp ON unified\_logs(timestamp);

-- TRIGGER function example for notifying system updates

CREATE OR REPLACE FUNCTION notify\_system\_update()

RETURNS trigger AS $$

BEGIN

PERFORM pg\_notify('system\_update', row\_to\_json(NEW)::text);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER system\_update\_trigger

AFTER INSERT OR UPDATE ON systems

FOR EACH ROW EXECUTE FUNCTION notify\_system\_update();

**LifeOS: Unified Gamified Automation Platform**

**Executive Overview**

**LifeOS** integrates the **Life Game** progression model with the **SBS automation framework** into a single unified system that fuses **personal development**, **routine automation**, and **system orchestration**.  
The platform acts as a living ecosystem — blending gamification, behavioral tracking, and automation scripting into a single database, where every habit, workflow, or project becomes both a **quest** and a **system**.

**System Architecture Overview**

At its core, LifeOS is built on a **PostgreSQL** foundation designed for both **data normalization** and **cross-system reference integrity**.  
The schema merges SBS’s systems, system\_steps, and routines tables with Life Game’s users, characters, habits, and projects.

**Key Integration Principle:**  
Every human-facing entity (user, character, or guild) can own systems, while every system can automate game mechanics like habit streaks, skill progression, or quest completion.

**Core Functional Domains**

**1. User Management**

The **users** table combines authentication, theming, and relationship data.  
Each user can have:

* One **character** (primary identity in the gamified world)
* Multiple **guilds** (social connections)
* Optional **mentor-mentee** relations for guided growth
* Shared system access for collaborative automations

**2. Characters and Gamification**

**Characters** serve as avatars representing each user’s progress and data sandbox.  
They own **skills**, **habits**, **quests**, and even **SBS systems**.

Gamification metrics such as XP, coins, and habit streaks are tracked through:

* **skills** → exp-level progressions
* **habits** → behavioral XP / HP triggers
* **projects** → quest-like initiatives tied to automation templates
* **events** → real-time reflections of system updates or triggers

**3. Systems and Automation**

Borrowed from SBS, **systems** abstract autonomous processes (for instance, “Daily Review System” or “Budget Tracker”).  
Each system contains:

* **Steps** defining its progress through the lifecycle (define → design → build → automate → review)
* **Routines** representing repeated or scheduled behaviors
* **Triggers** that tie back into Life Game (XP awards, penalty events, log updates)

Each **system** can belong to:

* A **user** (personal automation)
* A **character** (in-game entity automation)
* A **guild** (shared collaborative automation)

**System Templates** act as blueprints for complex Life Game workflows like “Productivity Sprint” or “Habit Builder AB Test”.

**Automation Loop via n8n**

The platform extends automation through **n8n** (a self-hosted workflow automation tool).  
A PostgreSQL trigger (notify\_system\_update) broadcasts events via pg\_notify, which are received by n8n workflows.

This creates a **live automation loop**:

1. User completes a **habit** → record written to habits
2. Trigger updates linked **routine** → system log recorded to unified\_logs
3. n8n listens through webhook → executes automation (e.g., update XP or send Telegram message)
4. Result is persisted in system\_logs and reflected in **Life Game UI**

**Unified Data Model (Schema Summary)**

| **Functional Area** | **Core Tables** | **Relationships** |
| --- | --- | --- |
| Identity | users, characters, guilds, guild\_members | 1:1 between users and characters; M:N for guild membership |
| Automation | systems, system\_steps, routines, system\_templates | Each system belongs to a user, character, or guild; routines may link to habits |
| Gamified Behavior | habits, skills, projects, tasks, areas | Each builds streaks, skills, and quest progress linked to SBS routines |
| Logs & Analytics | system\_logs, unified\_logs, events, ai\_logs | Standardized format for tracking both automation and game events |
| Social Systems | friends, mentorship, messages | Peer motivation, AI reflections, and event collaboration |
| Assets | items, inventory, transactions | Mirror economy for in-game / reward structure integration |

**Unified Logging and Analytics**

The **unified\_logs** table harmonizes SBS system logs and Life Game event data, storing both manual and automated actions under one audit framework.  
Example entries could include:

* "action": "habit\_completed" → triggered XP increase via routine
* "action": "system\_advanced" → SBS advancement through lifecycle
* "source": "n8n" | "web" | "telegram\_bot" → identifies entry origin

With proper indexing on timestamps and JSONB data, the logs allow for comprehensive user insights, debugging visibility, and achievement validation.

**Automation-as-Gameplay Model**

Every automation event inside SBS becomes part of the Life Game’s world logic:

* Completing an SBS system step can issue **quest rewards**
* Missing a routine triggers **habit decay**
* Reaching an automation milestone unlocks **achievements**
* Telegram messages and dashboards serve as **NPC dialogues** or “system voices”

**Integration Flow Example**

1. **User Setup**
   * Creates an account
   * The system spawns a corresponding character and basic systems
2. **Habit Linkage**
   * User creates a habit (e.g., “Morning Routine”)
   * Links it to an automation routine that checks calendar data from n8n
3. **Automation**
   * n8n detects completion, triggers PostgreSQL event
   * The event advances habit streaks and pushes a reward via projects (quest completion)
4. **Feedback**
   * System logs store all state changes
   * Unified dashboard (in Supabase, Retool, or Notion API) displays progress analytics

**Technology Stack**

| **Component** | **Tool** | **Purpose** |
| --- | --- | --- |
| Database | PostgreSQL | Core unified schema |
| Automation Engine | n8n | Workflow orchestration and triggers |
| Messaging | Telegram Bot API | Notifications, status updates |
| Frontend | Supabase / React / Next.js | User & character dashboards |
| AI Assistant | OpenAI / Local LLM | Journaling, mentoring, reflective XP |
| Cloud Integration | Docker, Node.js | Deployment and scaling |

**Benefits of the Unified System**

* **Behavior Sync:** Habits directly drive system automations.
* **Gamified Motivation:** Daily systems fuel progression, XP, and quests.
* **Data Visibility:** Unified logs enable transparent debugging and AI feedback.
* **Automation Scalability:** Reusable templates and routines allow new systems to spawn and evolve with minimal setup.
* **Flexible Ownership:** Every entity (user, character, or guild) can run independent automations.
* **Cross-Platform Control:** Integrations through n8n and PostgreSQL events make it immediately compatible with external APIs or smart devices.

**Future Expansion**

* Integrate Google Drive and calendar sync through n8n workflows.
* Extend guild automation with shared project dashboards.
* Develop machine learning insight models on habit and system successes.
* Add “AI Mentor” who reacts to unified\_logs events and provides adaptive feedback.