**Gaming the System: Complete System Specification**

**Version:** 1.0  
**Last Updated:** October 27, 2025  
**Architecture:** Event-Driven Productivity Gamification + SBS Automation Engine

**Table of Contents**

1. Executive Overview
2. System Architecture
3. Database Schema
4. N8N Workflow Specifications
5. API Contracts & Webhooks
6. Deployment Guide
7. Testing & Validation

**Executive Overview**

**What is Gaming the System?**

Gaming the system merges a **gamified productivity application** with the **System for Building Systems (SBS)** automation engine into a unified platform where:

* **Users** create characters, complete habits/quests, and earn XP/coins
* **Systems** automate recurring workflows through a 5-stage lifecycle (Define → Design → Build → Automate → Review)
* **Events** drive real-time reactions using PostgreSQL pg\_notify
* **AI** generates personalized missions and prestige messages
* **Telegram Bot** provides conversational check-ins and system management

**Core Principles**

1. **Single Source of Truth:** One PostgreSQL database
2. **Event-Driven:** All changes trigger pg\_notify events consumed by n8n
3. **Modular Workflows:** Reusable n8n sub-workflows for common operations
4. **Unified Logging:** All actions logged to unified\_logs and system\_logs
5. **Polymorphic Ownership:** Systems can belong to users, characters, or guilds

**Key Statistics**

* **10 Core Game Workflows:** User setup, habits, quests, damage, prestige, etc.
* **5 SBS Workflows:** System spawner, orchestrator, routine engine, event listener, Telegram bot
* **30+ Database Tables:** Characters, habits, projects, systems, routines, inventory, etc.
* **2 Notification Channels:** system\_update, unified\_event

**System Architecture**

**High-Level Architecture**

┌─────────────────┐ ┌──────────────────┐ ┌─────────────────┐

│ Frontend App │────────▶│ PostgreSQL DB │────────▶│ pg-listener │

│ (Next.js/UI) │ │ (Single Source) │ │ (Node Service) │

└─────────────────┘ └──────────────────┘ └─────────────────┘

│ │

│ pg\_notify │ HTTP POST

▼ ▼

┌──────────────────┐ ┌─────────────────┐

│ Triggers/Events │────────▶│ N8N Workflows │

└──────────────────┘ │ (Orchestration)│

└─────────────────┘

│

┌─────────────────────────────┼─────────────────┐

│ │ │

┌──────▼──────┐ ┌────────▼────────┐ ┌─────▼─────┐

│ OpenAI API │ │ Telegram Bot │ │ Webhooks │

│ (Missions) │ │ (Interaction) │ │ (Frontend)│

└─────────────┘ └─────────────────┘ └───────────┘

**Component Responsibilities**

| **Component** | **Purpose** | **Technology** |
| --- | --- | --- |
| **PostgreSQL** | Data storage, event generation via triggers | PostgreSQL 15+ |
| **pg-listener** | Listens to pg\_notify, forwards to n8n webhooks | Node.js |
| **N8N** | Workflow orchestration, API calls, business logic | n8n (Docker) |
| **Frontend** | User interface, authentication, API calls | Next.js |
| **OpenAI** | AI mission generation, prestige messages | GPT-4 |
| **Telegram** | Bot notifications, conversational commands | Telegram Bot API |

**Event Flow Example**

1. User marks habit complete (Frontend → API)

2. Habit record updated in DB

3. PostgreSQL trigger fires: pg\_notify('unified\_event', habit\_data)

4. pg-listener receives notification

5. pg-listener POSTs to n8n webhook: /webhook/pg-notify

6. N8N HABIT\_CHECKIN workflow executes:

- Calculate rewards (XP, coins, streak)

- Update character stats

- Update skill XP

- Log to events table

- Log to systems\_log

- Return success response

7. Frontend receives success, updates UI

**Database Schema**

**Core Tables Overview**

| **Category** | **Tables** | **Purpose** |
| --- | --- | --- |
| **Authentication** | users | User accounts, credentials, settings |
| **Game Entities** | characters, skills, habits, projects, tasks, areas | Core gameplay mechanics |
| **Economy** | items, inventory, transactions | Shop, items, coins |
| **Social** | guilds, guild\_members | Multiplayer features |
| **SBS Automation** | systems, system\_steps, routines, system\_templates, system\_logs | Lifecycle automation |
| **Logging** | unified\_logs, events, ai\_logs | Audit trail, analytics |
| **Content** | rng\_events, achievements, journal | Dynamic content, progression |

**Complete Schema**

-- ============================================================

-- CORE SCHEMA: LifeOS Database

-- PostgreSQL 15+ Required

-- Extensions: uuid-ossp, pgcrypto

-- ============================================================

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- ============================================================

-- AUTHENTICATION & USERS

-- ============================================================

CREATE TABLE IF NOT EXISTS users (

id SERIAL PRIMARY KEY,

email VARCHAR(255) UNIQUE NOT NULL,

username VARCHAR(80) UNIQUE NOT NULL,

avatar VARCHAR(255),

join\_date TIMESTAMP WITH TIME ZONE DEFAULT now(),

password\_hash VARCHAR(255) NULL,

theme VARCHAR(40) DEFAULT 'default',

cloud\_sync\_token VARCHAR(128) NULL,

total\_prestiges INTEGER DEFAULT 0,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- ============================================================

-- GAME ENTITIES

-- ============================================================

CREATE TABLE IF NOT EXISTS 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 BIGINT DEFAULT 0,

total\_xp BIGINT DEFAULT 0,

hp INTEGER DEFAULT 100,

max\_hp INTEGER DEFAULT 100,

coins INTEGER DEFAULT 100,

prestige\_level INTEGER DEFAULT 0,

xp\_multiplier DECIMAL(3,2) DEFAULT 1.00,

title VARCHAR(120),

last\_login TIMESTAMP WITH TIME ZONE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS skills (

id SERIAL PRIMARY KEY,

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

name VARCHAR(64) NOT NULL,

xp BIGINT DEFAULT 0,

level INTEGER DEFAULT 1,

unlocked BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS habit\_templates (

id SERIAL PRIMARY KEY,

name VARCHAR(80),

skill\_name VARCHAR(64),

description TEXT

);

CREATE TABLE IF NOT EXISTS 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,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS areas (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

name VARCHAR(50),

description TEXT

);

CREATE TABLE IF NOT EXISTS 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,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS 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),

deadline DATE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- ============================================================

-- ECONOMY

-- ============================================================

CREATE TABLE IF NOT EXISTS items (

id SERIAL PRIMARY KEY,

name VARCHAR(64),

item\_type VARCHAR(32),

rarity VARCHAR(32),

description TEXT,

effect TEXT,

cost INTEGER DEFAULT 0

);

CREATE TABLE IF NOT EXISTS inventory (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

item\_id INTEGER REFERENCES items(id),

quantity INTEGER DEFAULT 1,

acquired TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS transactions (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

type VARCHAR(32),

amount INTEGER,

item\_id INTEGER REFERENCES items(id),

description TEXT,

trans\_date TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- ============================================================

-- SOCIAL

-- ============================================================

CREATE TABLE IF NOT EXISTS guilds (

id SERIAL PRIMARY KEY,

name VARCHAR(100) UNIQUE,

description TEXT,

leader\_id INTEGER REFERENCES users(id),

xp\_pool INTEGER DEFAULT 0,

created TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS guild\_members (

guild\_id INTEGER REFERENCES guilds(id),

user\_id INTEGER REFERENCES users(id),

joined TIMESTAMP WITH TIME ZONE DEFAULT now(),

is\_admin BOOLEAN DEFAULT FALSE,

PRIMARY KEY(guild\_id, user\_id)

);

-- ============================================================

-- SBS (SYSTEM FOR BUILDING SYSTEMS)

-- ============================================================

CREATE TABLE IF NOT EXISTS system\_templates (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

category TEXT,

description TEXT,

default\_inputs JSONB,

default\_outputs JSONB,

schema\_ref TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS 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 '{}'::jsonb,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

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

owner\_id INTEGER

);

CREATE TABLE IF NOT EXISTS 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 '{}'::jsonb,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS 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 '{}'::jsonb,

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,

guild\_id INTEGER REFERENCES guilds(id) ON DELETE SET NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS system\_logs (

id SERIAL PRIMARY KEY,

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

event TEXT NOT NULL,

details JSONB,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- ============================================================

-- LOGGING & AUDIT

-- ============================================================

CREATE TABLE IF NOT EXISTS unified\_logs (

id SERIAL PRIMARY KEY,

timestamp TIMESTAMP WITH TIME ZONE 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'

);

CREATE TABLE IF NOT EXISTS events (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

event\_type VARCHAR(50),

xp\_change INTEGER DEFAULT 0,

hp\_change INTEGER DEFAULT 0,

coins\_change INTEGER DEFAULT 0,

description TEXT,

event\_date TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS ai\_logs (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

message TEXT,

insight\_type VARCHAR(32),

timestamp TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- ============================================================

-- CONTENT & PROGRESSION

-- ============================================================

CREATE TABLE IF NOT EXISTS rng\_events (

id SERIAL PRIMARY KEY,

description TEXT,

effect TEXT,

rarity VARCHAR(32),

available BOOLEAN DEFAULT TRUE,

last\_issued DATE

);

CREATE TABLE IF NOT EXISTS achievements (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

title VARCHAR(100),

description TEXT,

reward\_type VARCHAR(32),

bonus\_value INTEGER,

unlocked\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS journal (

id SERIAL PRIMARY KEY,

character\_id INTEGER REFERENCES characters(id),

entry TEXT,

wisdom\_xp INTEGER DEFAULT 0,

entry\_date TIMESTAMP WITH TIME ZONE DEFAULT now()

);

CREATE TABLE IF NOT EXISTS settings (

user\_id INTEGER PRIMARY KEY REFERENCES users(id),

level\_xp\_formula TEXT,

overdraft\_rule TEXT,

notification\_times TEXT,

theme VARCHAR(32),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- ============================================================

-- TRIGGERS FOR EVENT-DRIVEN ARCHITECTURE

-- ============================================================

-- System Update Trigger

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;

DROP TRIGGER IF EXISTS systems\_notify\_trigger ON systems;

CREATE TRIGGER systems\_notify\_trigger

AFTER INSERT OR UPDATE ON systems

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

-- Unified Event Trigger

CREATE OR REPLACE FUNCTION notify\_unified\_event()

RETURNS trigger AS $$

DECLARE

payload json;

BEGIN

payload := row\_to\_json(NEW);

PERFORM pg\_notify('unified\_event', payload::text);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS habits\_notify\_trigger ON habits;

CREATE TRIGGER habits\_notify\_trigger

AFTER INSERT OR UPDATE ON habits

FOR EACH ROW EXECUTE FUNCTION notify\_unified\_event();

DROP TRIGGER IF EXISTS tasks\_notify\_trigger ON tasks;

CREATE TRIGGER tasks\_notify\_trigger

AFTER INSERT OR UPDATE ON tasks

FOR EACH ROW EXECUTE FUNCTION notify\_unified\_event();

-- ============================================================

-- INDEXES FOR PERFORMANCE

-- ============================================================

CREATE INDEX IF NOT EXISTS idx\_systems\_owner ON systems(owner\_type, owner\_id);

CREATE INDEX IF NOT EXISTS idx\_system\_steps\_system\_id ON system\_steps(system\_id);

CREATE INDEX IF NOT EXISTS idx\_routines\_system\_id ON routines(system\_id);

CREATE INDEX IF NOT EXISTS idx\_routines\_habit\_id ON routines(habit\_id);

CREATE INDEX IF NOT EXISTS idx\_projects\_character\_id ON projects(character\_id);

CREATE INDEX IF NOT EXISTS idx\_unified\_logs\_timestamp ON unified\_logs(timestamp);

CREATE INDEX IF NOT EXISTS idx\_characters\_user\_id ON characters(user\_id);

CREATE INDEX IF NOT EXISTS idx\_skills\_character\_id ON skills(character\_id);

CREATE INDEX IF NOT EXISTS idx\_habits\_character\_id ON habits(character\_id);

CREATE INDEX IF NOT EXISTS idx\_events\_character\_id ON events(character\_id);

**N8N Workflow Specifications**

**Workflow Catalog**

**Game Workflows**

| **ID** | **Name** | **Trigger** | **Purpose** |
| --- | --- | --- | --- |
| 1 | INIT\_USER\_SETUP | Webhook (user signup) | Create character, skills, tutorial quest |
| 2 | HABIT\_CHECKIN | Webhook (habit complete) | Award XP, coins, update streaks |
| 3 | DAMAGE\_CALC | Webhook (bad habit) | Calculate HP damage with defense modifiers |
| 4 | QUEST\_ENGINE | Webhook (task complete) | Grant rewards, check project completion |
| 5 | SHOP\_CHECK | Webhook (purchase) | Validate coins, deduct, add to inventory |
| 6 | CRON\_MANAGER | Schedule (daily) | Apply HP penalties, generate daily events |
| 7 | AI\_MISSIONS | Schedule (daily 6am) | Generate personalized missions via AI |
| 8 | ACHIEVEMENT\_UNLOCK | Webhook (milestone) | Check thresholds, grant achievements |
| 9 | EVENT\_SEEDER | Schedule (monthly) | Generate new random events via AI |
| 10 | PRESTIGE\_CALC | DB Trigger (level max) | Reset stats, add permanent bonuses |

**SBS Workflows**

| **ID** | **Name** | **Trigger** | **Purpose** |
| --- | --- | --- | --- |
| 11 | SBS\_SYSTEM\_SPAWNER | Webhook (system created) | Initialize lifecycle steps, routines |
| 12 | SBS\_SYSTEM\_ORCHESTRATOR | Webhook (system update) | Route to step handlers (define/design/build/automate/review) |
| 13 | SBS\_ROUTINE\_ENGINE | Schedule (daily 9am) | Execute due routines, send reminders |
| 14 | SBS\_PG\_LISTENER | Webhook (pg\_notify) | Forward DB events to appropriate workflows |
| 15 | SBS\_TELEGRAM\_BOT | Telegram (message) | Handle commands (/complete, /skip, /status, /help) |

**Reusable Modules**

All workflows leverage these shared modules:

| **Module** | **Used By** | **Purpose** |
| --- | --- | --- |
| **Reward Calculation** | HABIT\_CHECKIN, QUEST\_ENGINE, PRESTIGE\_CALC | Calculate XP, coins, streak multipliers |
| **Event Logging** | All workflows | Write to events and unified\_logs |
| **HP/XP Modifier** | DAMAGE\_CALC, CRON\_MANAGER, PRESTIGE\_CALC | Adjust character stats with validation |
| **Skill Update** | HABIT\_CHECKIN, QUEST\_ENGINE, PRESTIGE\_CALC | Increment skill XP, recalculate level |
| **Task Creation** | INIT\_USER\_SETUP, AI\_MISSIONS | Generate projects and tasks |

**API Contracts & Webhooks**

**N8N Webhook Endpoints**

All webhooks follow the pattern: https://your-n8n-domain.com/webhook/{endpoint}

| **Endpoint** | **Method** | **Purpose** | **Payload** |
| --- | --- | --- | --- |
| /webhook/user-signup | POST | Initialize new user | {user\_id, username, email, class, goals} |
| /webhook/habit-checkin | POST | Mark habit complete | {habit\_id, character\_id} |
| /webhook/bad-habit-battle | POST | Apply damage | {habit\_id, character\_id} |
| /webhook/complete-task | POST | Complete quest task | {task\_id, character\_id} |
| /webhook/shop/purchase | POST | Purchase item | {character\_id, item\_id, quantity} |
| /webhook/check-achievements | POST | Check for unlocks | {character\_id} |
| /webhook/pg-notify | POST | Receive DB events | {channel, payload} |
| /webhook/sbs-system-created | POST | Initialize system | {system\_id, name, category, purpose} |
| /webhook/sbs-system-update | POST | Advance system stage | {system\_id, current\_stage, name} |

**Response Formats**

**Success Response (Standard)**

{

"success": true,

"data": { /\* relevant data \*/ },

"message": "Operation completed successfully"

}

**Error Response (Standard)**

{

"success": false,

"error": "Error description",

"code": "ERROR\_CODE",

"details": { /\* optional additional context \*/ }

}

**Example Payloads**

**Habit Check-in Request**

{

"habit\_id": 42,

"character\_id": 13

}

**Habit Check-in Success Response**

{

"success": true,

"xpEarned": 30,

"coinsEarned": 15,

"newStreak": 7,

"streakBonus": 1.5,

"message": "Great work! Keep the momentum going!"

}

**Shop Purchase Request**

{

"character\_id": 13,

"item\_id": 5,

"quantity": 2

}

**Shop Purchase Error Response**

{

"success": false,

"error": "Insufficient coins",

"required": 500,

"available": 320,

"shortfall": 180

}

**Deployment Guide**

**Prerequisites**

* Docker & Docker Compose
* PostgreSQL 15+
* Node.js 18+ (for pg-listener)
* N8N account or self-hosted instance
* OpenAI API key
* Telegram Bot token (optional)

**Docker Compose Configuration**

version: '3.8'

services:

postgres:

image: postgres:15

restart: always

environment:

POSTGRES\_USER: lifeos\_app

POSTGRES\_PASSWORD: ${DB\_PASSWORD}

POSTGRES\_DB: lifeos\_db

volumes:

- pgdata:/var/lib/postgresql/data

- ./schema.sql:/docker-entrypoint-initdb.d/schema.sql

ports:

- "5432:5432"

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U lifeos\_app"]

interval: 10s

timeout: 5s

retries: 5

n8n:

image: n8nio/n8n:latest

restart: always

environment:

- DB\_TYPE=postgresdb

- DB\_POSTGRESDB\_HOST=postgres

- DB\_POSTGRESDB\_PORT=5432

- DB\_POSTGRESDB\_DATABASE=lifeos\_db

- DB\_POSTGRESDB\_USER=lifeos\_app

- DB\_POSTGRESDB\_PASSWORD=${DB\_PASSWORD}

- N8N\_HOST=0.0.0.0

- N8N\_PORT=5678

- N8N\_PROTOCOL=https

- WEBHOOK\_URL=${N8N\_WEBHOOK\_BASE\_URL}

- GENERIC\_TIMEZONE=America/Denver

- EXECUTIONS\_PROCESS=main

- OPENAI\_API\_KEY=${OPENAI\_API\_KEY}

- TELEGRAM\_BOT\_TOKEN=${TELEGRAM\_BOT\_TOKEN}

ports:

- "5678:5678"

depends\_on:

postgres:

condition: service\_healthy

volumes:

- n8n\_data:/home/node/.n8n

pg-listener:

build: ./pg-listener

restart: always

environment:

- DB\_HOST=postgres

- DB\_PORT=5432

- DB\_USER=lifeos\_app

- DB\_PASSWORD=${DB\_PASSWORD}

- DB\_NAME=lifeos\_db

- N8N\_WEBHOOK\_BASE\_URL=${N8N\_WEBHOOK\_BASE\_URL}

depends\_on:

postgres:

condition: service\_healthy

n8n:

condition: service\_started

adminer:

image: adminer

restart: always

ports:

- "8080:8080"

depends\_on:

- postgres

volumes:

pgdata:

n8n\_data:

**Environment Variables (.env)**

# Database

DB\_PASSWORD=your\_secure\_password\_here

# N8N

N8N\_WEBHOOK\_BASE\_URL=https://your-n8n-domain.com

# OpenAI

OPENAI\_API\_KEY=sk-your-openai-key-here

# Telegram (Optional)

TELEGRAM\_BOT\_TOKEN=123456:ABC-DEF...

TELEGRAM\_CHAT\_ID=your\_chat\_id

**pg-listener Service**

Create pg-listener/listener.js:

const { Client } = require('pg');

const fetch = require('node-fetch');

const client = new Client({

host: process.env.DB\_HOST,

port: process.env.DB\_PORT,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME

});

async function main() {

await client.connect();

console.log('✅ Connected to PostgreSQL');

client.on('notification', async (msg) => {

const channel = msg.channel;

const payload = JSON.parse(msg.payload);

console.log(`📢 Notification received: ${channel}`);

try {

const response = await fetch(`${process.env.N8N\_WEBHOOK\_BASE\_URL}/webhook/pg-notify`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ channel, payload })

});

if (response.ok) {

console.log(`✅ Forwarded to n8n: ${channel}`);

} else {

console.error(`❌ Failed to forward: ${response.statusText}`);

}

} catch (error) {

console.error(`❌ Error forwarding notification:`, error);

}

});

await client.query('LISTEN system\_update');

await client.query('LISTEN unified\_event');

console.log('👂 Listening to: system\_update, unified\_event');

}

main().catch(console.error);

Create pg-listener/Dockerfile:

FROM node:18-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

CMD ["node", "listener.js"]

Create pg-listener/package.json:

{

"name": "lifeos-pg-listener",

"version": "1.0.0",

"dependencies": {

"pg": "^8.11.0",

"node-fetch": "^2.6.9"

}

}

**Deployment Steps**

1. **Clone repository and configure**
2. git clone your-repo
3. cd lifeos
4. cp .env.example .env
5. # Edit .env with your credentials
6. **Start services**
7. docker-compose up -d
8. **Initialize database**
9. docker-compose exec postgres psql -U lifeos\_app -d lifeos\_db -f /docker-entrypoint-initdb.d/schema.sql
10. **Import n8n workflows**
    * Access n8n at http://localhost:5678
    * Go to Workflows → Import
    * Upload each JSON workflow from /n8n-workflows/ directory
11. **Configure n8n credentials**
    * PostgreSQL: lifeos\_app user
    * OpenAI API: Your API key
    * Telegram Bot: Your bot token
12. **Activate workflows**
    * Enable all imported workflows
    * Test with a simple webhook call
13. **Verify pg-listener**
14. docker-compose logs -f pg-listener
15. # Should show: "Listening to: system\_update, unified\_event"

**Testing & Validation**

**Unit Tests**

**Test 1: User Initialization**

curl -X POST http://localhost:5678/webhook/user-signup \

-H "Content-Type: application/json" \

-d '{

"user\_id": 1,

"username": "alice",

"email": "alice@example.com",

"class": "Starter",

"goals": "Get fit and learn new skills"

}'

**Expected Result:**

* Character created with ID
* 7 skills initialized (3 unlocked, 4 locked)
* Tutorial quest created with 3 tasks
* Default settings created
* Starter items added to inventory
* Events logged

**Test 2: Habit Check-in**

curl -X POST http://localhost:5678/webhook/habit-checkin \

-H "Content-Type: application/json" \

-d '{

"habit\_id": 1,

"character\_id": 1

}'

**Expected Result:**

* XP earned (base + streak multiplier)
* Coins earned
* Streak incremented
* Skill XP updated
* Character stats updated
* Event logged

**Test 3: Quest Completion**

curl -X POST http://localhost:5678/webhook/complete-task \

-H "Content-Type: application/json" \

-d '{

"task\_id": 1,

"character\_id": 1

}'

**Expected Result:**

* Task marked complete
* XP and coins awarded
* Related skill updated
* If all tasks done: project archived

**Test 4: System Creation (SBS)**

INSERT INTO systems (name, category, purpose, owner\_type, owner\_id)

VALUES ('Morning Routine', 'Habits', 'Automate morning habits', 'character', 1);

**Expected Result:**

* pg\_notify fired
* pg-listener forwards to n8n
* 5 system\_steps created (define complete, others pending)
* 2 default routines created (Monday, Friday)
* System stage advanced to 'design'
* Telegram notification sent

**Integration Tests**

**Test 5: Complete User Journey**

// 1. Create user

POST /webhook/user-signup

// 2. Create habit

INSERT INTO habits (character\_id, name, type, xp\_value, skill\_id)

// 3. Check in habit 7 days in a row

POST /webhook/habit-checkin (x7)

// 4. Verify streak bonus applied

GET character stats -> streak should be 7, multiplier 1.5x

// 5. Check achievement unlock

POST /webhook/check-achievements

// 6. Verify "Week Warrior" achievement granted

**Test 6: Shop Transaction Flow**

// 1. Check initial coins

GET character -> coins = 100

// 2. Attempt purchase beyond budget

POST /webhook/shop/purchase {item\_id: 1, quantity: 10}

// Expected: 400 error, insufficient coins

// 3. Valid purchase

POST /webhook/shop/purchase {item\_id: 1, quantity: 1}

// Expected: 200, inventory updated, coins deducted

// 4. Verify transaction logged

SELECT \* FROM transactions WHERE character\_id = 1

**Test 7: SBS Lifecycle Progression**

// 1. Create system

INSERT INTO systems (name, category, purpose, owner\_type, owner\_id)

// 2. Verify spawner ran

SELECT \* FROM system\_steps WHERE system\_id = NEW.id

// 3. Manually trigger orchestrator

POST /webhook/sbs-system-update {system\_id: 1, current\_stage: 'design'}

// 4. Verify design handler executed

SELECT \* FROM system\_logs WHERE event = 'design\_canvas\_generated'

// 5. Advance through all stages

POST /webhook/sbs-system-update (x4)

// 6. Verify completion

SELECT current\_stage FROM systems WHERE id = 1

// Expected: 'complete'

**Acceptance Criteria Checklist**

* [ ] User can sign up and character is created with default skills
* [ ] Habits can be checked in and rewards are calculated correctly
* [ ] Bad habits apply damage with defense modifiers
* [ ] Quests award XP and coins on task completion
* [ ] Shop validates coins and updates inventory
* [ ] Daily cron applies HP penalties for overdue habits
* [ ] AI generates personalized daily missions
* [ ] Achievements unlock at correct thresholds
* [ ] Prestige resets stats and applies permanent bonuses
* [ ] Systems progress through 5-stage lifecycle
* [ ] Routines execute on schedule and send reminders
* [ ] Telegram bot responds to commands correctly
* [ ] All events are logged to unified\_logs
* [ ] pg\_notify triggers are reliable

**N8N Workflow JSON Files**

**Directory Structure**

n8n-workflows/

├── game/

│ ├── 01\_INIT\_USER\_SETUP.json

│ ├── 02\_HABIT\_CHECKIN.json

│ ├── 03\_DAMAGE\_CALC.json

│ ├── 04\_QUEST\_ENGINE.json

│ ├── 05\_SHOP\_CHECK.json

│ ├── 06\_CRON\_MANAGER.json

│ ├── 07\_AI\_MISSIONS.json

│ ├── 08\_ACHIEVEMENT\_UNLOCK.json

│ ├── 09\_EVENT\_SEEDER.json

│ └── 10\_PRESTIGE\_CALC.json

├── sbs/

│ ├── 11\_SBS\_SYSTEM\_SPAWNER.json

│ ├── 12\_SBS\_SYSTEM\_ORCHESTRATOR.json

│ ├── 13\_SBS\_ROUTINE\_ENGINE.json

│ ├── 14\_SBS\_PG\_LISTENER.json

│ └── 15\_SBS\_TELEGRAM\_BOT.json

└── README.md

**Import Instructions**

1. **Access n8n Interface**
   * Navigate to https://your-n8n-domain.com
   * Login with credentials
2. **Import Workflows**
   * Click "Workflows" in left sidebar
   * Click "Import from File"
   * Select each JSON file from the appropriate directory
   * Repeat for all 15 workflows
3. **Configure Credentials**
   * Go to "Credentials" in left sidebar
   * Add PostgreSQL connection: lifeos\_app
   * Add OpenAI API: Your API key
   * Add Telegram Bot API: Your bot token
4. **Update Environment Variables**
   * Edit each workflow
   * Update N8N\_WEBHOOK\_BASE\_URL references
   * Update TELEGRAM\_CHAT\_ID references
5. **Activate Workflows**
   * Toggle each workflow to "Active"
   * Monitor execution logs for errors

**Detailed Workflow Specifications**

**1. INIT\_USER\_SETUP**

**Trigger:** Webhook POST /webhook/user-signup

**Input Payload:**

{

"user\_id": 1,

"username": "alice",

"email": "alice@example.com",

"class": "Starter",

"goals": "Get fit and learn new skills"

}

**Node Flow:**

1. **Webhook Trigger** → Receive signup data
2. **Create Character** → Insert into characters table
3. **Prepare Default Skills** → Define 7 skills array
4. **Split Skills** → Loop through each skill
5. **Insert Skills** → Create skill records
6. **Create Tutorial Quest** → Insert welcome project
7. **Prepare Tutorial Tasks** → Define 3 starter tasks
8. **Split Tasks** → Loop through each task
9. **Insert Tasks** → Create task records
10. **Create Settings** → Initialize user settings
11. **Prepare Starter Items** → Define inventory items
12. **Insert Items** → Add to inventory
13. **Log Events** → Write to events and unified\_logs
14. **Respond** → Return success with character\_id

**Key Logic:**

// Default Skills

const defaultSkills = [

{name: 'Health & Fitness', xp: 0, level: 1, unlocked: true},

{name: 'Career & Work', xp: 0, level: 1, unlocked: true},

{name: 'Finance & Wealth', xp: 0, level: 1, unlocked: true},

{name: 'Social & Relationships', xp: 0, level: 1, unlocked: false},

{name: 'Learning & Knowledge', xp: 0, level: 1, unlocked: false},

{name: 'Creativity & Arts', xp: 0, level: 1, unlocked: false},

{name: 'Mindfulness & Wisdom', xp: 0, level: 1, unlocked: false}

];

**2. HABIT\_CHECKIN**

**Trigger:** Webhook POST /webhook/habit-checkin

**Input Payload:**

{

"habit\_id": 42,

"character\_id": 13

}

**Node Flow:**

1. **Webhook Trigger** → Receive check-in request
2. **Fetch Habit Data** → Join habits, skills, characters
3. **Calculate Rewards** → Apply streak multipliers
4. **Check Already Completed** → Prevent double check-in
5. **Update Habit Streak** → Increment streak, set last\_completed
6. **Update Skill XP** → Add skill XP (40% of habit XP)
7. **Update Character** → Add XP, coins, recalculate level
8. **Log Event** → Write to events table
9. **Log System** → Write to unified\_logs
10. **Respond** → Return rewards earned

**Reward Calculation Logic:**

const baseXP = habitData.xp\_value || 10;

const baseCoins = Math.floor(baseXP \* 0.5);

const currentStreak = habitData.streak || 0;

let newStreak = currentStreak + 1;

let streakMultiplier = 1.0;

if (newStreak >= 7) streakMultiplier = 1.5;

else if (newStreak >= 30) streakMultiplier = 2.0;

else if (newStreak >= 90) streakMultiplier = 3.0;

const finalXP = Math.floor(baseXP \* streakMultiplier);

const finalCoins = Math.floor(baseCoins \* streakMultiplier);

const skillXP = Math.floor(finalXP \* 0.4);

**Level Calculation Formula:**

// Character Level = FLOOR(POWER(total\_xp / 100, 0.66)) + 1

// Skill Level = FLOOR(POWER(skill\_xp / 100, 0.5)) + 1

**3. DAMAGE\_CALC**

**Trigger:** Webhook POST /webhook/bad-habit-battle

**Input Payload:**

{

"habit\_id": 15,

"character\_id": 13

}

**Node Flow:**

1. **Webhook Trigger** → Receive battle request
2. **Fetch Data** → Join habit, character, skill data
3. **Calculate Damage** → Apply defense modifiers
4. **Update HP** → Reduce character HP (min 0)
5. **Update Timestamp** → Mark habit last\_completed
6. **Log Battle** → Write to events table
7. **Log System** → Write to unified\_logs
8. **Respond** → Return damage dealt, narrative

**Damage Calculation Logic:**

const baseDamage = Math.abs(habitData.hp\_value) || 15;

const skillLevel = habitData.skill\_level || 1;

// Higher skill = better defense

const defenseModifier = Math.max(0.5, 1 - (skillLevel \* 0.05));

const finalDamage = Math.floor(baseDamage \* defenseModifier);

const newHP = Math.max(0, currentHP - finalDamage);

const isDefeated = newHP === 0;

let battleNarrative = `You faced the ${habitData.name} and took ${finalDamage} damage!`;

if (isDefeated) {

battleNarrative += " You've been defeated and must recover at the Hotel.";

} else if (newHP < 30) {

battleNarrative += " Your HP is critically low!";

}

**4. QUEST\_ENGINE**

**Trigger:** Webhook POST /webhook/complete-task

**Input Payload:**

{

"task\_id": 7,

"character\_id": 13

}

**Node Flow:**

1. **Webhook Trigger** → Receive task completion
2. **Fetch Task Data** → Join task, project, character, area
3. **Calculate Rewards** → Apply difficulty multipliers
4. **Mark Complete** → Set task.completed = true
5. **Find Related Skill** → Lookup skill by area
6. **Update Skill XP** → Add skill experience
7. **Update Character** → Add XP and coins
8. **Check Project** → Count remaining tasks
9. **If Complete** → Mark project done, archive
10. **Log Event** → Write to events
11. **Log System** → Write to unified\_logs
12. **Respond** → Return rewards and completion status

**Difficulty Multipliers:**

const difficultyMultipliers = {

'easy': 1.0,

'tutorial': 1.0,

'medium': 1.5,

'hard': 2.0,

'epic': 3.0,

'legendary': 5.0

};

const multiplier = difficultyMultipliers[difficulty] || 1.5;

const finalXP = Math.floor(baseXP \* multiplier);

// Time bonus: 20% extra for on-time completion

if (completedBeforeDeadline) {

timeBonus = Math.floor(finalXP \* 0.2);

}

**5. SHOP\_CHECK**

**Trigger:** Webhook POST /webhook/shop/purchase

**Input Payload:**

{

"character\_id": 13,

"item\_id": 5,

"quantity": 2

}

**Node Flow:**

1. **Webhook Trigger** → Receive purchase request
2. **Parse Request** → Extract parameters
3. **Fetch Item** → Get item details and cost
4. **Fetch Character** → Get current coins
5. **Validate Purchase** → Calculate total cost, check affordability
6. **If Affordable:**
   * Deduct coins from character
   * Add item to inventory (or increment quantity)
   * Log transaction
   * Log event
   * Log system
   * Return success
7. **If Not Affordable:**
   * Log failure
   * Return error with shortfall

**Transaction Logging:**

// transactions table

{

character\_id: 13,

type: 'spend',

amount: totalCost,

item\_id: 5,

description: 'Purchased 2x Health Potion',

trans\_date: NOW()

}

**6. CRON\_MANAGER**

**Trigger:** Schedule (Daily at midnight)

**Node Flow:**

1. **Schedule Trigger** → Cron: 0 0 \* \* \*
2. **Fetch Active Characters** → Last login within 30 days
3. **Check Overdue Habits** → Good habits not completed in 2+ days
4. **Check Overdue Tasks** → Tasks past deadline
5. **Calculate Penalties** → HP deductions based on overdue items
6. **Update Character HP** → Apply penalties or bonuses
7. **Log Events** → Write daily maintenance events
8. **Fetch Available Events** → Get RNG events not issued recently
9. **Generate Daily Events** → Randomly assign 1-3 events per character
10. **Insert Events** → Write to events table, update last\_issued
11. **Reset Broken Streaks** → Set streak = 0 for habits >2 days old
12. **Log Streak Breaks** → Write streak break events
13. **Log System** → Summary of daily run

**Penalty Calculation:**

// Penalty for overdue habits: 2 HP per habit (max 20)

const habitPenalty = Math.min(overdueHabits \* 2, 20);

// Penalty for overdue tasks: 5 HP per task (max 25)

const taskPenalty = Math.min(overdueTasks \* 5, 25);

// Penalty for negative coins (overdraft): 10% of debt (max 15)

const overdraftPenalty = Math.min(Math.abs(coins) \* 0.1, 15);

// Daily wellness bonus for active high-level players

if (level >= 10 && hpPenalty === 0) {

hpBonus = 5;

}

const netHpChange = hpBonus - (habitPenalty + taskPenalty + overdraftPenalty);

**7. AI\_MISSIONS**

**Trigger:** Schedule (Daily at 6 AM)

**Node Flow:**

1. **Schedule Trigger** → Cron: 0 6 \* \* \*
2. **Fetch Active Users** → Last login within 7 days
3. **For Each User:**
   * Fetch top 3 skills
   * Fetch recent activity (last 7 days)
   * Fetch habit streaks
   * Fetch active projects
4. **Prepare AI Context** → Combine all user data
5. **Generate Missions (AI)** → Call OpenAI GPT-4
6. **Parse Response** → Extract JSON missions array
7. **Find/Create Missions Area** → Ensure "Daily Missions" area exists
8. **Create Mission Projects** → Insert projects with 1-day deadline
9. **Create Mission Tasks** → Insert corresponding tasks
10. **Log AI Generation** → Write to ai\_logs
11. **Log Events** → Write mission creation events
12. **Log System** → Record AI run summary

**AI Prompt Template:**

You are a wise AI companion for {username} in their personal growth journey.

Player Profile:

- Class: {class}

- Level: {level}

- Goals: {goals}

Top Skills:

- Health: Level 5 (500 XP)

- Work: Level 3 (250 XP)

Recent Activity (Last 7 Days):

- habit\_completed: 5 times (75 XP earned)

- task\_completed: 3 times (120 XP earned)

Generate 3 personalized daily missions. Each mission should:

1. Be specific and actionable

2. Align with their goals

3. Challenge them appropriately for their level

4. Include XP reward (10-50)

5. Include coin reward (5-25)

6. Relate to one of their top skills

Respond ONLY with JSON array:

[

{

"title": "Morning Meditation Session",

"description": "Practice mindfulness for 10 minutes",

"xp": 25,

"coins": 10,

"difficulty": "medium",

"skill\_name": "Mindfulness"

}

]

**8. ACHIEVEMENT\_UNLOCK**

**Trigger:** Webhook POST /webhook/check-achievements

**Input Payload:**

{

"character\_id": 13

}

**Node Flow:**

1. **Webhook Trigger** → Receive check request
2. **Fetch Character Stats** → Aggregate habits, projects, level, XP
3. **Fetch Skill Stats** → Max level, level 5+ skills, level 10+ skills
4. **Fetch Habit Stats** → Max streak, 30-day streaks, 90-day streaks
5. **Fetch Wealth Stats** → Total coins earned
6. **Fetch Existing Achievements** → Already unlocked titles
7. **Check Criteria** → Evaluate 30+ achievement definitions
8. **If New Achievements:**
   * Insert achievement records
   * Apply rewards (XP or coins)
   * Log events
   * Log system
   * Aggregate results
   * Return achievements list
9. **If No New:**
   * Return empty list

**Achievement Definitions (Sample):**

const achievements = [

// Level Milestones

{

title: 'Novice Adventurer',

condition: level >= 5,

reward\_type: 'xp',

bonus\_value: 100,

description: 'Reached Level 5'

},

{

title: 'Legendary Hero',

condition: level >= 50,

reward\_type: 'coins',

bonus\_value: 5000,

description: 'Reached Level 50'

},

// Streak Achievements

{

title: 'Week Warrior',

condition: maxStreak >= 7,

reward\_type: 'xp',

bonus\_value: 75,

description: 'Maintained a 7-day streak'

},

{

title: 'Year of Discipline',

condition: maxStreak >= 365,

reward\_type: 'coins',

bonus\_value: 10000,

description: 'Maintained a 365-day streak'

},

// Skill Mastery

{

title: 'Renaissance Soul',

condition: level5Skills >= 3,

reward\_type: 'coins',

bonus\_value: 300,

description: 'Have 3 skills at level 5+'

}

];

**9. EVENT\_SEEDER**

**Trigger:** Schedule (Monthly on 1st)

**Node Flow:**

1. **Schedule Trigger** → Cron: 0 0 1 \* \*
2. **Check Event Pool** → Count total and available events
3. **Generate Events (AI)** → Call OpenAI for 20 new events
4. **Parse and Validate** → Extract JSON, validate structure
5. **Insert Events** → Add to rng\_events table
6. **Retire Old Events** → Mark events not used in 60 days as unavailable
7. **Generate Seasonal Events** → Create month-specific events
8. **Insert Seasonal** → Add seasonal events
9. **Cleanup Old Seasonal** → Remove past seasonal events
10. **Analyze Distribution** → Count events by rarity
11. **Create Summary** → Aggregate statistics
12. **Log System** → Record seeder run
13. **Notify Users** → Send "new events available" notification

**AI Prompt:**

Generate 20 diverse random events for a life-gamification app.

Event Pool Status:

- Total events: 150

- Currently available: 120

Create events that are:

1. Diverse (positive, negative, neutral, mysterious)

2. Varied in rarity (common, uncommon, rare, legendary)

3. Thematic (personal growth, productivity)

4. Fun and engaging

5. Include specific effects (HP, XP, coins, bonuses)

Categories:

- Fortune (lucky finds, bonus rewards)

- Misfortune (setbacks, penalties)

- Mystery (random outcomes)

- Wisdom (reflective, philosophical)

- Social (community interactions)

- Seasonal (time of year)

Respond with JSON array:

[

{

"description": "Found a lucky coin while cleaning!",

"effect": "+15 coins",

"rarity": "common"

}

]

**Seasonal Event Generation:**

const month = new Date().getMonth() + 1;

const seasons = {

1: {name: 'New Year', theme: 'fresh starts and resolutions'},

3: {name: 'Spring', theme: 'renewal and growth'},

6: {name: 'Summer', theme: 'outdoor adventures'},

9: {name: 'Autumn', theme: 'harvest and preparation'},

12: {name: 'Winter Holiday', theme: 'celebration and reflection'}

};

const currentSeason = seasons[month];

// Generate 3 season-specific events

**10. PRESTIGE\_CALC**

**Trigger:** Database Trigger (Level reaches max\_level)

**Input:** Triggered automatically when characters.level >= max\_level

**Node Flow:**

1. **Trigger** → PostgreSQL trigger on characters UPDATE
2. **Fetch Character & User** → Join characters and users tables
3. **Fetch Current Skills** → Get all skills with XP
4. **Calculate Prestige Bonus** → Determine rewards based on prestige level
5. **Generate AI Message** → Create epic celebration message and title
6. **Parse AI Response** → Extract title, message, quote
7. **Reset Character Stats** → Level = 1, XP = 0, HP increased, multiplier increased
8. **Update User** → Increment total\_prestiges, add coin bonus
9. **Reset Skills** → Keep 10% of skill XP, reset to level 1
10. **Grant Achievement** → Add prestige achievement with AI title
11. **Add Token** → Insert prestige token to inventory
12. **Log Event** → Write prestige unlock event
13. **Notify Frontend** → Trigger celebration animation
14. **Output Summary** → Return prestige details

**Prestige Bonus Formula:**

const prestigeLevel = currentPrestigeCount + 1;

const prestigeBonus = {

hp\_bonus: 10 \* prestigeLevel, // +10 HP per prestige

xp\_multiplier: 1 + (0.05 \* prestigeLevel), // +5% XP per prestige

coin\_bonus: 100 \* prestigeLevel, // +100 coins per prestige

prestige\_level: prestigeLevel,

permanent\_perk: `prestige\_${prestigeLevel}`

};

// Skill Retention: Keep 10% of XP

const retainedXP = Math.floor(currentSkillXP \* 0.1);

**AI Prestige Message Prompt:**

Generate a prestige celebration message and title for {username} who has reached Prestige Level {level}.

Pre-prestige stats:

- Level {level}

- {total\_xp} total XP

- {skills\_count} skills mastered

Make it epic and motivational.

Return JSON:

{

"title": "Eternal Ascendant",

"message": "You have transcended mortal limits...",

"quote": "Every end is a new beginning"

}

**SBS (System for Building Systems) Workflows**

**11. SBS\_SYSTEM\_SPAWNER**

**Trigger:** Webhook POST /webhook/sbs-system-created

**Input Payload:**

{

"system\_id": 42,

"name": "Morning Routine System",

"category": "Habits",

"purpose": "Automate morning habits"

}

**Node Flow:**

1. **Webhook Trigger** → Receive system creation event
2. **Validate Input** → Ensure system\_id exists
3. **Insert Lifecycle Steps** → Create 5 steps (define, design, build, automate, review)
4. **Create Default Routines** → Insert 2 routines (Monday, Friday)
5. **Advance Stage** → Update systems.current\_stage to 'design'
6. **Log Event** → Write system\_spawned to system\_logs
7. **Send Notification** → Telegram message with system details
8. **Respond** → Return success with next\_stage

**Lifecycle Steps Created:**

INSERT INTO system\_steps (system\_id, step, status, notes) VALUES

(42, 'define', 'complete', 'System creation completed'),

(42, 'design', 'pending', 'Design system architecture'),

(42, 'build', 'pending', 'Build working components'),

(42, 'automate', 'pending', 'Add triggers and schedules'),

(42, 'review', 'pending', 'Schedule review cycle');

**Default Routines:**

INSERT INTO routines (name, system\_id, day\_of\_week, description, status)

VALUES

('Morning Routine System - Monday', 42, 'Monday', 'Auto-generated routine', 'active'),

('Morning Routine System - Friday', 42, 'Friday', 'Auto-generated routine', 'active');

**12. SBS\_SYSTEM\_ORCHESTRATOR**

**Trigger:** Webhook POST /webhook/sbs-system-update

**Input Payload:**

{

"system\_id": 42,

"current\_stage": "design",

"name": "Morning Routine System"

}

**Node Flow:**

1. **Webhook Trigger** → Receive system update
2. **Get Current Step** → Fetch next pending step from system\_steps
3. **Route by Step:**
   * **Design** → Generate design canvas, store in system\_logs
   * **Build** → Send Telegram notification, create infrastructure
   * **Automate** → Configure triggers, add to system\_logs
   * **Review** → Schedule review cycle, log next review date
4. **Mark Step Complete** → Update system\_steps.status = 'complete'
5. **Update System Stage** → Advance to next pending step
6. **Respond** → Return success with new stage

**Step Handlers:**

**Design Handler**

// Generate design canvas markdown

INSERT INTO system\_logs (system\_id, event, details)

VALUES (42, 'design\_canvas\_generated', {

name: 'Morning Routine System',

timestamp: NOW(),

canvas\_template: 'markdown'

});

**Build Handler**

// Send Telegram notification

POST https://api.telegram.org/bot{token}/sendMessage

{

chat\_id: "{CHAT\_ID}",

text: "🔧 \*Build Phase Started\*\n\nSystem: \*Morning Routine System\*\n\n📁 Creating folders and database schemas..."

}

**Automate Handler**

// Log automation configuration

INSERT INTO system\_logs (system\_id, event, details)

VALUES (42, 'automation\_configured', {

triggers\_added: true,

schedules\_created: true,

timestamp: NOW()

});

**Review Handler**

// Schedule next review

INSERT INTO system\_logs (system\_id, event, details)

VALUES (42, 'review\_scheduled', {

next\_review: NOW() + INTERVAL '30 days',

review\_frequency: 'monthly',

timestamp: NOW()

});

**13. SBS\_ROUTINE\_ENGINE**

**Trigger:** Schedule (Daily at 9 AM)

**Node Flow:**

1. **Schedule Trigger** → Cron: 0 9 \* \* \*
2. **Get System Stats** → Count active systems, categories, stages
3. **Send Daily Summary** → Telegram overview message
4. **Get Today's Routines** → Fetch routines matching current day\_of\_week
5. **Check Routines Exist** → Branch: yes/no
6. **If Routines Exist:**
   * Split into individual items
   * For each routine:
     + Send Telegram reminder with routine details
     + Log reminder sent to system\_logs
   * Aggregate results
   * Send summary message
7. **If No Routines:**
   * Send "no routines today" message

**Daily Summary Message:**

📊 \*SBS System Summary\*

🔧 Active Systems: 5

🗂️ Categories: 3

⚙️ Stages in Progress: design, build, automate

\_Daily routine check initiated...\_

**Routine Reminder Message:**

📅 \*Daily Routine Reminder\*

\*System:\* Morning Routine System

\*Category:\* Habits

\*Routine:\* Morning Page

📝 Write 3 pages of stream-of-consciousness

\*Current Stage:\* automate

✅ Reply with `/complete 15` when done

⏭️ Reply with `/skip 15` to skip today

**14. SBS\_PG\_LISTENER**

**Trigger:** Webhook POST /webhook/pg-notify

**Input Payload:**

{

"channel": "system\_update",

"payload": {

"id": 42,

"name": "Morning Routine System",

"current\_stage": "define",

"owner\_type": "character",

"owner\_id": 13,

"created\_at": "2025-10-27T13:12:00Z"

}

}

**Node Flow:**

1. **Webhook Trigger** → Receive pg\_notify forwarded from pg-listener
2. **Parse Payload** → Extract and parse JSON
3. **Check Stage:**
   * **If 'define'** → Trigger System Spawner
   * **If 'complete'** → Notify completion, log
   * **Otherwise** → Trigger System Orchestrator
4. **Respond** → Acknowledge receipt

**pg-listener Service Logic:**

// This runs as a separate Node.js service

client.on('notification', async (msg) => {

const channel = msg.channel; // 'system\_update' or 'unified\_event'

const payload = JSON.parse(msg.payload);

// Forward to n8n

await fetch(`${N8N\_WEBHOOK\_BASE\_URL}/webhook/pg-notify`, {

method: 'POST',

headers: {'Content-Type': 'application/json'},

body: JSON.stringify({channel, payload})

});

});

**15. SBS\_TELEGRAM\_BOT**

**Trigger:** Telegram Bot /message Updates

**Input Payload:**

{

"update\_id": 123456789,

"message": {

"message\_id": 42,

"from": {

"id": 100200300,

"first\_name": "User",

"username": "user123"

},

"chat": {

"id": 100200300,

"type": "private"

},

"date": 1730030400,

"text": "/complete 17"

}

}

**Node Flow:**

1. **Telegram Trigger →** Receive incoming Telegram message event
   * Listens for updates of type "message" from the Telegram Bot API.
2. **Command Routing →** Conditional checks on message text:
   * /complete [routine\_id] → Complete Routine
   * /skip [routine\_id] → Skip Routine
   * /advance [system\_id] → Advance System Stage
   * /status → Fetch & Display Active System Status
   * /help → Display Help Message
   * *(Fallback → Unknown command response)*
3. **Database Actions (PostgreSQL):**
   * /complete:  
     Inserts routine\_completed log entry into system\_logs
   * /skip:  
     Inserts routine\_skipped log entry into system\_logs
   * /status:  
     Runs query to summarize all active systems (progress, routines, last activity)
4. **System Actions (HTTP Request):**
   * /advance:  
     Sends POST request to  
     {{ N8N\_WEBHOOK\_BASE\_URL }}/webhook/sbs-system-update  
     with body:
   * {
   * "system\_id": "{{ system\_id }}",
   * "action": "advance\_stage"
   * }
5. **Response Flow:**
   * Each command returns a formatted Telegram message (Markdown) back to the chat:
     + ✅ *Routine Completed!*
     + ⏭️ *Routine Skipped*
     + 🚀 *System Stage Advanced!*
     + 📊 *System Status Summary*
     + 🤖 *Help Menu*
     + ❓ *Unknown Command*
6. **Respond →** Acknowledge via Telegram message (per command outcome)