**1 — Executive summary (what this document covers)**

LifeOS merges your gamified productivity app (users, characters, habits, quests, economy, AI companion) with the System for Building Systems (SBS) automation engine (systems, system\_steps, routines, system\_templates, system\_logs). The result:

* One PostgreSQL database as the single source of truth.
* n8n automations for lifecycle orchestration, routine engine, chat/notification bot, and game automations.
* Event-driven architecture using pg\_notify / LISTEN so n8n reacts to DB events in real time.
* Ownership linking: systems can belong to users, characters or guilds.
* Unified logs for analytics and auditing.

This document is split into Schema (SQL), n8n workflow specifications, environment & deployment, integration contracts (webhooks & payloads), developer notes & API contracts, test cases, and runbook.

-- 00\_core\_schema.sql

-- Postgres 13+ recommended, with pg\_notify enabled (built-in)

-- EXTENSIONS (use as needed)

CREATE EXTENSION IF NOT EXISTS "uuid-ossp"; -- if you want UUIDs later

CREATE EXTENSION IF NOT EXISTS pgcrypto; -- for secure tokens/hashing

-- USERS (Authentication)

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,

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

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- CHARACTERS (Gamified avatars)

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,

hp INTEGER DEFAULT 100,

coins INTEGER DEFAULT 100,

prestige\_level INTEGER DEFAULT 0,

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()

);

-- SKILLS

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()

);

-- HABIT\_TEMPLATES

CREATE TABLE IF NOT EXISTS habit\_templates (

id SERIAL PRIMARY KEY,

name VARCHAR(80),

skill\_name VARCHAR(64),

description TEXT

);

-- HABITS

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()

);

-- PROJECTS / QUESTS / AREAS / TASKS

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),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- ITEMS / INVENTORY / TRANSACTIONS

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()

);

-- GUILDS & MEMBERS

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)

);

-- SYSTEM\_TEMPLATES (SBS)

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()

);

-- SYSTEMS (SBS) — merged with owner\_type/owner\_id

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,

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

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

);

-- Note: Postgres doesn't allow two FKs on same column to different tables reliably; given this, you can either:

-- 1) Use owner\_id without explicit FKs and enforce ownership in app layer OR

-- 2) Use polymorphic references pattern. The SQL above includes both FKs for readability — consider selecting one strategy and implementing via application constraints/migrations.

-- SYSTEM\_STEPS

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()

);

-- ROUTINES (SBS) — can link to habits and guilds

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()

);

-- SYSTEM\_LOGS (SBS)

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()

);

-- UNIFIED\_LOGS (auditing for both game events and automation)

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'

);

-- MISC TABLES (journal, ai\_logs, rng\_events, settings)

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 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()

);

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 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()

);

-- SYSTEMS LOG TRIGGER for pg\_notify (event-driven)

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 LOGGING TRIGGER FOR HABIT / TASK / PROJECT EVENTS (example)

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;

-- Example: listen to changes on habits and tasks

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);

-- END SCHEMA

**Notes & decisions**

* For ownership (owner\_type + owner\_id), I suggest implementing polymorphic ownership in the application layer rather than relying on two FKs pointing to the same column — Postgres doesn't do polymorphic FKs out-of-the-box. Either: use owner\_type + owner\_id without FK constraints and enforce via app logic, or change to separate owner\_user\_id, owner\_character\_id, owner\_guild\_id columns with FKs and mutual exclusion rules in app code.
* All created\_at/updated\_at columns should be updated by your backend or via triggers.

# 3 — Eventing & real-time contracts

**Channels used**

* pg\_notify('system\_update', payload) — for changes to systems
* pg\_notify('unified\_event', payload) — for generic events: habits, tasks, projects, routines
* You may add pg\_notify('routine\_trigger', payload) for routine-specific triggers.

**Payload format**

* All payloads are plain JSON, the row\_to\_json(NEW)::text content. n8n must parse it.

Example system\_update payload (stringified JSON):

{

"id": 42,

"name": "Daily Review",

"category": "Habits",

"purpose": "Nightly reflection and XP awards",

"current\_stage": "define",

"owner\_type": "character",

"owner\_id": 13,

"metadata": {},

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

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

}

# n8n automation suite (blueprints & node-by-node specification)

I'll provide four canonical workflows (modularized, reusable sub-workflows) that implement SBS + Life Game automations:

1. sbs\_system\_spawner — run whenever a new system row is created (stage='define')
2. sbs\_system\_orchestrator — core orchestrator: listens to system updates (pg\_notify) and routes to subflows
3. sbs\_routine\_engine — scheduled Cron that checks/executes routines and links to habits/projects
4. lifeos\_telegram\_bot — Telegram bot / notification & interactive check-ins

For each workflow I give: trigger, nodes, node purpose, input & output shapes, error handling, environment vars, sample payloads, and tests.

Implementation note: n8n supports listening to PostgreSQL LISTEN via the “Postgres Trigger” community node or using a small bridge that listens to pg\_notify and forwards to an n8n webhook. If you run n8n in Docker, easiest reliable pattern: run a tiny Node service that LISTENs to channels and POSTs to n8n webhook endpoints. Alternatively, use Webhook + Postgres Trigger node (if available in your environment). This spec includes both options.

## 4.A — Environment variables (used across n8n workflows)

Set these in n8n environment or your docker-compose .env:

# Database

DB\_HOST=postgres

DB\_PORT=5432

DB\_USER=lifeos\_app

DB\_PASS=supersecurepassword

DB\_NAME=lifeos\_db

DB\_SSL=false

# n8n

N8N\_HOST=0.0.0.0

N8N\_PORT=5678

N8N\_BASIC\_AUTH\_ACTIVE=false

# Webhooks

N8N\_WEBHOOK\_BASE\_URL=https://<your-n8n-domain>.example.com/webhook

# Telegram

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

# App / Frontend

APP\_BASE\_URL=https://app.lifeos.example.com

# Optional: OpenAI API if used for AI companion

OPENAI\_API\_KEY=sk-xxx

# Optional: Monitoring/Logging

SENTRY\_DSN=...

## 4.B — Workflow 1: sbs\_system\_spawner (n8n filename: sbs\_system\_spawner.json)

**Purpose**  
When a new systems row is created (initial stage = define), this workflow:

* inserts 5 lifecycle steps into system\_steps
* creates default routines from templates
* creates minimal game link objects (if owner is a character: create a tutorial quest or initial project)
* writes to system\_logs and unified\_logs
* notifies the owner via Telegram/email
* advances systems.current\_stage to design

**Trigger options (choose one):**

* Option A (recommended): Triggered by a HTTP webhook that receives pg\_notify('system\_update') payload from a small pg-listener service.
* Option B: n8n Postgres Trigger node (if your n8n instance supports it) listening to systems inserts.

**Nodes (ordered)**

1. Webhook Trigger (or Postgres Trigger): receives the system\_update payload (the NEW row).
   * Input: stringified JSON (payload)
   * Output: parsed JSON object with id, current\_stage, etc.
   * Condition: only continue if current\_stage == 'define' and trigger\_type == 'INSERT' (use node expressions).
2. If node: ensure the event is a create and stage is 'define'
   * branch: true -> continue; false -> Stop.
3. Postgres - Insert Default Steps node:
   * Operation: bulk insert 5 rows into system\_steps with system\_id = payload.id and steps = define, design, build, automate, review. Status 'pending' except first maybe 'pending' or 'active' depending on design.
   * SQL (parameterized):

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

VALUES

({{ $json["id"] }}, 'define', 'pending', 'Auto-created', '{}'),

({{ $json["id"] }}, 'design', 'pending', 'Auto-created', '{}'),

({{ $json["id"] }}, 'build', 'pending', 'Auto-created', '{}'),

({{ $json["id"] }}, 'automate', 'pending', 'Auto-created', '{}'),

({{ $json["id"] }}, 'review', 'pending', 'Auto-created', '{}')

RETURNING \*;  
  
Postgres - Create Default Routines:

* Logic:
  + If system\_templates exists for payload.category, create routines from template's default\_inputs (map)
  + Otherwise create 1–3 default routines (daily/weekly) using the name derived from system name.
* Example SQL param:

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

VALUES ($1, $2, 'daily', 'Auto-generated routine', 'active', '{}'::jsonb)

* RETURNING \*;  
    
  

Input args set with expressions.

 Optional: Postgres - Create Project/Quest (if owner\_type == 'character'):

* Create a minimal project (quest) and tasks that mirror the system — this binds gamification to the automation pipeline.
* Example:

INSERT INTO projects (character\_id, title, description, total\_xp, coin\_reward, system\_template\_id)

VALUES ({{owner\_id}}, 'System: {{system\_name}}', 'Auto-created quest for system automation', 50, 20, NULL)

RETURNING \*;  
  
Postgres - Write system\_logs:

* Insert event: system created and steps spawned
* Example SQL:

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

VALUES ({{system\_id}}, 'system\_spawned', jsonb\_build\_object('created\_steps', created\_steps))

RETURNING id;

 Postgres - Write unified\_logs:

* Insert an audit row with source='n8n' and action='system\_spawned'.

 Postgres - Advance system stage:

* Update systems.current\_stage -> 'design' and update updated\_at.
* SQL:

1. UPDATE systems SET current\_stage = 'design', updated\_at = now() WHERE id = $1 RETURNING \*;  
   Telegram / Email Notification node:
   * Send message: Your system "{{name}}" was created and is ready for design. Next step: define inputs.
   * If owner is user or character, resolve owner\_id to a Telegram chat ID mapping (you must maintain user -> telegram\_chat\_id mapping in users.metadata or a dedicated table).
2. Success Logger node — final unified log with outcome='success'.

**Error handling**

* Add Error Trigger or connect each DB node to a IF node capturing error outputs that:
  + writes to unified\_logs with severity='error'
  + Send admin alert (slack/email)
  + Retry policy: n8n can be configured (execute once, or retry via schedule) — set exponential backoff for DB errors.

**Example sample payload (input)**  
Use the system\_update payload example from Section 3. Test by POSTing to the webhook with that body.

**Acceptance tests**

* Insert a systems row into Postgres; observe:
  + 5 system\_steps rows created
  + at least 1 routines row created
  + systems.current\_stage updated to 'design'
  + notifications (Telegram/email) sent
  + system\_logs and unified\_logs contain entries

## 4.C — Workflow 2: sbs\_system\_orchestrator (n8n filename: sbs\_system\_orchestrator.json)

**Purpose**  
Central orchestrator: listens for system\_steps updates or systems updates and routes to the correct step handler sub-workflow (Define, Design, Build, Automate, Review).

**Trigger**

* Primary: pg\_notify('system\_update') or webhook entrypoint receiving these notifications (recommended).
* Secondary: manual webhook or scheduled check (as fallback for missing notifications).

**High-level node flow**

1. Webhook Trigger receives system\_update payload.
2. Postgres - Fetch system\_steps — determine which step is status='pending' or status='active'.
   * SQL:
3. SELECT \* FROM system\_steps WHERE system\_id = $1 ORDER BY array\_position(array['define','design','build','automate','review'], step) LIMIT 1;  
   Switch / If node: branch by step value (define, design, ...).
4. For each branch: call a sub-workflow via Execute Workflow node (n8n supports calling workflows).
   * Subflows:
     + sbs\_step\_define
     + sbs\_step\_design
     + sbs\_step\_build
     + sbs\_step\_automate
     + sbs\_step\_review
5. After subflow returns success, Postgres - Update system\_steps set this step status='complete' and set updated\_at.
6. Write system\_logs and unified\_logs.
7. If there is a next step, change systems.current\_stage accordingly and notify owner.

**Subflows (brief)**

* sbs\_step\_define:
  + Generate and persist a "System Design Canvas" markdown (store in system\_logs as details or in external Drive if integrated).
  + Create WebUI form schema stored in systems.metadata.formschema.
  + Optional: send interactive Telegram form with a link.
  + Output: define\_complete success flag.
* sbs\_step\_design:
  + Use system\_templates if available to auto-generate default DB schema or resource stubs (write into metadata).
  + Generate a list of integration tasks (e.g., "create google drive folder", "enable calendar sync").
  + Output: design\_complete.
* sbs\_step\_build:
  + Provision resources via API nodes (e.g., call cloud APIs, create folders, create tasks in external tools).
  + Optionally create Postgres schema scaffolding (if system requires new tables) — careful: require DBA approval. If auto-sql creation is allowed, store SQL in metadata and run via Postgres Execute.
  + Output: build\_complete.
* sbs\_step\_automate:
  + Create new n8n workflows or scheduled flows dynamically (n8n can create workflows via its REST API).
  + Add routines as scheduled Cron jobs in routines table.
  + Output: automate\_complete.
* sbs\_step\_review:
  + Trigger evaluation flows: gather KPIs, create review tasks for owner, schedule next-review via routines.
  + Write system\_logs and schedule next iteration via system\_logs and routines.
  + Output: review\_complete.

**Notes**

* The orchestrator should include idempotency checks (if step already complete, skip).
* Each subflow must return consistent output shape: { success: true, logs: {...}, changed\_ids: [...] }.

**Example of decision routing**

* If system\_steps returns step='design' & status='pending':
  + run sbs\_step\_design subflow, then update system\_steps.status => complete, set systems.current\_stage => 'build'.

## 4.D — Workflow 3: sbs\_routine\_engine (n8n filename: sbs\_routine\_engine.json)

**Purpose**  
Cron-based engine to check active routines (daily/weekly), trigger their actions, and bind to gamified behavior (habits, tasks). It also handles habit => routine cross triggers.

**Trigger**

* n8n Cron node: runs every minute (small interval) or once per hour/day depending on routine granularity. For scale, run per minute but logic decides which routines to execute.

**Node flow**

1. Cron Trigger node (e.g., every 5 minutes).
2. Postgres - Fetch due routines:
   * Query example:

SELECT \* FROM routines WHERE active = true AND (trigger\_type = 'scheduled') AND (

(day\_of\_week IS NULL) OR (day\_of\_week ILIKE to\_char(now(), 'Day'))

);

* + Additional checks: metadata.next\_run if stored.

1. For each routine:
   * If habit\_id is set and the habit exists and is good:
     + Execute habit reward flow: award XP, update streak, push event to unified\_logs.
   * If routine maps to system actions:
     + Call sbs\_system\_action subflow that runs the routine's defined steps (e.g., check data sources, run calculations).
   * If routine triggers external API (calendar, Drive), call the corresponding n8n nodes.
2. Postgres - Update routine metadata with last\_run, status.
3. Postgres - Write system\_logs/unified\_logs for traceability.
4. Optional: Telegram Notification for owner summary.

**Error handling**

* Trap failures and write unified\_logs with severity='error'. For transient errors (API rate limit), set metadata.next\_retry and let orchestrator retry.

**Sample routine**

* Routine: "Morning Check"
  + day\_of\_week null (every day)
  + trigger\_type scheduled
  + bound to character\_id via owning system -> grants daily XP when executed.

## 4.E — Workflow 4: lifeos\_telegram\_bot (n8n filename: lifeos\_telegram\_bot.json)

**Purpose**  
Conversational UI: send daily check-ins, allow owners to mark routines/habits complete, advance system stages, and fetch summaries.

**Trigger**

* Telegram Trigger node — receives messages and callback queries.

**Node flow**

1. Telegram Trigger receives user message.
2. Switch node based on message type (text, callback\_query).
3. Auth/Map node: map Telegram chat\_id -> user\_id or character\_id via a user\_settings table or users.metadata.telegram\_chat\_id.
4. Supported interactions:
   * daily\_checkin (text or button) — present list of routines due via InlineKeyboard.
   * complete\_routine:<routine\_id> (callback) — call Postgres to mark progress and trigger reward flow.
   * system\_status:<system\_id> — show current stage and next actions.
   * create\_system (button or link) — open web UI to create system or send form template inside Telegram (small).
5. After user input, run Postgres nodes to update tables (habits, routines, system\_steps) and then call sbs\_system\_orchestrator if advancement is required (e.g., user marks system step complete).
6. Send message node back to user with results (XP gained, coins, next steps).

**Security**

* Validate user permissions: ensure Telegram chat\_id is linked to the acting user; do not process requests for other users.

**Example messages**

* Bot sends: Good morning, @username! You have 3 routines due: [1] Morning Page, [2] Workout, [3] Check Inbox. Tap to mark done.
* User presses: Complete Routine 2 -> n8n marks habit completed, awards XP, records event.

## 4.F — Optional additional workflows & utilities

* init\_user\_setup — create a new character, default skills, starter quests, default settings (run when user signs up).
* habit\_checkin — triggered by user or routine; reward calculation and logging.
* achievement\_unlock — triggered when XP thresholds cross; writes achievements, unified\_logs, notifies user.
* event\_seeder — monthly cron for random events.
* ai\_missions — daily creation of tasks via OpenAI.

Each of these is implemented as modular sub-workflows that can be called by orchestrator or triggered by DB events.

# 5 — n8n implementation specifics & API usage

**n8n REST API**

* Use n8n REST API to create workflows dynamically (used by sbs\_step\_automate if you want to spawn new n8n workflows programmatically).
* Secure the n8n REST API with an internal API key.

**Idempotency & race conditions**

* Always use DB transactions when making multiple writes (e.g., creating system\_steps then updating systems.current\_stage). n8n Postgres node supports multi-statement transactions or use a stored procedure.
* Use unique constraints + INSERT ... ON CONFLICT DO NOTHING for idempotency.

**Workflow naming conventions**

* sbs\_system\_spawner
* sbs\_system\_orchestrator
* sbs\_step\_define / sbs\_step\_design / sbs\_step\_build / sbs\_step\_automate / sbs\_step\_review
* sbs\_routine\_engine
* lifeos\_telegram\_bot
* lifeos\_init\_user\_setup

**Logging**

* Each node that mutates data must write to unified\_logs with source='n8n' and action describing the mutation.

# 6 — Docker Compose deployment blueprint (Postgres + n8n + pg-listener + optional Redis / Adminer)

Below is an example docker-compose.yml to get a local production-ish environment.

version: '3.8'

services:

postgres:

image: postgres:15

restart: always

environment:

POSTGRES\_USER: lifeos\_app

POSTGRES\_PASSWORD: supersecurepassword

POSTGRES\_DB: lifeos\_db

volumes:

- pgdata:/var/lib/postgresql/data

ports:

- "5432:5432"

n8n:

image: n8nio/n8n:latest

restart: always

environment:

DB\_TYPE: postgres

DB\_POSTGRESDB\_HOST: postgres

DB\_POSTGRESDB\_PORT: 5432

DB\_POSTGRESDB\_DATABASE: lifeos\_db

DB\_POSTGRESDB\_USER: lifeos\_app

DB\_POSTGRESDB\_PASSWORD: supersecurepassword

N8N\_HOST: 0.0.0.0

N8N\_PORT: 5678

N8N\_PROTOCOL: http

WEBHOOK\_URL: https://<your-n8n-domain>.example.com

EXECUTIONS\_PROCESS: main

GENERIC\_TIMEZONE: "UTC"

TELEGRAM\_BOT\_TOKEN: ${TELEGRAM\_BOT\_TOKEN}

ports:

- "5678:5678"

depends\_on:

- postgres

volumes:

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

pg-listener:

image: node:18

working\_dir: /app

volumes:

- ./pg-listener:/app

command: ["node","listener.js"]

environment:

DB\_HOST: postgres

DB\_PORT: 5432

DB\_USER: lifeos\_app

DB\_PASSWORD: supersecurepassword

DB\_NAME: lifeos\_db

N8N\_WEBHOOK\_BASE\_URL: http://n8n:5678/webhook/pg-notify

depends\_on:

- postgres

- n8n

adminer:

image: adminer

restart: always

ports:

- 8080:8080

volumes:

pgdata:

n8n\_data:  
  
**pg-listener service**

* Simple Node script (listener.js) that connects to Postgres, LISTEN system\_update, unified\_event, and on notification sends a POST to N8N\_WEBHOOK\_BASE\_URL with payload:

// skeleton of listener.js (implement robust reconnect & logging)

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 });

await client.connect();

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

const channel = msg.channel; // e.g., system\_update

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

await fetch(process.env.N8N\_WEBHOOK\_BASE\_URL, {

method: 'POST',

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

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

});

});

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

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

This is resilient and simple; prefer this pattern to complex DB plugins inside n8n for reliability.

# 7 — Webhook & API contracts

**n8n webhook endpoints** (examples)

* /webhook/pg-notify (receives { channel: 'system\_update', payload: {...} })
* /webhook/telegram (bot webhook if you prefer Telegram -> n8n via webhook)
* /webhook/ui-action (frontend calls to drive actions: mark habit, create system, etc.)

**Frontend -> Backend (API)**

* Use your usual API server (Next.js / Node) to manage auth and user flows. The DB is authoritative; workflows are triggered by DB changes.

**Payload shape conventions**

* Standardize keys: entity, id, actor { type: 'user'|'system'|'cron', id: 123 }, timestamp, metadata.

# 8 — Security, auth and permissions

* **DB credentials**: Use strong passwords, rotate via secrets manager (Vault, AWS Secrets Manager).
* **n8n**: protect with HTTP Basic Auth in production or run behind OAuth-protected gateway, set N8N\_BASIC\_AUTH\_ACTIVE=true.
* **Webhook endpoints**: include an X-Signature HMAC header using a shared secret between pg-listener and n8n to guard against spoofing.
* **Telegram bot**: ensure mapping from chat\_id to user\_id is authenticated and confirmed (users must opt-in).
* **Access controls**: workflows that mutate resources must validate owner\_type + owner\_id before performing changes.
* **Data privacy**: mark PII columns and redact logs before sending to non-authorized channels.
* **Rate-limiting**: apply rate-limits to bot endpoints.

# 9 — Monitoring, observability & backups

**Monitoring**

* Expose Prometheus metrics for n8n and pg-listener.
* Monitor queue lengths (if n8n uses Redis), DB slow queries.

**Logging**

* All n8n key nodes write to unified\_logs. Also forward n8n execution logs to centralized logging (ELK/Datadog).

**Backups**

* Postgres daily backups (base backups + WAL shipping). Keep at least 7 days of retention for dev; longer for prod.
* Export n8n workflows regularly (n8n GUI -> Export) and store in git.

# 10 — Developer notes, best practices & migration strategy

**Development workflow**

* Keep schema migrations in version control (Flyway or similar).
* Keep n8n workflows in git (export JSON).
* Store templates system\_templates as versioned JSONB (include a schema\_ref).

**Testing**

* Unit test reward calculations (XP, coins).
* Integration tests: create systems row -> assert steps & routines created; run routine\_engine -> assert habit updates.

**Operations**

* Run n8n workflows in “production mode” with proper concurrency limits and EXECUTIONS\_PROCESS=main.
* Use Execute Workflow nodes for subflows rather than copying nodes.

**Scale considerations**

* If you expect thousands of routines, implement lightweight queues or microservices to handle routine execution so n8n is not overloaded. Consider a small job-queue (Redis + Bull) for high-volume tasks, with n8n picking results.

# 11 — Sample test cases & acceptance criteria

**Test 1: System spawn**

* Action: Insert system row via SQL: INSERT INTO systems (name, category, purpose, owner\_type, owner\_id) VALUES (...).
* Expected:
  + 5 system\_steps created.
  + At least one routines created.
  + systems.current\_stage becomes design.
  + Telegram/Email notification sent to owner.
  + system\_logs & unified\_logs entries present.

**Test 2: Routine execution**

* Setup: Create routine with trigger\_type='scheduled' and habit\_id pointing to a good habit.
* Action: Run sbs\_routine\_engine Cron.
* Expected:
  + Habit streak incremented, XP awarded.
  + events or unified\_logs written.
  + Player characters.xp updated.

**Test 3: Orchestrator step advancement**

* Setup: System with system\_steps where first step is define with status pending.
* Action: send pg\_notify or call orchestrator webhook.
* Expected:
  + sbs\_step\_define runs, writes design canvas to system\_logs.
  + system\_steps step updated to complete, systems.current\_stage to design.

**Test 4: Telegram flow**

* Action: Send /checkin in Telegram.
* Expected:
  + Bot replies with due routines list.
  + Mark a routine as done via callback -> DB updated and XP awarded.

# 12 — Example SQL & sample data (for tests)

-- Create a sample user and character

INSERT INTO users (email, username) VALUES ('alice@example.com', 'alice') RETURNING id;

-- Suppose id = 10

INSERT INTO characters (user\_id, class, goals) VALUES (10, 'Starter', 'Get fit') RETURNING id;

-- Suppose character id = 20

-- Create a system owned by character

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

VALUES ('Morning Routine System','Habits','Automate morning routines','character',20) RETURNING \*;

-- Insert triggers will fire and n8n should spawn steps/routines

# 13 — Migration & backward-compatibility guidance

* If you already have separate DBs for SBS and Life App, write migration scripts to:
  + Map users table into unified users table
  + Map systems and add owner\_type/owner\_id
  + Consolidate system\_logs into unified\_logs (copy & tag source='sbs' or source='life')
* Use transactional migration for each table and run migration in off-hours with feature-toggle control.

# 14 — Operational runbook & troubleshooting

**When notifications aren’t reaching n8n**

* Validate pg-listener service running and connected to DB.
* Check LISTEN subscriptions and pg\_notify triggers.
* Verify N8N\_WEBHOOK\_BASE\_URL reachable from pg-listener.

**When workflows fail**

* Inspect n8n execution logs.
* Look into unified\_logs for severity='error'.
* Validate DB connection settings, and check user permissions.

**When DB migrations fail**

* Rollback and inspect the migration script; ensure no locking issues.

# 15 — Deliverables for your team (what to check off / handover items)

* SQL migration scripts in git (one file per migration).
* n8n workflow JSON files exported:
  + sbs\_system\_spawner.json
  + sbs\_system\_orchestrator.json
  + sbs\_step\_define.json
  + sbs\_step\_design.json
  + sbs\_step\_build.json
  + sbs\_step\_automate.json
  + sbs\_step\_review.json
  + sbs\_routine\_engine.json
  + lifeos\_telegram\_bot.json
  + lifeos\_init\_user\_setup.json
* pg-listener Node service (listener.js) with env configuration and health endpoint.
* Docker Compose file for local dev.
* Postman collection / API doc for webhooks & example payloads.
* Acceptance tests & scripts for CI (create test DB, run migrations, run pg-listener, import n8n flows, run tests).

# 16 — Quick do’s & don’ts (summary)

Do:

* Use event-driven pg\_notify for real-time reaction.
* Keep n8n workflows modular and idempotent.
* Use unified\_logs for auditability and postmortems.
* Map ownership (user/character/guild) strictly on the application side.

Don't:

* Auto-create large DB tables arbitrarily inside n8n without code review.
* Expose n8n admin console without auth.
* Store long-term PII in public logs.

# 17 — Appendix: sample n8n node detail (copyable blueprint)

Below is a concise blueprint for a single n8n node chain (spawner main nodes) — use this when building the exported n8n JSON.

* Webhook Trigger:
  + HTTP Method: POST
  + Path: /webhook/pg-notify
  + Response Code: 200
  + Output: {{ $json["payload"] }}
* Set (extract payload):
  + Fields:
    - system\_id = {{$json["payload"]["id"]}}
    - system\_name = {{$json["payload"]["name"]}}
    - current\_stage = {{$json["payload"]["current\_stage"]}}
    - owner\_type = {{$json["payload"]["owner\_type"]}}
    - owner\_id = {{$json["payload"]["owner\_id"]}}
* IF:
  + Condition: {{$json["current\_stage"]}} === 'define' && {{ $json["event"] === 'INSERT' || true }}
* Postgres (insert steps):
  + Credentials: DB credentials env
  + Query:

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

VALUES

($1, 'define','pending','auto','{}'::jsonb),

($1, 'design','pending','auto','{}'::jsonb),

($1, 'build','pending','auto','{}'::jsonb),

($1, 'automate','pending','auto','{}'::jsonb),

($1, 'review','pending','auto','{}'::jsonb) RETURNING \*;

* 

Parameters: $1 = {{$json["system\_id"]}}

 Postgres (create routine):

* Query:

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

VALUES ($1,$2,null,$3,'active','{}'::jsonb) RETURNING \*;

Postgres (log):

* Query:

INSERT INTO system\_logs (system\_id, event, details) VALUES ($1, 'system\_spawned', $2::jsonb);

* HTTP Request (Telegram):
  + Endpoint: https://api.telegram.org/bot${{env.TELEGRAM\_BOT\_TOKEN}}/sendMessage
  + Body: { "chat\_id": "{{owner\_telegram\_id}}", "text": "System created: {{system\_name}}. Next stage: design." }
  + Use application/json.

# 18 — Final checklist + next steps for the dev team

1. Provision Postgres, run migrations (schema above).
2. Deploy pg-listener service and test pg\_notify events.
3. Deploy n8n and import the JSON workflows (exported from dev).
4. Register Telegram bot token & set webhook to n8n if using direct webhook.
5. Run acceptance tests (section 11).
6. Integrate with frontend: implement UI for creating systems and linking them to characters/guilds.
7. Implement RBAC checks and security gateways.
8. Schedule recurring backups and monitoring.

## N8N JSONs





{

"name": "ACHIEVEMENT\_UNLOCK",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "check-achievements",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook\_achievement\_check",

"name": "Webhook - Check Achievements",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1.1,

"position": [250, 300],

"webhookId": "achievement-check"

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT c.\*, COUNT(DISTINCT h.id) as habit\_count, COUNT(DISTINCT p.id) as project\_count, SUM(CASE WHEN h.streak >= 7 THEN 1 ELSE 0 END) as week\_streaks FROM characters c LEFT JOIN habits h ON c.id = h.character\_id LEFT JOIN projects p ON c.id = p.character\_id WHERE c.id = $1 GROUP BY c.id",

"options": {}

},

"id": "fetch\_character\_stats",

"name": "Fetch Character Stats",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT MAX(s.level) as max\_skill\_level, COUNT(CASE WHEN s.level >= 5 THEN 1 END) as level5\_skills, COUNT(CASE WHEN s.level >= 10 THEN 1 END) as level10\_skills FROM skills s WHERE s.character\_id = $1",

"options": {}

},

"id": "fetch\_skill\_stats",

"name": "Fetch Skill Stats",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT MAX(h.streak) as max\_streak, COUNT(CASE WHEN h.streak >= 30 THEN 1 END) as month\_streaks, COUNT(CASE WHEN h.streak >= 90 THEN 1 END) as quarter\_streaks FROM habits h WHERE h.character\_id = $1 AND h.type = 'good'",

"options": {}

},

"id": "fetch\_habit\_stats",

"name": "Fetch Habit Stats",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [850, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT SUM(coins\_change) as total\_earned FROM events WHERE character\_id = $1 AND coins\_change > 0",

"options": {}

},

"id": "fetch\_wealth\_stats",

"name": "Fetch Wealth Stats",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT title FROM achievements WHERE character\_id = $1",

"options": {}

},

"id": "fetch\_existing\_achievements",

"name": "Fetch Existing Achievements",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "// Define all achievement criteria\nconst characterStats = $('Fetch Character Stats').first().json;\nconst skillStats = $('Fetch Skill Stats').first().json;\nconst habitStats = $('Fetch Habit Stats').first().json;\nconst wealthStats = $('Fetch Wealth Stats').first().json;\nconst existingAchievements = $('Fetch Existing Achievements').all().map(a => a.json.title);\n\n// Achievement definitions\nconst achievementDefinitions = [\n // Level-based achievements\n { title: 'Novice Adventurer', condition: characterStats.level >= 5, reward\_type: 'xp', bonus\_value: 100, description: 'Reached Level 5' },\n { title: 'Skilled Warrior', condition: characterStats.level >= 10, reward\_type: 'xp', bonus\_value: 250, description: 'Reached Level 10' },\n { title: 'Master of Growth', condition: characterStats.level >= 25, reward\_type: 'xp', bonus\_value: 1000, description: 'Reached Level 25' },\n { title: 'Legendary Hero', condition: characterStats.level >= 50, reward\_type: 'coins', bonus\_value: 5000, description: 'Reached Level 50' },\n \n // XP Milestones\n { title: 'First Thousand', condition: characterStats.xp >= 1000, reward\_type: 'coins', bonus\_value: 100, description: 'Earned 1,000 XP' },\n { title: 'Ten Thousand Strong', condition: characterStats.xp >= 10000, reward\_type: 'coins', bonus\_value: 500, description: 'Earned 10,000 XP' },\n { title: 'XP Legend', condition: characterStats.xp >= 50000, reward\_type: 'xp', bonus\_value: 5000, description: 'Earned 50,000 XP' },\n \n // Streak achievements\n { title: 'Week Warrior', condition: habitStats.max\_streak >= 7, reward\_type: 'xp', bonus\_value: 75, description: 'Maintained a 7-day streak' },\n { title: 'Month Master', condition: habitStats.max\_streak >= 30, reward\_type: 'xp', bonus\_value: 300, description: 'Maintained a 30-day streak' },\n { title: 'Quarter Champion', condition: habitStats.max\_streak >= 90, reward\_type: 'coins', bonus\_value: 1000, description: 'Maintained a 90-day streak' },\n { title: 'Year of Discipline', condition: habitStats.max\_streak >= 365, reward\_type: 'coins', bonus\_value: 10000, description: 'Maintained a 365-day streak' },\n \n // Habit achievements\n { title: 'Habit Builder', condition: characterStats.habit\_count >= 5, reward\_type: 'xp', bonus\_value: 50, description: 'Created 5 habits' },\n { title: 'Routine Master', condition: characterStats.habit\_count >= 10, reward\_type: 'coins', bonus\_value: 200, description: 'Created 10 habits' },\n { title: 'Streak Collector', condition: habitStats.month\_streaks >= 3, reward\_type: 'xp', bonus\_value: 500, description: 'Have 3 habits with 30+ day streaks' },\n \n // Skill achievements\n { title: 'Skill Apprentice', condition: skillStats.max\_skill\_level >= 5, reward\_type: 'xp', bonus\_value: 100, description: 'Reached level 5 in any skill' },\n { title: 'Skill Expert', condition: skillStats.max\_skill\_level >= 10, reward\_type: 'xp', bonus\_value: 500, description: 'Reached level 10 in any skill' },\n { title: 'Renaissance Soul', condition: skillStats.level5\_skills >= 3, reward\_type: 'coins', bonus\_value: 300, description: 'Have 3 skills at level 5+' },\n { title: 'Jack of All Trades', condition: skillStats.level5\_skills >= 5, reward\_type: 'coins', bonus\_value: 750, description: 'Have 5 skills at level 5+' },\n { title: 'Master of Many', condition: skillStats.level10\_skills >= 3, reward\_type: 'xp', bonus\_value: 2000, description: 'Have 3 skills at level 10+' },\n \n // Quest achievements\n { title: 'Quest Starter', condition: characterStats.project\_count >= 1, reward\_type: 'xp', bonus\_value: 25, description: 'Started your first quest' },\n { title: 'Quest Hunter', condition: characterStats.project\_count >= 10, reward\_type: 'coins', bonus\_value: 250, description: 'Started 10 quests' },\n { title: 'Epic Questor', condition: characterStats.project\_count >= 25, reward\_type: 'coins', bonus\_value: 1000, description: 'Started 25 quests' },\n \n // Wealth achievements\n { title: 'First Fortune', condition: wealthStats.total\_earned >= 1000, reward\_type: 'coins', bonus\_value: 100, description: 'Earned 1,000 coins total' },\n { title: 'Wealthy Adventurer', condition: wealthStats.total\_earned >= 10000, reward\_type: 'coins', bonus\_value: 500, description: 'Earned 10,000 coins total' },\n { title: 'Coin Hoarder', condition: characterStats.coins >= 5000, reward\_type: 'xp', bonus\_value: 500, description: 'Hold 5,000 coins at once' },\n \n // Special achievements\n { title: 'Survivor', condition: characterStats.hp > 0 && characterStats.level >= 10, reward\_type: 'xp', bonus\_value: 200, description: 'Reached level 10 without being defeated' },\n { title: 'Balanced Life', condition: skillStats.level5\_skills >= 5 && habitStats.week\_streaks >= 5, reward\_type: 'coins', bonus\_value: 1000, description: 'Have 5 skills at level 5+ and 5 weekly streaks' }\n];\n\n// Check which achievements should be unlocked\nconst newAchievements = achievementDefinitions.filter(achievement => {\n return achievement.condition && !existingAchievements.includes(achievement.title);\n});\n\nif (newAchievements.length === 0) {\n return [{ json: { hasNewAchievements: false, message: 'No new achievements unlocked' } }];\n}\n\nreturn newAchievements.map(achievement => ({\n json: {\n characterId: characterStats.id,\n hasNewAchievements: true,\n ...achievement\n }\n}));"

},

"id": "check\_achievement\_criteria",

"name": "Check Achievement Criteria",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [1450, 300]

},

{

"parameters": {

"conditions": {

"boolean": [

{

"value1": "={{ $json.hasNewAchievements }}",

"value2": true

}

]

}

},

"id": "has\_new\_achievements",

"name": "Has New Achievements?",

"type": "n8n-nodes-base.if",

"typeVersion": 2,

"position": [1650, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO achievements (character\_id, title, description, reward\_type, bonus\_value) VALUES ($1, $2, $3, $4, $5) RETURNING \*",

"options": {}

},

"id": "insert\_achievement",

"name": "Insert Achievement",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1850, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE characters SET xp = xp + CASE WHEN $2 = 'xp' THEN $3 ELSE 0 END, coins = coins + CASE WHEN $2 = 'coins' THEN $3 ELSE 0 END WHERE id = $1 RETURNING \*",

"options": {}

},

"id": "apply\_achievement\_reward",

"name": "Apply Achievement Reward",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2050, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, xp\_change, coins\_change, description) VALUES ($1, 'achievement\_unlocked', CASE WHEN $2 = 'xp' THEN $3 ELSE 0 END, CASE WHEN $2 = 'coins' THEN $3 ELSE 0 END, $4) RETURNING \*",

"options": {}

},

"id": "log\_achievement\_event",

"name": "Log Achievement Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2250, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('system', 0, 'character', $1, 'achievement\_unlocked', $2, 'success', 'info', 'n8n') RETURNING \*",

"options": {}

},

"id": "log\_system\_achievement",

"name": "Log System Achievement",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2450, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"aggregate": "aggregateAllItemData",

"destinationFieldName": "achievements",

"include": "selectedFields",

"fieldsToInclude": {

"fields": [

{

"fieldName": "title"

},

{

"fieldName": "description"

},

{

"fieldName": "reward\_type"

},

{

"fieldName": "bonus\_value"

}

]

},

"options": {}

},

"id": "aggregate\_achievements",

"name": "Aggregate Achievements",

"type": "n8n-nodes-base.aggregate",

"typeVersion": 1,

"position": [2650, 200]

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": true,\n \"newAchievements\": $json.achievements || [],\n \"count\": ($json.achievements || []).length,\n \"message\": \"Congratulations! You've unlocked new achievements!\"\n} }}",

"options": {}

},

"id": "respond\_achievements",

"name": "Respond Achievements",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [2850, 300]

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\"success\": true, \"newAchievements\": [], \"count\": 0, \"message\": \"No new achievements at this time\"} }}",

"options": {}

},

"id": "respond\_no\_achievements",

"name": "Respond No Achievements",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [1850, 400]

}

],

"pinData": {},

"connections": {

"Webhook - Check Achievements": {

"main": [

[

{

"node": "Fetch Character Stats",

"type": "main",

"index": 0

}

]

]

},

"Fetch Character Stats": {

"main": [

[

{

"node": "Fetch Skill Stats",

"type": "main",

"index": 0

}

]

]

},

"Fetch Skill Stats": {

"main": [

[

{

"node": "Fetch Habit Stats",

"type": "main",

"index": 0

}

]

]

},

"Fetch Habit Stats": {

"main": [

[

{

"node": "Fetch Wealth Stats",

"type": "main",

"index": 0

}

]

]

},

"Fetch Wealth Stats": {

"main": [

[

{

"node": "Fetch Existing Achievements",

"type": "main",

"index": 0

}

]

]

},

"Fetch Existing Achievements": {

"main": [

[

{

"node": "Check Achievement Criteria",

"type": "main",

"index": 0

}

]

]

},

"Check Achievement Criteria": {

"main": [

[

{

"node": "Has New Achievements?",

"type": "main",

"index": 0

}

]

]

},

"Has New Achievements?": {

"main": [

[

{

"node": "Insert Achievement",

"type": "main",

"index": 0

}

],

[

{

"node": "Respond No Achievements",

"type": "main",

"index": 0

}

]

]

},

"Insert Achievement": {

"main": [

[

{

"node": "Apply Achievement Reward",

"type": "main",

"index": 0

}

]

]

},

"Apply Achievement Reward": {

"main": [

[

{

"node": "Log Achievement Event",

"type": "main",

"index": 0

}

]

]

},

"Log Achievement Event": {

"main": [

[

{

"node": "Log System Achievement",

"type": "main",

"index": 0

}

]

]

},

"Log System Achievement": {

"main": [

[

{

"node": "Aggregate Achievements",

"type": "main",

"index": 0

}

]

]

},

"Aggregate Achievements": {

"main": [

[

{

"node": "Respond Achievements",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"executionOrder": "v1"

},

"versionId": "1",

"meta": {

"instanceId": "life-game-production"

},

"id": "6",

"tags": []

}

{

"name": "AI\_MISSIONS",

"nodes": [

{

"parameters": {

"rule": {

"interval": [

{

"field": "cronExpression",

"expression": "0 6 \* \* \*"

}

]

}

},

"id": "schedule\_daily",

"name": "Schedule - Daily 6 AM",

"type": "n8n-nodes-base.scheduleTrigger",

"typeVersion": 1.2,

"position": [250, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT c.id, c.user\_id, c.level, c.xp, c.hp, c.coins, c.goals, c.class, u.username FROM characters c JOIN users u ON c.user\_id = u.id WHERE c.last\_login > NOW() - INTERVAL '7 days'",

"options": {}

},

"id": "fetch\_active\_users",

"name": "Fetch Active Users",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT s.name, s.level, s.xp FROM skills s WHERE s.character\_id = $1 ORDER BY s.xp DESC LIMIT 3",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Fetch Active Users').item.json.id }}"

}

]

}

}

},

"id": "fetch\_user\_skills",

"name": "Fetch User Skills",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT event\_type, COUNT(\*) as count, SUM(xp\_change) as total\_xp FROM events WHERE character\_id = $1 AND event\_date > NOW() - INTERVAL '7 days' GROUP BY event\_type",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Fetch Active Users').item.json.id }}"

}

]

}

}

},

"id": "fetch\_recent\_activity",

"name": "Fetch Recent Activity",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 450],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT h.name, h.streak FROM habits h WHERE h.character\_id = $1 AND h.type = 'good' ORDER BY h.streak DESC LIMIT 5",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Fetch Active Users').item.json.id }}"

}

]

}

}

},

"id": "fetch\_habit\_streaks",

"name": "Fetch Habit Streaks",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 600],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT p.title, COUNT(t.id) as total\_tasks, SUM(CASE WHEN t.completed = true THEN 1 ELSE 0 END) as completed\_tasks FROM projects p LEFT JOIN tasks t ON t.project\_id = p.id WHERE p.character\_id = $1 AND p.completed = false GROUP BY p.id, p.title ORDER BY p.deadline ASC LIMIT 3",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Fetch Active Users').item.json.id }}"

}

]

}

}

},

"id": "fetch\_active\_projects",

"name": "Fetch Active Projects",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 750],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"assignments": {

"assignments": [

{

"id": "user\_context",

"name": "userContext",

"value": "={{ {\n characterId: $('Fetch Active Users').item.json.id,\n username: $('Fetch Active Users').item.json.username,\n level: $('Fetch Active Users').item.json.level,\n hp: $('Fetch Active Users').item.json.hp,\n coins: $('Fetch Active Users').item.json.coins,\n goals: $('Fetch Active Users').item.json.goals,\n class: $('Fetch Active Users').item.json.class,\n topSkills: $('Fetch User Skills').all().map(s => s.json),\n recentActivity: $('Fetch Recent Activity').all().map(a => a.json),\n habitStreaks: $('Fetch Habit Streaks').all().map(h => h.json),\n activeProjects: $('Fetch Active Projects').all().map(p => p.json)\n} }}",

"type": "object"

}

]

},

"options": {}

},

"id": "prepare\_ai\_context",

"name": "Prepare AI Context",

"type": "n8n-nodes-base.set",

"typeVersion": 3.3,

"position": [850, 475]

},

{

"parameters": {

"authentication": "predefinedCredentialType",

"nodeCredentialType": "openAiApi",

"resource": "text",

"model": {

"\_\_rl": true,

"value": "gpt-4o-mini",

"mode": "list",

"cachedResultName": "gpt-4o-mini"

},

"messages": {

"values": [

{

"content": "=You are a wise AI companion for {{ $json.userContext.username }} in their personal growth journey.\n\nPlayer Profile:\n- Class: {{ $json.userContext.class }}\n- Level: {{ $json.userContext.level }}\n- HP: {{ $json.userContext.hp }}/100\n- Coins: {{ $json.userContext.coins }}\n- Goals: {{ $json.userContext.goals }}\n\nTop Skills:\n{{ $json.userContext.topSkills.map(s => `- ${s.name}: Level ${s.level} (${s.xp} XP)`).join('\\n') }}\n\nRecent Activity (Last 7 Days):\n{{ $json.userContext.recentActivity.map(a => `- ${a.event\_type}: ${a.count} times (${a.total\_xp || 0} XP earned)`).join('\\n') }}\n\nTop Habit Streaks:\n{{ $json.userContext.habitStreaks.map(h => `- ${h.name}: ${h.streak} days`).join('\\n') }}\n\nActive Projects:\n{{ $json.userContext.activeProjects.map(p => `- ${p.title}: ${p.completed\_tasks}/${p.total\_tasks} tasks done`).join('\\n') }}\n\nGenerate 3 personalized daily missions for today. Each mission should:\n1. Be specific and actionable\n2. Align with their goals and current progress\n3. Challenge them appropriately for their level\n4. Include XP reward (10-50 based on difficulty)\n5. Include coin reward (5-25 based on difficulty)\n6. Relate to one of their top skills\n\nRespond ONLY with a JSON array in this exact format:\n[\n {\"title\": \"Mission title\", \"description\": \"Detailed description\", \"xp\": 25, \"coins\": 10, \"difficulty\": \"medium\", \"skill\_name\": \"Health\"},\n {\"title\": \"Mission title 2\", \"description\": \"Detailed description 2\", \"xp\": 15, \"coins\": 8, \"difficulty\": \"easy\", \"skill\_name\": \"Work\"}\n]"

}

]

},

"options": {

"temperature": 0.8,

"maxTokens": 1000

}

},

"id": "generate\_missions\_ai",

"name": "Generate Missions (AI)",

"type": "@n8n/n8n-nodes-langchain.lmChatOpenAi",

"typeVersion": 1,

"position": [1050, 475],

"credentials": {

"openAiApi": {

"id": "2",

"name": "OpenAI account"

}

}

},

{

"parameters": {

"jsCode": "// Parse AI response and validate missions\nconst response = $input.item.json.response || $input.item.json.text || $input.item.json.content;\nlet missions = [];\n\ntry {\n // Try to parse as JSON\n const parsed = JSON.parse(response);\n missions = Array.isArray(parsed) ? parsed : [parsed];\n} catch (e) {\n // Try to extract JSON from markdown code blocks\n const jsonMatch = response.match(/```json\\s\*([\\s\\S]\*?)```/);\n if (jsonMatch) {\n missions = JSON.parse(jsonMatch[1]);\n } else {\n // Try to find JSON array directly\n const arrayMatch = response.match(/\\[\\s\*\\{[\\s\\S]\*\\}\\s\*\\]/);\n if (arrayMatch) {\n missions = JSON.parse(arrayMatch[0]);\n }\n }\n}\n\n// Validate and enhance missions\nconst characterId = $('Prepare AI Context').item.json.userContext.characterId;\nconst validatedMissions = missions.slice(0, 3).map((mission, index) => ({\n characterId: characterId,\n title: mission.title || `Daily Mission ${index + 1}`,\n description: mission.description || 'Complete this mission to earn rewards',\n xp: Math.min(Math.max(mission.xp || 20, 10), 50),\n coins: Math.min(Math.max(mission.coins || 10, 5), 25),\n difficulty: mission.difficulty || 'medium',\n skillName: mission.skill\_name || mission.skillName || 'General'\n}));\n\nreturn validatedMissions.map(m => ({ json: m }));"

},

"id": "parse\_ai\_response",

"name": "Parse AI Response",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [1250, 475]

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT a.id FROM areas a WHERE a.character\_id = $1 AND a.name = 'Daily Missions' LIMIT 1",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.characterId }}"

}

]

}

}

},

"id": "find\_missions\_area",

"name": "Find Missions Area",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1450, 475],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO areas (character\_id, name, description) VALUES ($1, 'Daily Missions', 'AI-generated daily missions') ON CONFLICT DO NOTHING RETURNING id",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Parse AI Response').item.json.characterId }}"

}

]

}

}

},

"id": "create\_missions\_area",

"name": "Create Missions Area",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1650, 575],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"conditions": {

"options": {

"caseSensitive": true,

"leftValue": "",

"typeValidation": "loose"

},

"conditions": [

{

"id": "area\_exists",

"leftValue": "={{ $json.id }}",

"rightValue": "",

"operator": {

"type": "string",

"operation": "notEmpty"

}

}

],

"combinator": "and"

},

"options": {}

},

"id": "if\_area\_exists",

"name": "Area Exists?",

"type": "n8n-nodes-base.if",

"typeVersion": 2,

"position": [1650, 475]

},

{

"parameters": {

"assignments": {

"assignments": [

{

"id": "area\_id",

"name": "areaId",

"value": "={{ $json.id }}",

"type": "number"

},

{

"id": "mission\_data",

"name": "missionData",

"value": "={{ $('Parse AI Response').item.json }}",

"type": "object"

}

]

}

},

"id": "merge\_area\_id",

"name": "Merge Area ID",

"type": "n8n-nodes-base.set",

"typeVersion": 3.3,

"position": [1850, 525]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO projects (character\_id, area\_id, title, description, total\_xp, coin\_reward, difficulty, deadline) VALUES ($1, $2, $3, $4, $5, $6, $7, CURRENT\_DATE + INTERVAL '1 day') RETURNING id",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.missionData.characterId }}"

},

{

"parameter": "={{ $json.areaId }}"

},

{

"parameter": "={{ $json.missionData.title }}"

},

{

"parameter": "={{ $json.missionData.description }}"

},

{

"parameter": "={{ $json.missionData.xp }}"

},

{

"parameter": "={{ $json.missionData.coins }}"

},

{

"parameter": "={{ $json.missionData.difficulty }}"

}

]

}

}

},

"id": "create\_mission\_project",

"name": "Create Mission Project",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2050, 525],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO tasks (project\_id, title, xp, coins, difficulty, completed) VALUES ($1, $2, $3, $4, $5, false) RETURNING id",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Create Mission Project').item.json.id }}"

},

{

"parameter": "={{ $('Merge Area ID').item.json.missionData.title }}"

},

{

"parameter": "={{ $('Merge Area ID').item.json.missionData.xp }}"

},

{

"parameter": "={{ $('Merge Area ID').item.json.missionData.coins }}"

},

{

"parameter": "={{ $('Merge Area ID').item.json.missionData.difficulty }}"

}

]

}

}

},

"id": "create\_mission\_task",

"name": "Create Mission Task",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2250, 525],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO ai\_logs (character\_id, message, insight\_type) VALUES ($1, $2, 'daily\_mission')",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Merge Area ID').item.json.missionData.characterId }}"

},

{

"parameter": "=Generated mission: {{ $('Merge Area ID').item.json.missionData.title }}"

}

]

}

}

},

"id": "log\_ai",

"name": "Log AI Generation",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2450, 525],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, description) VALUES ($1, 'ai\_mission\_created', $2)",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Merge Area ID').item.json.missionData.characterId }}"

},

{

"parameter": "=New AI mission: {{ $('Merge Area ID').item.json.missionData.title }} - {{ $('Merge Area ID').item.json.missionData.description }}"

}

]

}

}

},

"id": "log\_event",

"name": "Log Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2650, 525],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('system', 0, 'ai\_mission', $1, 'mission\_generated', $2, 'success', 'info', 'n8n')",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Create Mission Project').item.json.id }}"

},

{

"parameter": "={{ JSON.stringify({ characterId: $('Merge Area ID').item.json.missionData.characterId, missionTitle: $('Merge Area ID').item.json.missionData.title, xp: $('Merge Area ID').item.json.missionData.xp, coins: $('Merge Area ID').item.json.missionData.coins, skillName: $('Merge Area ID').item.json.missionData.skillName }) }}"

}

]

}

}

},

"id": "log\_system",

"name": "Log System",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2850, 525],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

}

],

"connections": {

"Schedule - Daily 6 AM": {

"main": [

[

{

"node": "Fetch Active Users",

"type": "main",

"index": 0

}

]

]

},

"Fetch Active Users": {

"main": [

[

{

"node": "Fetch User Skills",

"type": "main",

"index": 0

},

{

"node": "Fetch Recent Activity",

"type": "main",

"index": 0

},

{

"node": "Fetch Habit Streaks",

"type": "main",

"index": 0

},

{

"node": "Fetch Active Projects",

"type": "main",

"index": 0

}

]

]

},

"Fetch User Skills": {

"main": [

[

{

"node": "Prepare AI Context",

"type": "main",

"index": 0

}

]

]

},

"Fetch Recent Activity": {

"main": [

[

{

"node": "Prepare AI Context",

"type": "main",

"index": 0

}

]

]

},

"Fetch Habit Streaks": {

"main": [

[

{

"node": "Prepare AI Context",

"type": "main",

"index": 0

}

]

]

},

"Fetch Active Projects": {

"main": [

[

{

"node": "Prepare AI Context",

"type": "main",

"index": 0

}

]

]

},

"Prepare AI Context": {

"main": [

[

{

"node": "Generate Missions (AI)",

"type": "main",

"index": 0

}

]

]

},

"Generate Missions (AI)": {

"main": [

[

{

"node": "Parse AI Response",

"type": "main",

"index": 0

}

]

]

},

"Parse AI Response": {

"main": [

[

{

"node": "Find Missions Area",

"type": "main",

"index": 0

}

]

]

},

"Find Missions Area": {

"main": [

[

{

"node": "Area Exists?",

"type": "main",

"index": 0

}

]

]

},

"Create Missions Area": {

"main": [

[

{

"node": "Merge Area ID",

"type": "main",

"index": 0

}

]

]

},

"Area Exists?": {

"main": [

[

{

"node": "Merge Area ID",

"type": "main",

"index": 0

}

],

[

{

"node": "Create Missions Area",

"type": "main",

"index": 0

}

]

]

},

"Merge Area ID": {

"main": [

[

{

"node": "Create Mission Project",

"type": "main",

"index": 0

}

]

]

},

"Create Mission Project": {

"main": [

[

{

"node": "Create Mission Task",

"type": "main",

"index": 0

}

]

]

},

"Create Mission Task": {

"main": [

[

{

"node": "Log AI Generation",

"type": "main",

"index": 0

}

]

]

},

"Log AI Generation": {

"main": [

[

{

"node": "Log Event",

"type": "main",

"index": 0

}

]

]

},

"Log Event": {

"main": [

[

{

"node": "Log System",

"type": "main",

"index": 0

}

]

]

}

},

"settings": {

"executionOrder": "v1"

}

}

{

"name": "CRON\_MANAGER",

"nodes": [

{

"parameters": {

"rule": {

"interval": [

{

"field": "cronExpression",

"expression": "0 0 \* \* \*"

}

]

}

},

"id": "schedule\_daily",

"name": "Schedule - Daily Midnight",

"type": "n8n-nodes-base.scheduleTrigger",

"typeVersion": 1.2,

"position": [250, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT c.id, c.user\_id, c.hp, c.coins, c.level, u.username, u.email, s.overdraft\_rule FROM characters c JOIN users u ON c.user\_id = u.id LEFT JOIN settings s ON s.user\_id = u.id WHERE c.last\_login > NOW() - INTERVAL '30 days'",

"options": {}

},

"id": "fetch\_active\_characters",

"name": "Fetch Active Characters",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('system', 0, 'cron', 0, 'daily\_maintenance', $1, 'success', 'info', 'n8n')",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ JSON.stringify({ processedCharacters: $('Calculate Daily Penalties').all().length, eventsGenerated: $('Generate Daily Events').all().length, streaksBroken: $('Reset Broken Streaks').all().length, timestamp: new Date().toISOString() }) }}"

}

]

}

}

},

"id": "log\_system",

"name": "Log System Run",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

}

],

"connections": {

"Schedule - Daily Midnight": {

"main": [

[

{

"node": "Fetch Active Characters",

"type": "main",

"index": 0

},

{

"node": "Check Overdue Habits",

"type": "main",

"index": 0

},

{

"node": "Check Overdue Tasks",

"type": "main",

"index": 0

},

{

"node": "Fetch Available Events",

"type": "main",

"index": 0

},

{

"node": "Reset Broken Streaks",

"type": "main",

"index": 0

}

]

]

},

"Fetch Active Characters": {

"main": [

[

{

"node": "Calculate Daily Penalties",

"type": "main",

"index": 0

}

]

]

},

"Check Overdue Habits": {

"main": [

[

{

"node": "Calculate Daily Penalties",

"type": "main",

"index": 0

}

]

]

},

"Check Overdue Tasks": {

"main": [

[

{

"node": "Calculate Daily Penalties",

"type": "main",

"index": 0

}

]

]

},

"Calculate Daily Penalties": {

"main": [

[

{

"node": "Update Character HP",

"type": "main",

"index": 0

}

]

]

},

"Update Character HP": {

"main": [

[

{

"node": "Log Event",

"type": "main",

"index": 0

}

]

]

},

"Log Event": {

"main": [

[

{

"node": "Log System Run",

"type": "main",

"index": 0

}

]

]

},

"Fetch Available Events": {

"main": [

[

{

"node": "Generate Daily Events",

"type": "main",

"index": 0

}

]

]

},

"Generate Daily Events": {

"main": [

[

{

"node": "Insert Daily Events",

"type": "main",

"index": 0

}

]

]

},

"Reset Broken Streaks": {

"main": [

[

{

"node": "Log Streak Breaks",

"type": "main",

"index": 0

}

]

]

}

},

"settings": {

"executionOrder": "v1"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT h.character\_id, COUNT(\*) as overdue\_habits FROM habits h WHERE h.type = 'good' AND h.last\_completed < CURRENT\_DATE - INTERVAL '2 days' GROUP BY h.character\_id",

"options": {}

},

"id": "check\_overdue\_habits",

"name": "Check Overdue Habits",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 450],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT t.character\_id, COUNT(\*) as overdue\_tasks FROM (SELECT c.id as character\_id FROM characters c JOIN projects p ON p.character\_id = c.id JOIN tasks t ON t.project\_id = p.id WHERE t.completed = false AND t.deadline < CURRENT\_DATE GROUP BY c.id) as t GROUP BY t.character\_id",

"options": {}

},

"id": "check\_overdue\_tasks",

"name": "Check Overdue Tasks",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 600],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "// Calculate daily HP penalties based on character activity\nconst characters = $('Fetch Active Characters').all().map(item => item.json);\nconst overdueHabits = $('Check Overdue Habits').all().map(item => item.json);\nconst overdueTasks = $('Check Overdue Tasks').all().map(item => item.json);\n\nconst penalties = characters.map(char => {\n // Find overdue data for this character\n const habitData = overdueHabits.find(h => h.character\_id === char.id) || { overdue\_habits: 0 };\n const taskData = overdueTasks.find(t => t.character\_id === char.id) || { overdue\_tasks: 0 };\n \n // Calculate penalties\n let hpPenalty = 0;\n let reason = [];\n \n // Penalty for overdue habits (2 HP per overdue habit, max 20)\n if (habitData.overdue\_habits > 0) {\n const habitPenalty = Math.min(habitData.overdue\_habits \* 2, 20);\n hpPenalty += habitPenalty;\n reason.push(`${habitData.overdue\_habits} overdue habits (-${habitPenalty} HP)`);\n }\n \n // Penalty for overdue tasks (5 HP per overdue task, max 25)\n if (taskData.overdue\_tasks > 0) {\n const taskPenalty = Math.min(taskData.overdue\_tasks \* 5, 25);\n hpPenalty += taskPenalty;\n reason.push(`${taskData.overdue\_tasks} overdue tasks (-${taskPenalty} HP)`);\n }\n \n // Penalty for negative coins (overdraft penalty)\n if (char.coins < 0) {\n const overdraftPenalty = Math.min(Math.abs(char.coins) \* 0.1, 15);\n hpPenalty += Math.floor(overdraftPenalty);\n reason.push(`Overdraft penalty (-${Math.floor(overdraftPenalty)} HP)`);\n }\n \n // Daily maintenance bonus for active players (level 10+)\n let hpBonus = 0;\n if (char.level >= 10 && hpPenalty === 0) {\n hpBonus = 5;\n reason.push(`Daily wellness bonus (+${hpBonus} HP)`);\n }\n \n return {\n characterId: char.id,\n userId: char.user\_id,\n username: char.username,\n currentHp: char.hp,\n hpPenalty: hpPenalty,\n hpBonus: hpBonus,\n netHpChange: hpBonus - hpPenalty,\n newHp: Math.max(0, Math.min(100, char.hp + hpBonus - hpPenalty)),\n reason: reason.join(', ') || 'No changes',\n shouldUpdate: (hpPenalty > 0 || hpBonus > 0)\n };\n});\n\nreturn penalties.filter(p => p.shouldUpdate).map(p => ({ json: p }));"

},

"id": "calculate\_penalties",

"name": "Calculate Daily Penalties",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [650, 400]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE characters SET hp = $1 WHERE id = $2 RETURNING id, hp",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.newHp }}"

},

{

"parameter": "={{ $json.characterId }}"

}

]

}

}

},

"id": "update\_character\_hp",

"name": "Update Character HP",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [850, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, hp\_change, description) VALUES ($1, $2, $3, $4)",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Calculate Daily Penalties').item.json.characterId }}"

},

{

"parameter": "=daily\_maintenance"

},

{

"parameter": "={{ $('Calculate Daily Penalties').item.json.netHpChange }}"

},

{

"parameter": "=Daily check: {{ $('Calculate Daily Penalties').item.json.reason }}"

}

]

}

}

},

"id": "log\_event",

"name": "Log Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT id, description, effect, rarity FROM rng\_events WHERE available = true AND (last\_issued IS NULL OR last\_issued < CURRENT\_DATE - INTERVAL '7 days') ORDER BY RANDOM() LIMIT 50",

"options": {}

},

"id": "fetch\_available\_events",

"name": "Fetch Available Events",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [850, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "// Generate daily random events for active characters\nconst characters = $('Fetch Active Characters').all().map(item => item.json);\nconst availableEvents = $('Fetch Available Events').all().map(item => item.json);\n\n// Assign 1-3 random events to each character\nconst assignments = [];\n\ncharacters.forEach(char => {\n // 70% chance of getting an event\n if (Math.random() < 0.7) {\n const numEvents = Math.floor(Math.random() \* 3) + 1; // 1-3 events\n \n for (let i = 0; i < numEvents && i < availableEvents.length; i++) {\n const randomEvent = availableEvents[Math.floor(Math.random() \* availableEvents.length)];\n \n assignments.push({\n characterId: char.id,\n eventId: randomEvent.id,\n description: randomEvent.description,\n effect: randomEvent.effect,\n rarity: randomEvent.rarity\n });\n }\n }\n});\n\nreturn assignments.map(a => ({ json: a }));"

},

"id": "generate\_daily\_events",

"name": "Generate Daily Events",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [1050, 200]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, description) VALUES ($1, 'daily\_event', $2); UPDATE rng\_events SET last\_issued = CURRENT\_DATE WHERE id = $3",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.characterId }}"

},

{

"parameter": "={{ $json.description }} - {{ $json.effect }}"

},

{

"parameter": "={{ $json.eventId }}"

}

]

}

}

},

"id": "insert\_daily\_events",

"name": "Insert Daily Events",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE habits SET streak = 0 WHERE last\_completed < CURRENT\_DATE - INTERVAL '2 days' AND type = 'good' RETURNING id, character\_id, name",

"options": {}

},

"id": "reset\_broken\_streaks",

"name": "Reset Broken Streaks",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 600]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, description) VALUES ($1, 'streak\_broken', $2)",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.character\_id }}"

},

{

"parameter": "=Streak broken for habit: {{ $json.name }}"

}

]

}

}

},

"id": "log\_streak\_breaks",

"name": "Log Streak Breaks",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [850, 600],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

}

{

"name": "DAMAGE\_CALC",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "bad-habit-battle",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook\_bad\_habit",

"name": "Webhook - Bad Habit Battle",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1.1,

"position": [250, 300],

"webhookId": "bad-habit-damage"

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT h.\*, c.hp, c.level, c.id as character\_id, s.level as skill\_level FROM habits h JOIN characters c ON h.character\_id = c.id JOIN skills s ON h.skill\_id = s.id WHERE h.id = $1 AND h.type = 'bad'",

"options": {}

},

"id": "fetch\_bad\_habit\_data",

"name": "Fetch Bad Habit Data",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "// Calculate damage with defense modifiers\nconst habitData = $input.first().json;\nconst baseDamage = Math.abs(habitData.hp\_value) || 15;\nconst characterLevel = habitData.level || 1;\nconst skillLevel = habitData.skill\_level || 1;\nconst currentHP = habitData.hp || 100;\n\n// Defense calculation (higher skill level = less damage)\nconst defenseModifier = Math.max(0.5, 1 - (skillLevel \* 0.05));\nconst finalDamage = Math.floor(baseDamage \* defenseModifier);\n\n// Calculate new HP (minimum 0)\nconst newHP = Math.max(0, currentHP - finalDamage);\n\n// Check if character is defeated\nconst isDefeated = newHP === 0;\n\n// Generate battle narrative\nlet battleNarrative = `You faced the ${habitData.name} and took ${finalDamage} damage!`;\nif (isDefeated) {\n battleNarrative += \" You've been defeated and must recover at the Hotel.\";\n} else if (newHP < 30) {\n battleNarrative += \" Your HP is critically low!\";\n}\n\nreturn [{\n json: {\n habitId: habitData.id,\n characterId: habitData.character\_id,\n skillId: habitData.skill\_id,\n damageDealt: finalDamage,\n oldHP: currentHP,\n newHP: newHP,\n isDefeated: isDefeated,\n battleNarrative: battleNarrative,\n defenseModifier: defenseModifier\n }\n}];"

},

"id": "calculate\_damage",

"name": "Calculate Damage",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [650, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE characters SET hp = $1 WHERE id = $2 RETURNING \*",

"options": {}

},

"id": "update\_character\_hp",

"name": "Update Character HP",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [850, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE habits SET last\_completed = CURRENT\_DATE WHERE id = $1 RETURNING \*",

"options": {}

},

"id": "update\_habit\_timestamp",

"name": "Update Habit Timestamp",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, hp\_change, description) VALUES ($1, 'bad\_habit\_battle', $2, $3) RETURNING \*",

"options": {}

},

"id": "log\_battle\_event",

"name": "Log Battle Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('user', $1, 'habit', $2, 'bad\_habit\_battle', $3, $4, $5, 'web') RETURNING \*",

"options": {}

},

"id": "log\_system\_battle",

"name": "Log System Battle",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": true,\n \"damageDealt\": $('Calculate Damage').item.json.damageDealt,\n \"newHP\": $('Calculate Damage').item.json.newHP,\n \"oldHP\": $('Calculate Damage').item.json.oldHP,\n \"isDefeated\": $('Calculate Damage').item.json.isDefeated,\n \"narrative\": $('Calculate Damage').item.json.battleNarrative,\n \"animationTrigger\": true\n} }}",

"options": {}

},

"id": "respond\_battle\_result",

"name": "Respond Battle Result",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [1650, 300]

}

],

"pinData": {},

"connections": {

"Webhook - Bad Habit Battle": {

"main": [

[

{

"node": "Fetch Bad Habit Data",

"type": "main",

"index": 0

}

]

]

},

"Fetch Bad Habit Data": {

"main": [

[

{

"node": "Calculate Damage",

"type": "main",

"index": 0

}

]

]

},

"Calculate Damage": {

"main": [

[

{

"node": "Update Character HP",

"type": "main",

"index": 0

}

]

]

},

"Update Character HP": {

"main": [

[

{

"node": "Update Habit Timestamp",

"type": "main",

"index": 0

}

]

]

},

"Update Habit Timestamp": {

"main": [

[

{

"node": "Log Battle Event",

"type": "main",

"index": 0

}

]

]

},

"Log Battle Event": {

"main": [

[

{

"node": "Log System Battle",

"type": "main",

"index": 0

}

]

]

},

"Log System Battle": {

"main": [

[

{

"node": "Respond Battle Result",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"executionOrder": "v1"

},

"versionId": "1",

"meta": {

"instanceId": "life-game-production"

},

"id": "3",

"tags": []

}

{

"name": "EVENT\_SEEDER",

"nodes": [

{

"parameters": {

"rule": {

"interval": [

{

"field": "cronExpression",

"expression": "0 0 1 \* \*"

}

]

}

},

"id": "schedule\_monthly",

"name": "Schedule - Monthly (1st)",

"type": "n8n-nodes-base.scheduleTrigger",

"typeVersion": 1.2,

"position": [250, 400]

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT COUNT(\*) as total\_events, COUNT(CASE WHEN available = true THEN 1 END) as available\_events FROM rng\_events",

"options": {}

},

"id": "check\_event\_pool",

"name": "Check Event Pool",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"authentication": "predefinedCredentialType",

"nodeCredentialType": "openAiApi",

"promptType": "define",

"text": "=You are a creative game designer for a life-gamification app. Generate 20 diverse random events for players.\n\nEvent Pool Status:\n- Total events in database: {{ $('Check Event Pool').item.json.total\_events }}\n- Currently available: {{ $('Check Event Pool').item.json.available\_events }}\n\nCreate events that are:\n1. Diverse in nature (positive, negative, neutral, mysterious)\n2. Varied in rarity (common, uncommon, rare, legendary)\n3. Thematically appropriate for personal growth and productivity\n4. Fun, engaging, and sometimes humorous\n5. Include specific mechanical effects (HP, XP, coins, or special bonuses)\n\nEvent Categories to include:\n- Fortune events (lucky finds, bonus rewards)\n- Misfortune events (setbacks, small penalties)\n- Mystery events (random outcomes)\n- Wisdom events (reflective, philosophical)\n- Social events (community interactions)\n- Seasonal events (tied to time of year)\n\nRespond ONLY with a JSON array in this exact format:\n[\n {\n \"description\": \"You found a lucky coin while cleaning your workspace!\",\n \"effect\": \"+15 coins\",\n \"rarity\": \"common\"\n },\n {\n \"description\": \"A mysterious mentor appears and shares ancient wisdom\",\n \"effect\": \"+50 XP to highest skill\",\n \"rarity\": \"rare\"\n },\n {\n \"description\": \"You overslept and missed your morning routine\",\n \"effect\": \"-10 HP\",\n \"rarity\": \"uncommon\"\n }\n]\n\nGenerate 20 unique, creative events now.",

"options": {

"temperature": 0.9,

"maxTokens": 2000

}

},

"id": "generate\_events\_ai",

"name": "Generate Events (AI)",

"type": "@n8n/n8n-nodes-langchain.openAi",

"typeVersion": 1.3,

"position": [650, 400],

"credentials": {

"openAiApi": {

"id": "2",

"name": "OpenAI account"

}

}

},

{

"parameters": {

"jsCode": "// Parse AI response and validate events\nconst response = $input.item.json.output || $input.item.json.response || $input.item.json.text;\nlet events = [];\n\ntry {\n const parsed = JSON.parse(response);\n events = Array.isArray(parsed) ? parsed : [parsed];\n} catch (e) {\n const jsonMatch = response.match(/```json\\s\*([\\s\\S]\*?)```/);\n if (jsonMatch) {\n events = JSON.parse(jsonMatch[1]);\n } else {\n const arrayMatch = response.match(/\\[\\s\*\\{[\\s\\S]\*\\}\\s\*\\]/);\n if (arrayMatch) {\n events = JSON.parse(arrayMatch[0]);\n }\n }\n}\n\nfunction detectCategory(description, effect) {\n const desc = description.toLowerCase();\n const eff = effect.toLowerCase();\n \n if (desc.includes('found') || desc.includes('lucky') || eff.includes('+')) {\n return 'fortune';\n } else if (desc.includes('missed') || desc.includes('lost') || eff.includes('-')) {\n return 'misfortune';\n } else if (desc.includes('mystery') || desc.includes('unknown')) {\n return 'mystery';\n } else if (desc.includes('wisdom') || desc.includes('reflect') || desc.includes('learn')) {\n return 'wisdom';\n } else if (desc.includes('friend') || desc.includes('guild') || desc.includes('community')) {\n return 'social';\n } else {\n return 'general';\n }\n}\n\nconst validatedEvents = events.map((event, index) => {\n let rarity = (event.rarity || 'common').toLowerCase();\n if (!['common', 'uncommon', 'rare', 'legendary'].includes(rarity)) {\n rarity = 'common';\n }\n \n return {\n description: event.description || `Random event ${index + 1}`,\n effect: event.effect || 'No effect',\n rarity: rarity,\n available: true,\n category: event.category || detectCategory(event.description, event.effect)\n };\n});\n\nreturn validatedEvents.map(e => ({ json: e }));"

},

"id": "parse\_and\_validate",

"name": "Parse and Validate Events",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [850, 400]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO rng\_events (description, effect, rarity, available) VALUES ($1, $2, $3, $4) ON CONFLICT DO NOTHING RETURNING id",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.description }}"

},

{

"parameter": "={{ $json.effect }}"

},

{

"parameter": "={{ $json.rarity }}"

},

{

"parameter": true

}

]

}

}

},

"id": "insert\_events",

"name": "Insert Events",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE rng\_events SET available = false WHERE last\_issued < NOW() - INTERVAL '60 days' AND available = true RETURNING id, description",

"options": {}

},

"id": "retire\_old\_events",

"name": "Retire Old Events",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 500],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT rarity, COUNT(\*) as count FROM rng\_events WHERE available = true GROUP BY rarity",

"options": {}

},

"id": "analyze\_distribution",

"name": "Analyze Event Distribution",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "const insertedEvents = $('Insert Events').all();\nconst retiredEvents = $('Retire Old Events').all();\nconst distribution = $('Analyze Event Distribution').all().map(item => item.json);\n\nconst summary = {\n timestamp: new Date().toISOString(),\n eventsAdded: insertedEvents.length,\n eventsRetired: retiredEvents.length,\n currentDistribution: distribution.reduce((acc, d) => {\n acc[d.rarity] = parseInt(d.count);\n return acc;\n }, {}),\n totalActiveEvents: distribution.reduce((sum, d) => sum + parseInt(d.count), 0)\n};\n\nreturn [{ json: summary }];"

},

"id": "create\_summary",

"name": "Create Summary",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [1450, 400]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('system', 0, 'rng\_events', 0, 'event\_seeder\_monthly', $1, 'success', 'info', 'n8n')",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ JSON.stringify($json) }}"

}

]

}

}

},

"id": "log\_system",

"name": "Log System",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1650, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT c.id, c.user\_id FROM characters c WHERE c.last\_login > NOW() - INTERVAL '7 days' LIMIT 100",

"options": {}

},

"id": "fetch\_active\_users",

"name": "Fetch Active Users for Notification",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, description) VALUES ($1, 'new\_events\_available', 'New random events have been added to the world! Check your daily surprises.')",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.id }}"

}

]

}

}

},

"id": "notify\_users",

"name": "Notify Users",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "const now = new Date();\nconst month = now.getMonth() + 1;\nconst seasonalEvents = [];\n\nconst seasons = {\n 1: { name: 'New Year', theme: 'fresh starts and resolutions' },\n 2: { name: 'Winter', theme: 'cozy indoor activities' },\n 3: { name: 'Spring', theme: 'renewal and growth' },\n 4: { name: 'Spring', theme: 'renewal and growth' },\n 5: { name: 'Spring', theme: 'renewal and growth' },\n 6: { name: 'Summer', theme: 'outdoor adventures' },\n 7: { name: 'Summer', theme: 'outdoor adventures' },\n 8: { name: 'Summer', theme: 'outdoor adventures' },\n 9: { name: 'Autumn', theme: 'harvest and preparation' },\n 10: { name: 'Autumn', theme: 'harvest and preparation' },\n 11: { name: 'Autumn', theme: 'harvest and preparation' },\n 12: { name: 'Winter Holiday', theme: 'celebration and reflection' }\n};\n\nconst currentSeason = seasons[month];\n\nseasonalEvents.push({\n description: `${currentSeason.name} Special: The season of ${currentSeason.theme} brings you inspiration!`,\n effect: '+25 XP',\n rarity: 'uncommon',\n available: true\n});\n\nseasonalEvents.push({\n description: `${currentSeason.name} Bonus: You feel energized by the ${currentSeason.name} atmosphere!`,\n effect: '+10 HP, +10 coins',\n rarity: 'rare',\n available: true\n});\n\nseasonalEvents.push({\n description: `${currentSeason.name} Challenge: Embrace the spirit of ${currentSeason.theme}!`,\n effect: 'Random skill gains +15 XP',\n rarity: 'common',\n available: true\n});\n\nreturn seasonalEvents.map(e => ({ json: e }));"

},

"id": "generate\_seasonal",

"name": "Generate Seasonal Events",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [650, 600]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO rng\_events (description, effect, rarity, available) VALUES ($1, $2, $3, $4) RETURNING id",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.description }}"

},

{

"parameter": "={{ $json.effect }}"

},

{

"parameter": "={{ $json.rarity }}"

},

{

"parameter": true

}

]

}

}

},

"id": "insert\_seasonal",

"name": "Insert Seasonal Events",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [850, 600],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "DELETE FROM rng\_events WHERE description LIKE '%Special:%' AND last\_issued < NOW() - INTERVAL '40 days'",

"options": {}

},

"id": "cleanup\_old\_seasonal",

"name": "Cleanup Old Seasonal Events",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 600],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

}

],

"connections": {

"Schedule - Monthly (1st)": {

"main": [

[

{

"node": "Check Event Pool",

"type": "main",

"index": 0

},

{

"node": "Fetch Active Users for Notification",

"type": "main",

"index": 0

},

{

"node": "Cleanup Old Seasonal Events",

"type": "main",

"index": 0

}

]

]

},

"Check Event Pool": {

"main": [

[

{

"node": "Generate Events (AI)",

"type": "main",

"index": 0

}

]

]

},

"Generate Events (AI)": {

"main": [

[

{

"node": "Parse and Validate Events",

"type": "main",

"index": 0

}

]

]

},

"Parse and Validate Events": {

"main": [

[

{

"node": "Insert Events",

"type": "main",

"index": 0

},

{

"node": "Retire Old Events",

"type": "main",

"index": 0

}

]

]

},

"Insert Events": {

"main": [

[

{

"node": "Analyze Event Distribution",

"type": "main",

"index": 0

}

]

]

},

"Retire Old Events": {

"main": [

[

{

"node": "Analyze Event Distribution",

"type": "main",

"index": 0

}

]

]

},

"Analyze Event Distribution": {

"main": [

[

{

"node": "Create Summary",

"type": "main",

"index": 0

}

]

]

},

"Create Summary": {

"main": [

[

{

"node": "Log System",

"type": "main",

"index": 0

}

]

]

},

"Fetch Active Users for Notification": {

"main": [

[

{

"node": "Notify Users",

"type": "main",

"index": 0

}

]

]

},

"Cleanup Old Seasonal Events": {

"main": [

[

{

"node": "Generate Seasonal Events",

"type": "main",

"index": 0

}

]

]

},

"Generate Seasonal Events": {

"main": [

[

{

"node": "Insert Seasonal Events",

"type": "main",

"index": 0

}

]

]

}

},

"settings": {

"executionOrder": "v1"

}

}

{

"name": "HABIT\_CHECKIN",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "habit-checkin",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook\_habit\_checkin",

"name": "Webhook - Habit Check-in",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1.1,

"position": [250, 300],

"webhookId": "habit-checkin"

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT h.\*, s.level as skill\_level, c.level as character\_level, c.id as character\_id FROM habits h JOIN skills s ON h.skill\_id = s.id JOIN characters c ON h.character\_id = c.id WHERE h.id = $1",

"options": {}

},

"id": "fetch\_habit\_data",

"name": "Fetch Habit & Character Data",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "// Calculate rewards with streak multipliers\nconst habitData = $input.first().json;\nconst baseXP = habitData.xp\_value || 10;\nconst baseCoins = Math.floor(baseXP \* 0.5);\nconst currentStreak = habitData.streak || 0;\nconst lastCompleted = habitData.last\_completed;\nconst today = new Date().toISOString().split('T')[0];\n\n// Check if completed today already\nif (lastCompleted === today) {\n return [{\n json: {\n alreadyCompleted: true,\n message: \"Habit already completed today\"\n }\n }];\n}\n\n// Calculate streak bonus\nlet newStreak = currentStreak + 1;\nlet streakMultiplier = 1.0;\n\nif (newStreak >= 7) streakMultiplier = 1.5;\nelse if (newStreak >= 30) streakMultiplier = 2.0;\nelse if (newStreak >= 90) streakMultiplier = 3.0;\n\nconst finalXP = Math.floor(baseXP \* streakMultiplier);\nconst finalCoins = Math.floor(baseCoins \* streakMultiplier);\n\n// Skill XP calculation (40% of habit XP)\nconst skillXP = Math.floor(finalXP \* 0.4);\n\nreturn [{\n json: {\n habitId: habitData.id,\n characterId: habitData.character\_id,\n skillId: habitData.skill\_id,\n xpEarned: finalXP,\n coinsEarned: finalCoins,\n skillXPEarned: skillXP,\n newStreak: newStreak,\n streakMultiplier: streakMultiplier,\n alreadyCompleted: false\n }\n}];"

},

"id": "calculate\_rewards",

"name": "Calculate Rewards",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [650, 300]

},

{

"parameters": {

"conditions": {

"boolean": [

{

"value1": "={{ $json.alreadyCompleted }}",

"value2": true

}

]

}

},

"id": "check\_already\_completed",

"name": "Already Completed?",

"type": "n8n-nodes-base.if",

"typeVersion": 2,

"position": [850, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE habits SET streak = $1, last\_completed = CURRENT\_DATE WHERE id = $2 RETURNING \*",

"options": {}

},

"id": "update\_habit\_streak",

"name": "Update Habit Streak",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE skills SET xp = xp + $1, level = FLOOR(POWER(xp / 100, 0.5)) + 1 WHERE id = $2 RETURNING \*",

"options": {}

},

"id": "update\_skill\_xp",

"name": "Update Skill XP",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE characters SET xp = xp + $1, coins = coins + $2, level = FLOOR(POWER(xp / 100, 0.66)) + 1 WHERE id = $3 RETURNING \*",

"options": {}

},

"id": "update\_character\_stats",

"name": "Update Character Stats",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1450, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, xp\_change, coins\_change, description) VALUES ($1, 'habit\_completed', $2, $3, $4) RETURNING \*",

"options": {}

},

"id": "log\_habit\_event",

"name": "Log Habit Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1650, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('user', $1, 'habit', $2, 'habit\_checkin', $3, 'success', 'info', 'web') RETURNING \*",

"options": {}

},

"id": "log\_system",

"name": "Log System",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1850, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": true,\n \"xpEarned\": $('Calculate Rewards').item.json.xpEarned,\n \"coinsEarned\": $('Calculate Rewards').item.json.coinsEarned,\n \"newStreak\": $('Calculate Rewards').item.json.newStreak,\n \"streakBonus\": $('Calculate Rewards').item.json.streakMultiplier,\n \"message\": \"Great work! Keep the momentum going!\"\n} }}",

"options": {}

},

"id": "respond\_success",

"name": "Respond Success",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [2050, 200]

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\"success\": false, \"message\": $json.message} }}",

"options": {}

},

"id": "respond\_already\_done",

"name": "Respond Already Done",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [1050, 400]

}

],

"pinData": {},

"connections": {

"Webhook - Habit Check-in": {

"main": [

[

{

"node": "Fetch Habit & Character Data",

"type": "main",

"index": 0

}

]

]

},

"Fetch Habit & Character Data": {

"main": [

[

{

"node": "Calculate Rewards",

"type": "main",

"index": 0

}

]

]

},

"Calculate Rewards": {

"main": [

[

{

"node": "Already Completed?",

"type": "main",

"index": 0

}

]

]

},

"Already Completed?": {

"main": [

[

{

"node": "Respond Already Done",

"type": "main",

"index": 0

}

],

[

{

"node": "Update Habit Streak",

"type": "main",

"index": 0

}

]

]

},

"Update Habit Streak": {

"main": [

[

{

"node": "Update Skill XP",

"type": "main",

"index": 0

}

]

]

},

"Update Skill XP": {

"main": [

[

{

"node": "Update Character Stats",

"type": "main",

"index": 0

}

]

]

},

"Update Character Stats": {

"main": [

[

{

"node": "Log Habit Event",

"type": "main",

"index": 0

}

]

]

},

"Log Habit Event": {

"main": [

[

{

"node": "Log System",

"type": "main",

"index": 0

}

]

]

},

"Log System": {

"main": [

[

{

"node": "Respond Success",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"executionOrder": "v1"

},

"versionId": "1",

"meta": {

"instanceId": "life-game-production"

},

"id": "2",

"tags": []

}

{

"name": "INIT\_USER\_SETUP",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "user-signup",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook\_trigger",

"name": "Webhook - New User Signup",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1.1,

"position": [250, 300],

"webhookId": "user-signup-init"

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO characters (user\_id, class, bio, goals, level, xp, hp, coins, prestige\_level, last\_login) VALUES ($1, $2, $3, $4, 1, 0, 100, 100, 0, NOW()) RETURNING \*",

"options": {}

},

"id": "create\_character",

"name": "Create Character",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"assignments": {

"assignments": [

{

"id": "default\_skills",

"name": "defaultSkills",

"value": "={{ [{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}] }}",

"type": "array"

},

{

"id": "character\_id",

"name": "characterId",

"value": "={{ $json.id }}",

"type": "number"

}

]

},

"options": {}

},

"id": "prepare\_default\_skills",

"name": "Prepare Default Skills",

"type": "n8n-nodes-base.set",

"typeVersion": 3.3,

"position": [650, 300]

},

{

"parameters": {

"mode": "multiplex",

"options": {}

},

"id": "split\_skills",

"name": "Split Skills Array",

"type": "n8n-nodes-base.splitOut",

"typeVersion": 1,

"position": [850, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO skills (character\_id, name, xp, level, unlocked) VALUES ($1, $2, $3, $4, $5) RETURNING \*",

"options": {}

},

"id": "insert\_skills",

"name": "Insert Skills",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO projects (character\_id, title, description, total\_xp, coin\_reward, difficulty, completed) VALUES ($1, 'Welcome to Your Adventure', 'Complete your first steps in the Land of Growth. Check in your first habit, explore your character stats, and learn how XP and HP work.', 50, 25, 'tutorial', false) RETURNING \*",

"options": {}

},

"id": "create\_tutorial\_quest",

"name": "Create Tutorial Quest",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 500],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"assignments": {

"assignments": [

{

"id": "tutorial\_tasks",

"name": "tutorialTasks",

"value": "={{ [{title: 'View Your Character Stats', xp: 10, coins: 5, difficulty: 'easy'}, {title: 'Create Your First Habit', xp: 15, coins: 10, difficulty: 'easy'}, {title: 'Complete a Habit Check-in', xp: 25, coins: 10, difficulty: 'easy'}] }}",

"type": "array"

},

{

"id": "project\_id",

"name": "projectId",

"value": "={{ $json.id }}",

"type": "number"

}

]

},

"options": {}

},

"id": "prepare\_tutorial\_tasks",

"name": "Prepare Tutorial Tasks",

"type": "n8n-nodes-base.set",

"typeVersion": 3.3,

"position": [1250, 500]

},

{

"parameters": {

"mode": "multiplex",

"options": {}

},

"id": "split\_tasks",

"name": "Split Tasks Array",

"type": "n8n-nodes-base.splitOut",

"typeVersion": 1,

"position": [1450, 500]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO tasks (project\_id, title, completed, xp, coins, difficulty) VALUES ($1, $2, false, $3, $4, $5) RETURNING \*",

"options": {}

},

"id": "insert\_tutorial\_tasks",

"name": "Insert Tutorial Tasks",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1650, 500],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO settings (user\_id, level\_xp\_formula, overdraft\_rule, notification\_times, theme) VALUES ($1, '100 \* level^1.5', 'Weekly HP -10 if coins < 0', '09:00,18:00', 'default') RETURNING \*",

"options": {}

},

"id": "create\_settings",

"name": "Create Default Settings",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 700],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"assignments": {

"assignments": [

{

"id": "starter\_items",

"name": "starterItems",

"value": "={{ [{item\_id: 1, quantity: 1}, {item\_id: 2, quantity: 3}] }}",

"type": "array"

}

]

},

"options": {}

},

"id": "prepare\_starter\_items",

"name": "Prepare Starter Items",

"type": "n8n-nodes-base.set",

"typeVersion": 3.3,

"position": [1250, 700]

},

{

"parameters": {

"mode": "multiplex",

"options": {}

},

"id": "split\_items",

"name": "Split Items Array",

"type": "n8n-nodes-base.splitOut",

"typeVersion": 1,

"position": [1450, 700]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO inventory (character\_id, item\_id, quantity) VALUES ($1, $2, $3) RETURNING \*",

"options": {}

},

"id": "insert\_starter\_items",

"name": "Insert Starter Items",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1650, 700],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, xp\_change, hp\_change, coins\_change, description) VALUES ($1, 'onboarding', 0, 0, 100, 'Welcome! Your journey begins.') RETURNING \*",

"options": {}

},

"id": "log\_onboarding\_event",

"name": "Log Onboarding Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1850, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('system', 0, 'user', $1, 'user\_onboarding\_complete', $2, 'success', 'info', 'n8n') RETURNING \*",

"options": {}

},

"id": "log\_system\_event",

"name": "Log System Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\"success\": true, \"characterId\": $('Create Character').item.json.id, \"message\": \"Account setup complete! Welcome to the Land of Growth.\"} }}",

"options": {}

},

"id": "respond\_success",

"name": "Respond Success",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [2250, 300]

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\"success\": false, \"error\": $json.message} }}",

"options": {

"responseCode": 500

}

},

"id": "respond\_error",

"name": "Respond Error",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [2250, 500]

}

],

"pinData": {},

"connections": {

"Webhook - New User Signup": {

"main": [

[

{

"node": "Create Character",

"type": "main",

"index": 0

}

]

]

},

"Create Character": {

"main": [

[

{

"node": "Prepare Default Skills",

"type": "main",

"index": 0

},

{

"node": "Create Tutorial Quest",

"type": "main",

"index": 0

},

{

"node": "Create Default Settings",

"type": "main",

"index": 0

}

]

]

},

"Prepare Default Skills": {

"main": [

[

{

"node": "Split Skills Array",

"type": "main",

"index": 0

}

]

]

},

"Split Skills Array": {

"main": [

[

{

"node": "Insert Skills",

"type": "main",

"index": 0

}

]

]

},

"Insert Skills": {

"main": [

[

{

"node": "Log Onboarding Event",

"type": "main",

"index": 0

}

]

]

},

"Create Tutorial Quest": {

"main": [

[

{

"node": "Prepare Tutorial Tasks",

"type": "main",

"index": 0

}

]

]

},

"Prepare Tutorial Tasks": {

"main": [

[

{

"node": "Split Tasks Array",

"type": "main",

"index": 0

}

]

]

},

"Split Tasks Array": {

"main": [

[

{

"node": "Insert Tutorial Tasks",

"type": "main",

"index": 0

}

]

]

},

"Create Default Settings": {

"main": [

[

{

"node": "Prepare Starter Items",

"type": "main",

"index": 0

}

]

]

},

"Prepare Starter Items": {

"main": [

[

{

"node": "Split Items Array",

"type": "main",

"index": 0

}

]

]

},

"Split Items Array": {

"main": [

[

{

"node": "Insert Starter Items",

"type": "main",

"index": 0

}

]

]

},

"Log Onboarding Event": {

"main": [

[

{

"node": "Log System Event",

"type": "main",

"index": 0

}

]

]

},

"Log System Event": {

"main": [

[

{

"node": "Respond Success",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"executionOrder": "v1"

},

"versionId": "1",

"meta": {

"instanceId": "life-game-production"

},

"id": "1",

"tags": []

}

{

"name": "PRESTIGE\_CALC",

"nodes": [

{

"parameters": {

"triggerOn": "update",

"tableName": "Characters",

"additionalFields": {

"where": "level >= {{$json[\"max\_level\"]}}"

}

},

"id": "trigger\_level\_max",

"name": "Trigger: Level Max Reached",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [250, 300],

"webhookId": "prestige\_trigger"

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT c.\*, u.username, u.email, u.total\_prestiges FROM Characters c JOIN Users u ON c.user\_id = u.id WHERE c.id = {{$json[\"id\"]}}"

},

"id": "fetch\_character\_data",

"name": "Fetch Character & User Data",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [450, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT \* FROM Skills WHERE user\_id = {{$json[\"user\_id\"]}}"

},

"id": "fetch\_current\_skills",

"name": "Fetch Current Skills",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [650, 300]

},

{

"parameters": {

"functionCode": "// REWARD CALCULATION MODULE (Reusable)\nconst items = $input.all();\nconst character = items[0].json;\nconst skills = items[1] ? items[1].json : [];\n\n// Calculate prestige bonus based on current stats\nconst currentPrestigeCount = character.total\_prestiges || 0;\nconst prestigeLevel = currentPrestigeCount + 1;\n\n// Prestige Bonus Formula\nconst baseHPBonus = 10;\nconst baseXPMultiplier = 0.05; // 5% per prestige\nconst baseCoinBonus = 100;\n\n// Calculate bonuses\nconst prestigeBonus = {\n hp\_bonus: baseHPBonus \* prestigeLevel,\n xp\_multiplier: 1 + (baseXPMultiplier \* prestigeLevel),\n coin\_bonus: baseCoinBonus \* prestigeLevel,\n prestige\_level: prestigeLevel,\n permanent\_perk: `prestige\_${prestigeLevel}`\n};\n\n// Calculate skill retention (keep 10% of current skill XP)\nconst skillRetention = skills.map(skill => ({\n skill\_name: skill.skill\_name,\n retained\_xp: Math.floor(skill.xp \* 0.1),\n original\_xp: skill.xp\n}));\n\n// Store pre-prestige stats for AI context\nconst prePrestigeStats = {\n level: character.level,\n total\_xp: character.total\_xp,\n hp: character.hp,\n coins: character.coins,\n skills: skills.length\n};\n\nreturn {\n json: {\n user\_id: character.user\_id,\n character\_id: character.id,\n username: character.username,\n prestige\_bonus: prestigeBonus,\n skill\_retention: skillRetention,\n pre\_prestige\_stats: prePrestigeStats,\n timestamp: new Date().toISOString()\n }\n};"

},

"id": "calculate\_prestige\_bonus",

"name": "Calculate Prestige Bonus",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [850, 300]

},

{

"parameters": {

"authentication": "headerAuth",

"method": "POST",

"url": "https://api.openai.com/v1/chat/completions",

"options": {

"timeout": 30000

},

"sendBody": true,

"bodyParameters": {

"parameters": [

{

"name": "model",

"value": "gpt-4"

},

{

"name": "messages",

"value": "=[{\n \"role\": \"system\",\n \"content\": \"You are an epic fantasy narrator for a productivity RPG game. Generate inspiring prestige messages and titles.\"\n}, {\n \"role\": \"user\",\n \"content\": `Generate a prestige celebration message and title for ${$json.username} who has reached Prestige Level ${$json.prestige\_bonus.prestige\_level}. Their pre-prestige stats: Level ${$json.pre\_prestige\_stats.level}, ${$json.pre\_prestige\_stats.total\_xp} total XP, ${$json.pre\_prestige\_stats.skills} skills mastered. Make it epic and motivational. Return as JSON with keys: 'title' (a prestigious rank/title), 'message' (2-3 sentences celebration), 'quote' (inspirational quote).`\n}]"

},

{

"name": "temperature",

"value": "0.8"

},

{

"name": "max\_tokens",

"value": "300"

}

]

},

"headerParameters": {

"parameters": [

{

"name": "Authorization",

"value": "Bearer {{$credentials.openAiApi.apiKey}}"

},

{

"name": "Content-Type",

"value": "application/json"

}

]

}

},

"id": "ai\_generate\_prestige\_message",

"name": "AI: Generate Prestige Message",

"type": "n8n-nodes-base.httpRequest",

"typeVersion": 4,

"position": [1050, 300],

"credentials": {

"openAiApi": {

"id": "1",

"name": "OpenAI API"

}

}

},

{

"parameters": {

"functionCode": "// Parse AI response\nconst aiResponse = $json.choices[0].message.content;\nlet parsedAI;\n\ntry {\n parsedAI = JSON.parse(aiResponse);\n} catch (e) {\n // Fallback if AI doesn't return proper JSON\n parsedAI = {\n title: `Prestige Master ${$node[\"Calculate Prestige Bonus\"].json.prestige\_bonus.prestige\_level}`,\n message: `Congratulations on reaching Prestige Level ${$node[\"Calculate Prestige Bonus\"].json.prestige\_bonus.prestige\_level}! Your journey begins anew, stronger than before.`,\n quote: \"Every end is a new beginning.\"\n };\n}\n\nreturn {\n json: {\n ...$node[\"Calculate Prestige Bonus\"].json,\n ai\_content: parsedAI\n }\n};"

},

"id": "parse\_ai\_response",

"name": "Parse AI Response",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [1250, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE Characters SET level = 1, xp = 0, total\_xp = total\_xp, hp = 100 + {{$json.prestige\_bonus.hp\_bonus}}, max\_hp = 100 + {{$json.prestige\_bonus.hp\_bonus}}, prestige\_level = {{$json.prestige\_bonus.prestige\_level}}, xp\_multiplier = {{$json.prestige\_bonus.xp\_multiplier}}, updated\_at = NOW() WHERE id = {{$json.character\_id}} RETURNING \*"

},

"id": "reset\_character\_stats",

"name": "Reset Character Stats",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1450, 200]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE Users SET total\_prestiges = total\_prestiges + 1, coins = coins + {{$json.prestige\_bonus.coin\_bonus}}, updated\_at = NOW() WHERE id = {{$json.user\_id}} RETURNING \*"

},

"id": "update\_user\_prestige",

"name": "Update User Prestige Count",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1450, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE Skills SET xp = {{$json.skill\_retention[0].retained\_xp}}, level = 1 WHERE user\_id = {{$json.user\_id}} AND skill\_name = '{{$json.skill\_retention[0].skill\_name}}'"

},

"id": "reset\_skills\_with\_retention",

"name": "Reset Skills (with Retention)",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1450, 400],

"executeOnce": false

},

{

"parameters": {

"operation": "insert",

"table": "Achievements",

"columns": "user\_id, achievement\_type, achievement\_name, achievement\_description, badge\_icon, unlocked\_at",

"values": "={{$json.user\_id}}, 'prestige', '{{$json.ai\_content.title}}', '{{$json.ai\_content.message}}', 'prestige\_{{$json.prestige\_bonus.prestige\_level}}.png', NOW()"

},

"id": "grant\_prestige\_achievement",

"name": "Grant Prestige Achievement",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1650, 200]

},

{

"parameters": {

"operation": "insert",

"table": "Inventory",

"columns": "user\_id, item\_name, item\_type, item\_description, quantity, acquired\_at",

"values": "={{$json.user\_id}}, 'Prestige Token {{$json.prestige\_bonus.prestige\_level}}', 'token', 'Permanent +{{$json.prestige\_bonus.hp\_bonus}} Max HP, {{Math.round(($json.prestige\_bonus.xp\_multiplier - 1) \* 100)}}% XP Bonus', 1, NOW()"

},

"id": "add\_prestige\_token",

"name": "Add Prestige Token to Inventory",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1650, 300]

},

{

"parameters": {

"operation": "insert",

"table": "Events",

"columns": "user\_id, event\_type, event\_category, event\_data, created\_at",

"values": "={{$json.user\_id}}, 'prestige\_unlock', 'progression', '{\"prestige\_level\": {{$json.prestige\_bonus.prestige\_level}}, \"hp\_bonus\": {{$json.prestige\_bonus.hp\_bonus}}, \"xp\_multiplier\": {{$json.prestige\_bonus.xp\_multiplier}}, \"coin\_bonus\": {{$json.prestige\_bonus.coin\_bonus}}, \"title\": \"{{$json.ai\_content.title}}\", \"message\": \"{{$json.ai\_content.message}}\", \"quote\": \"{{$json.ai\_content.quote}}\"}', NOW()"

},

"id": "log\_prestige\_event",

"name": "Log Prestige Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1650, 400]

},

{

"parameters": {

"authentication": "headerAuth",

"method": "POST",

"url": "={{$json.notification\_webhook\_url || 'https://your-app.com/api/notifications'}}",

"options": {},

"sendBody": true,

"bodyParameters": {

"parameters": [

{

"name": "user\_id",

"value": "={{$json.user\_id}}"

},

{

"name": "type",

"value": "prestige\_celebration"

},

{

"name": "title",

"value": "={{$json.ai\_content.title}}"

},

{

"name": "message",

"value": "={{$json.ai\_content.message}}"

},

{

"name": "quote",

"value": "={{$json.ai\_content.quote}}"

},

{

"name": "prestige\_level",

"value": "={{$json.prestige\_bonus.prestige\_level}}"

},

{

"name": "bonuses",

"value": "={{$json.prestige\_bonus}}"

}

]

}

},

"id": "notify\_user\_frontend",

"name": "Notify User (Frontend)",

"type": "n8n-nodes-base.httpRequest",

"typeVersion": 4,

"position": [1850, 300]

},

{

"parameters": {

"functionCode": "// Final summary output\nreturn {\n json: {\n success: true,\n user\_id: $json.user\_id,\n prestige\_level: $json.prestige\_bonus.prestige\_level,\n title\_earned: $json.ai\_content.title,\n bonuses\_applied: {\n hp\_bonus: $json.prestige\_bonus.hp\_bonus,\n xp\_multiplier: $json.prestige\_bonus.xp\_multiplier,\n coin\_bonus: $json.prestige\_bonus.coin\_bonus\n },\n message: $json.ai\_content.message,\n timestamp: new Date().toISOString()\n }\n};"

},

"id": "output\_summary",

"name": "Output Summary",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [2050, 300]

}

],

"connections": {

"Trigger: Level Max Reached": {

"main": [

[

{

"node": "Fetch Character & User Data",

"type": "main",

"index": 0

}

]

]

},

"Fetch Character & User Data": {

"main": [

[

{

"node": "Fetch Current Skills",

"type": "main",

"index": 0

}

]

]

},

"Fetch Current Skills": {

"main": [

[

{

"node": "Calculate Prestige Bonus",

"type": "main",

"index": 0

}

]

]

},

"Calculate Prestige Bonus": {

"main": [

[

{

"node": "AI: Generate Prestige Message",

"type": "main",

"index": 0

}

]

]

},

"AI: Generate Prestige Message": {

"main": [

[

{

"node": "Parse AI Response",

"type": "main",

"index": 0

}

]

]

},

"Parse AI Response": {

"main": [

[

{

"node": "Reset Character Stats",

"type": "main",

"index": 0

},

{

"node": "Update User Prestige Count",

"type": "main",

"index": 0

},

{

"node": "Reset Skills (with Retention)",

"type": "main",

"index": 0

}

]

]

},

"Reset Character Stats": {

"main": [

[

{

"node": "Grant Prestige Achievement",

"type": "main",

"index": 0

}

]

]

},

"Update User Prestige Count": {

"main": [

[

{

"node": "Add Prestige Token to Inventory",

"type": "main",

"index": 0

}

]

]

},

"Reset Skills (with Retention)": {

"main": [

[

{

"node": "Log Prestige Event",

"type": "main",

"index": 0

}

]

]

},

"Grant Prestige Achievement": {

"main": [

[

{

"node": "Notify User (Frontend)",

"type": "main",

"index": 0

}

]

]

},

"Add Prestige Token to Inventory": {

"main": [

[

{

"node": "Notify User (Frontend)",

"type": "main",

"index": 0

}

]

]

},

"Log Prestige Event": {

"main": [

[

{

"node": "Notify User (Frontend)",

"type": "main",

"index": 0

}

]

]

},

"Notify User (Frontend)": {

"main": [

[

{

"node": "Output Summary",

"type": "main",

"index": 0

}

]

]

}

},

"settings": {

"executionOrder": "v1"

},

"staticData": null,

"tags": [],

"triggerCount": 1,

"updatedAt": "2025-10-27T00:00:00.000Z",

"versionId": "1"

}

{

"name": "QUEST\_ENGINE",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "complete-task",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook\_task\_complete",

"name": "Webhook - Task Complete",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1.1,

"position": [250, 300],

"webhookId": "task-complete"

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT t.\*, p.title as project\_title, p.difficulty as project\_difficulty, p.area\_id, c.level as character\_level, c.id as character\_id FROM tasks t JOIN projects p ON t.project\_id = p.id JOIN characters c ON p.character\_id = c.id WHERE t.id = $1 AND t.completed = false",

"options": {}

},

"id": "fetch\_task\_data",

"name": "Fetch Task Data",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"jsCode": "// Calculate quest rewards with difficulty multipliers\nconst taskData = $input.first().json;\nconst baseXP = taskData.xp || 20;\nconst baseCoins = taskData.coins || 10;\nconst difficulty = taskData.difficulty || 'medium';\n\n// Difficulty multipliers\nconst difficultyMultipliers = {\n 'easy': 1.0,\n 'tutorial': 1.0,\n 'medium': 1.5,\n 'hard': 2.0,\n 'epic': 3.0,\n 'legendary': 5.0\n};\n\nconst multiplier = difficultyMultipliers[difficulty] || 1.5;\n\n// Apply multiplier\nconst finalXP = Math.floor(baseXP \* multiplier);\nconst finalCoins = Math.floor(baseCoins \* multiplier);\n\n// Bonus for completing within deadline (if deadline exists)\nlet timeBonus = 0;\nif (taskData.deadline) {\n const deadline = new Date(taskData.deadline);\n const now = new Date();\n if (now <= deadline) {\n timeBonus = Math.floor(finalXP \* 0.2); // 20% bonus for on-time completion\n }\n}\n\nconst totalXP = finalXP + timeBonus;\n\nreturn [{\n json: {\n taskId: taskData.id,\n projectId: taskData.project\_id,\n characterId: taskData.character\_id,\n areaId: taskData.area\_id,\n xpEarned: totalXP,\n coinsEarned: finalCoins,\n timeBonus: timeBonus,\n difficulty: difficulty,\n multiplier: multiplier,\n taskTitle: taskData.title\n }\n}];"

},

"id": "calculate\_quest\_rewards",

"name": "Calculate Quest Rewards",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [650, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE tasks SET completed = true WHERE id = $1 RETURNING \*",

"options": {}

},

"id": "mark\_task\_complete",

"name": "Mark Task Complete",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [850, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT s.id, s.name FROM skills s JOIN areas a ON s.character\_id = a.character\_id WHERE a.id = $1 AND s.character\_id = $2 LIMIT 1",

"options": {}

},

"id": "find\_related\_skill",

"name": "Find Related Skill",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE skills SET xp = xp + $1, level = FLOOR(POWER(xp / 100, 0.5)) + 1 WHERE id = $2 RETURNING \*",

"options": {}

},

"id": "update\_skill\_xp\_quest",

"name": "Update Skill XP",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE characters SET xp = xp + $1, coins = coins + $2, level = FLOOR(POWER(xp / 100, 0.66)) + 1 WHERE id = $3 RETURNING \*",

"options": {}

},

"id": "update\_character\_quest",

"name": "Update Character",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT COUNT(\*) as completed\_count, COUNT(\*) FILTER (WHERE completed = false) as remaining\_count FROM tasks WHERE project\_id = $1",

"options": {}

},

"id": "check\_project\_completion",

"name": "Check Project Completion",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1650, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"conditions": {

"number": [

{

"value1": "={{ $json.remaining\_count }}",

"value2": 0

}

]

}

},

"id": "is\_project\_complete",

"name": "Project Complete?",

"type": "n8n-nodes-base.if",

"typeVersion": 2,

"position": [1850, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE projects SET completed = true WHERE id = $1 RETURNING \*",

"options": {}

},

"id": "complete\_project",

"name": "Complete Project",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2050, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO archive (character\_id, project\_id, completed\_on, xp\_earned, coins\_earned) VALUES ($1, $2, NOW(), $3, $4) RETURNING \*",

"options": {}

},

"id": "archive\_project",

"name": "Archive Project",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2250, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, xp\_change, coins\_change, description) VALUES ($1, 'task\_completed', $2, $3, $4) RETURNING \*",

"options": {}

},

"id": "log\_quest\_event",

"name": "Log Quest Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2050, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('user', $1, 'task', $2, 'task\_completed', $3, 'success', 'info', 'web') RETURNING \*",

"options": {}

},

"id": "log\_system\_quest",

"name": "Log System",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2250, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": true,\n \"xpEarned\": $('Calculate Quest Rewards').item.json.xpEarned,\n \"coinsEarned\": $('Calculate Quest Rewards').item.json.coinsEarned,\n \"timeBonus\": $('Calculate Quest Rewards').item.json.timeBonus,\n \"projectComplete\": $('Project Complete?').item.json.remaining\_count === 0,\n \"message\": \"Quest completed! You're making great progress!\"\n} }}",

"options": {}

},

"id": "respond\_quest\_success",

"name": "Respond Quest Success",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [2450, 300]

}

],

"pinData": {},

"connections": {

"Webhook - Task Complete": {

"main": [

[

{

"node": "Fetch Task Data",

"type": "main",

"index": 0

}

]

]

},

"Fetch Task Data": {

"main": [

[

{

"node": "Calculate Quest Rewards",

"type": "main",

"index": 0

}

]

]

},

"Calculate Quest Rewards": {

"main": [

[

{

"node": "Mark Task Complete",

"type": "main",

"index": 0

}

]

]

},

"Mark Task Complete": {

"main": [

[

{

"node": "Find Related Skill",

"type": "main",

"index": 0

}

]

]

},

"Find Related Skill": {

"main": [

[

{

"node": "Update Skill XP",

"type": "main",

"index": 0

}

]

]

},

"Update Skill XP": {

"main": [

[

{

"node": "Update Character",

"type": "main",

"index": 0

}

]

]

},

"Update Character": {

"main": [

[

{

"node": "Check Project Completion",

"type": "main",

"index": 0

}

]

]

},

"Check Project Completion": {

"main": [

[

{

"node": "Project Complete?",

"type": "main",

"index": 0

}

]

]

},

"Project Complete?": {

"main": [

[

{

"node": "Complete Project",

"type": "main",

"index": 0

}

],

[

{

"node": "Log Quest Event",

"type": "main",

"index": 0

}

]

]

},

"Complete Project": {

"main": [

[

{

"node": "Archive Project",

"type": "main",

"index": 0

}

]

]

},

"Archive Project": {

"main": [

[

{

"node": "Log Quest Event",

"type": "main",

"index": 0

}

]

]

},

"Log Quest Event": {

"main": [

[

{

"node": "Log System",

"type": "main",

"index": 0

}

]

]

},

"Log System": {

"main": [

[

{

"node": "Respond Quest Success",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"executionOrder": "v1"

},

"versionId": "1",

"meta": {

"instanceId": "life-game-production"

},

"id": "4",

"tags": []

}

{

"name": "SBS System Orchestrator",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "sbs-system-update",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook-system-update",

"name": "Webhook - System Update",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1,

"position": [250, 400],

"webhookId": "sbs-orchestrator-webhook"

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT s.\*, \n ss.step, \n ss.status as step\_status,\n ss.id as step\_id\nFROM systems s\nLEFT JOIN system\_steps ss ON s.id = ss.system\_id\nWHERE s.id = $1\nAND ss.status = 'pending'\nORDER BY \n CASE ss.step\n WHEN 'define' THEN 1\n WHEN 'design' THEN 2\n WHEN 'build' THEN 3\n WHEN 'automate' THEN 4\n WHEN 'review' THEN 5\n END\nLIMIT 1;",

"additionalFields": {

"queryParameters": "={{ [$json.body.id || $json.body.system\_id] }}"

}

},

"id": "get-current-step",

"name": "Get Current Pending Step",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [450, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.step }}",

"operation": "equal",

"value2": "design"

}

]

}

},

"id": "check-design-step",

"name": "Check Design Step",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [650, 300]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.step }}",

"operation": "equal",

"value2": "build"

}

]

}

},

"id": "check-build-step",

"name": "Check Build Step",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [650, 400]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.step }}",

"operation": "equal",

"value2": "automate"

}

]

}

},

"id": "check-automate-step",

"name": "Check Automate Step",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [650, 500]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.step }}",

"operation": "equal",

"value2": "review"

}

]

}

},

"id": "check-review-step",

"name": "Check Review Step",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [650, 600]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nVALUES ($1, 'design\_canvas\_generated', jsonb\_build\_object(\n 'name', $2,\n 'timestamp', NOW(),\n 'canvas\_template', 'markdown'\n))\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.id, $json.name] }}"

}

},

"id": "design-handler",

"name": "Design Handler - Generate Canvas",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [850, 200],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"method": "POST",

"url": "https://api.telegram.org/bot{{ $env.TELEGRAM\_BOT\_TOKEN }}/sendMessage",

"sendBody": true,

"bodyParameters": {

"parameters": [

{

"name": "chat\_id",

"value": "={{ $env.TELEGRAM\_CHAT\_ID }}"

},

{

"name": "text",

"value": "=🔧 \*Build Phase Started\*\n\nSystem: \*{{ $json.name }}\*\n\n📁 Creating folders and database schemas\n🔗 Setting up API integrations\n⚙️ Scaffolding automation structure"

},

{

"name": "parse\_mode",

"value": "Markdown"

}

]

},

"options": {}

},

"id": "build-handler",

"name": "Build Handler - Create Infrastructure",

"type": "n8n-nodes-base.httpRequest",

"typeVersion": 3,

"position": [850, 350],

"alwaysOutputData": true

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nVALUES ($1, 'automation\_configured', jsonb\_build\_object(\n 'triggers\_added', true,\n 'schedules\_created', true,\n 'timestamp', NOW()\n))\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.id] }}"

}

},

"id": "automate-handler",

"name": "Automate Handler - Setup Triggers",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [850, 500],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nVALUES ($1, 'review\_scheduled', jsonb\_build\_object(\n 'next\_review', NOW() + interval '30 days',\n 'review\_frequency', 'monthly',\n 'timestamp', NOW()\n))\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.id] }}"

}

},

"id": "review-handler",

"name": "Review Handler - Schedule Review",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [850, 650],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE system\_steps\nSET status = 'complete',\n updated\_at = NOW()\nWHERE id = $1\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.step\_id] }}"

}

},

"id": "mark-step-complete",

"name": "Mark Step Complete",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1050, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE systems\nSET current\_stage = (\n SELECT step FROM system\_steps\n WHERE system\_id = $1\n AND status = 'pending'\n ORDER BY \n CASE step\n WHEN 'define' THEN 1\n WHEN 'design' THEN 2\n WHEN 'build' THEN 3\n WHEN 'automate' THEN 4\n WHEN 'review' THEN 5\n END\n LIMIT 1\n)\nWHERE id = $1\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.id || $json.system\_id] }}"

}

},

"id": "update-system-stage",

"name": "Update System Stage",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1250, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": true,\n \"system\_id\": $json.id,\n \"step\_completed\": $json.step,\n \"next\_stage\": $json.current\_stage\n} }}",

"options": {}

},

"id": "webhook-response",

"name": "Webhook Response",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1,

"position": [1450, 400]

}

],

"connections": {

"Webhook - System Update": {

"main": [

[

{

"node": "Get Current Pending Step",

"type": "main",

"index": 0

}

]

]

},

"Get Current Pending Step": {

"main": [

[

{

"node": "Check Design Step",

"type": "main",

"index": 0

}

]

]

},

"Check Design Step": {

"main": [

[

{

"node": "Design Handler - Generate Canvas",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Build Step",

"type": "main",

"index": 0

}

]

]

},

"Check Build Step": {

"main": [

[

{

"node": "Build Handler - Create Infrastructure",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Automate Step",

"type": "main",

"index": 0

}

]

]

},

"Check Automate Step": {

"main": [

[

{

"node": "Automate Handler - Setup Triggers",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Review Step",

"type": "main",

"index": 0

}

]

]

},

"Check Review Step": {

"main": [

[

{

"node": "Review Handler - Schedule Review",

"type": "main",

"index": 0

}

]

]

},

"Design Handler - Generate Canvas": {

"main": [

[

{

"node": "Mark Step Complete",

"type": "main",

"index": 0

}

]

]

},

"Build Handler - Create Infrastructure": {

"main": [

[

{

"node": "Mark Step Complete",

"type": "main",

"index": 0

}

]

]

},

"Automate Handler - Setup Triggers": {

"main": [

[

{

"node": "Mark Step Complete",

"type": "main",

"index": 0

}

]

]

},

"Review Handler - Schedule Review": {

"main": [

[

{

"node": "Mark Step Complete",

"type": "main",

"index": 0

}

]

]

},

"Mark Step Complete": {

"main": [

[

{

"node": "Update System Stage",

"type": "main",

"index": 0

}

]

]

},

"Update System Stage": {

"main": [

[

{

"node": "Webhook Response",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"timezone": "America/Denver"

},

"tags": []

}

{

"name": "SBS PostgreSQL Event Listener",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "sbs-pg-notify",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook-pg-notify",

"name": "Webhook - PostgreSQL Notify",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1,

"position": [250, 400],

"webhookId": "sbs-pg-notify"

},

{

"parameters": {

"jsCode": "// Parse the PostgreSQL NOTIFY payload\nconst payload = $input.first().json.body;\nlet parsedData;\n\nif (typeof payload === 'string') {\n try {\n parsedData = JSON.parse(payload);\n } catch (e) {\n parsedData = { raw: payload };\n }\n} else {\n parsedData = payload;\n}\n\nreturn [{ json: parsedData }];"

},

"id": "parse-notify-payload",

"name": "Parse NOTIFY Payload",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [450, 400]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.current\_stage }}",

"operation": "equal",

"value2": "define"

}

]

}

},

"id": "check-define-stage",

"name": "Check Define Stage",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [650, 300]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.current\_stage }}",

"operation": "equal",

"value2": "complete"

}

]

}

},

"id": "check-complete-stage",

"name": "Check Complete Stage",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [650, 500]

},

{

"parameters": {

"method": "POST",

"url": "={{ $env.N8N\_WEBHOOK\_BASE\_URL }}/webhook/sbs-system-created",

"sendBody": true,

"bodyParameters": {

"parameters": [

{

"name": "system\_id",

"value": "={{ $json.id }}"

},

{

"name": "name",

"value": "={{ $json.name }}"

},

{

"name": "category",

"value": "={{ $json.category }}"

},

{

"name": "purpose",

"value": "={{ $json.purpose }}"

}

]

},

"options": {}

},

"id": "trigger-spawner",

"name": "Trigger System Spawner",

"type": "n8n-nodes-base.httpRequest",

"typeVersion": 3,

"position": [850, 250]

},

{

"parameters": {

"method": "POST",

"url": "={{ $env.N8N\_WEBHOOK\_BASE\_URL }}/webhook/sbs-system-update",

"sendBody": true,

"bodyParameters": {

"parameters": [

{

"name": "system\_id",

"value": "={{ $json.id }}"

},

{

"name": "current\_stage",

"value": "={{ $json.current\_stage }}"

},

{

"name": "name",

"value": "={{ $json.name }}"

}

]

},

"options": {}

},

"id": "trigger-orchestrator",

"name": "Trigger System Orchestrator",

"type": "n8n-nodes-base.httpRequest",

"typeVersion": 3,

"position": [850, 400]

},

{

"parameters": {

"chatId": "={{ $env.TELEGRAM\_CHAT\_ID }}",

"text": "=🎊 \*System Complete!\*\n\n\*{{ $json.name }}\* has completed all lifecycle stages.\n\n✅ Define → Design → Build → Automate → Review\n\nCongratulations on building a self-sustaining system! 🚀",

"additionalFields": {

"parseMode": "Markdown"

}

},

"id": "notify-complete",

"name": "Notify System Complete",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [850, 550],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nVALUES ($1, 'system\_completed', jsonb\_build\_object(\n 'completed\_at', NOW(),\n 'total\_duration\_days', EXTRACT(EPOCH FROM (NOW() - created\_at)) / 86400\n))\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.id] }}"

}

},

"id": "log-completion",

"name": "Log System Completion",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1050, 550],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": true,\n \"event\": \"system\_update\_processed\",\n \"system\_id\": $json.id,\n \"stage\": $json.current\_stage\n} }}",

"options": {}

},

"id": "webhook-response",

"name": "Webhook Response",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1,

"position": [1050, 400]

}

],

"connections": {

"Webhook - PostgreSQL Notify": {

"main": [

[

{

"node": "Parse NOTIFY Payload",

"type": "main",

"index": 0

}

]

]

},

"Parse NOTIFY Payload": {

"main": [

[

{

"node": "Check Define Stage",

"type": "main",

"index": 0

}

]

]

},

"Check Define Stage": {

"main": [

[

{

"node": "Trigger System Spawner",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Complete Stage",

"type": "main",

"index": 0

}

]

]

},

"Check Complete Stage": {

"main": [

[

{

"node": "Notify System Complete",

"type": "main",

"index": 0

}

],

[

{

"node": "Trigger System Orchestrator",

"type": "main",

"index": 0

}

]

]

},

"Trigger System Spawner": {

"main": [

[

{

"node": "Webhook Response",

"type": "main",

"index": 0

}

]

]

},

"Trigger System Orchestrator": {

"main": [

[

{

"node": "Webhook Response",

"type": "main",

"index": 0

}

]

]

},

"Notify System Complete": {

"main": [

[

{

"node": "Log System Completion",

"type": "main",

"index": 0

}

]

]

},

"Log System Completion": {

"main": [

[

{

"node": "Webhook Response",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"timezone": "America/Denver"

},

"tags": []

}

{

"name": "SBS Routine Engine",

"nodes": [

{

"parameters": {

"rule": {

"interval": [

{

"triggerAtHour": 9

}

]

}

},

"id": "schedule-trigger",

"name": "Schedule - Daily 9am",

"type": "n8n-nodes-base.scheduleTrigger",

"typeVersion": 1,

"position": [250, 400]

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT \n r.id as routine\_id,\n r.name as routine\_name,\n r.description,\n r.day\_of\_week,\n s.id as system\_id,\n s.name as system\_name,\n s.category,\n s.current\_stage\nFROM routines r\nJOIN systems s ON r.system\_id = s.id\nWHERE r.status = 'active'\nAND r.day\_of\_week = TO\_CHAR(CURRENT\_DATE, 'Day')\nAND s.current\_stage != 'complete';",

"additionalFields": {}

},

"id": "get-todays-routines",

"name": "Get Today's Active Routines",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [450, 400],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"conditions": {

"number": [

{

"value1": "={{ $json.length }}",

"operation": "larger",

"value2": 0

}

]

}

},

"id": "check-routines-exist",

"name": "Check If Routines Exist",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [650, 400]

},

{

"parameters": {},

"id": "split-routines",

"name": "Split Into Items",

"type": "n8n-nodes-base.splitInBatches",

"typeVersion": 2,

"position": [850, 300],

"parameters": {

"batchSize": 1,

"options": {}

}

},

{

"parameters": {

"chatId": "={{ $env.TELEGRAM\_CHAT\_ID }}",

"text": "=📅 \*Daily Routine Reminder\*\n\n\*System:\* {{ $json.system\_name }}\n\*Category:\* {{ $json.category }}\n\*Routine:\* {{ $json.routine\_name }}\n\n📝 {{ $json.description }}\n\n\*Current Stage:\* {{ $json.current\_stage }}\n\n✅ Reply with `/complete {{ $json.routine\_id }}` when done\n⏭️ Reply with `/skip {{ $json.routine\_id }}` to skip today",

"additionalFields": {

"parseMode": "Markdown"

}

},

"id": "send-routine-reminder",

"name": "Send Routine Reminder",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [1050, 300],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nVALUES ($1, 'routine\_reminder\_sent', jsonb\_build\_object(\n 'routine\_id', $2,\n 'routine\_name', $3,\n 'day\_of\_week', $4,\n 'timestamp', NOW()\n))\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [\n $json.system\_id,\n $json.routine\_id,\n $json.routine\_name,\n $json.day\_of\_week\n] }}"

}

},

"id": "log-reminder-sent",

"name": "Log Reminder Sent",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1250, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"mode": "combine",

"combineBy": "combineAll"

},

"id": "aggregate-results",

"name": "Aggregate Results",

"type": "n8n-nodes-base.aggregate",

"typeVersion": 1,

"position": [1450, 300]

},

{

"parameters": {

"chatId": "={{ $env.TELEGRAM\_CHAT\_ID }}",

"text": "=✨ \*Daily Routine Check Complete\*\n\n{{ $json.length }} reminder(s) sent today.\n\nHave a productive day! 🚀",

"additionalFields": {

"parseMode": "Markdown"

}

},

"id": "send-summary",

"name": "Send Summary Message",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [1650, 300],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"chatId": "={{ $env.TELEGRAM\_CHAT\_ID }}",

"text": "=📭 \*No Active Routines Today\*\n\nNo routines scheduled for {{ TO\_CHAR(CURRENT\_DATE, 'Day, Month DD, YYYY') }}.\n\nEnjoy your free day! ✨",

"additionalFields": {

"parseMode": "Markdown"

}

},

"id": "no-routines-message",

"name": "No Routines Message",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [850, 500],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT \n COUNT(\*) FILTER (WHERE current\_stage != 'complete') as active\_systems,\n COUNT(DISTINCT category) as categories,\n array\_agg(DISTINCT current\_stage) as stages\_in\_progress\nFROM systems;",

"additionalFields": {}

},

"id": "get-system-stats",

"name": "Get System Stats",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [250, 600],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"chatId": "={{ $env.TELEGRAM\_CHAT\_ID }}",

"text": "=📊 \*SBS System Summary\*\n\n🔧 Active Systems: {{ $json.active\_systems }}\n🗂️ Categories: {{ $json.categories }}\n⚙️ Stages in Progress: {{ $json.stages\_in\_progress.join(', ') }}\n\n\_Daily routine check initiated...\_",

"additionalFields": {

"parseMode": "Markdown"

}

},

"id": "send-daily-summary",

"name": "Send Daily Summary",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [450, 600],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

}

],

"connections": {

"Schedule - Daily 9am": {

"main": [

[

{

"node": "Get System Stats",

"type": "main",

"index": 0

},

{

"node": "Get Today's Active Routines",

"type": "main",

"index": 0

}

]

]

},

"Get System Stats": {

"main": [

[

{

"node": "Send Daily Summary",

"type": "main",

"index": 0

}

]

]

},

"Get Today's Active Routines": {

"main": [

[

{

"node": "Check If Routines Exist",

"type": "main",

"index": 0

}

]

]

},

"Check If Routines Exist": {

"main": [

[

{

"node": "Split Into Items",

"type": "main",

"index": 0

}

],

[

{

"node": "No Routines Message",

"type": "main",

"index": 0

}

]

]

},

"Split Into Items": {

"main": [

[

{

"node": "Send Routine Reminder",

"type": "main",

"index": 0

}

]

]

},

"Send Routine Reminder": {

"main": [

[

{

"node": "Log Reminder Sent",

"type": "main",

"index": 0

}

]

]

},

"Log Reminder Sent": {

"main": [

[

{

"node": "Aggregate Results",

"type": "main",

"index": 0

}

]

]

},

"Aggregate Results": {

"main": [

[

{

"node": "Send Summary Message",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"timezone": "America/Denver",

"executionOrder": "v1"

},

"tags": []

}

{

"name": "SBS System Spawner",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "sbs-system-created",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook-new-system",

"name": "Webhook - New System Created",

"type": "n8n-nodes-base.webhook",

"typeVersion": 1,

"position": [250, 300],

"webhookId": "sbs-system-webhook"

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_steps (system\_id, step, status, notes) VALUES\n($1, 'define', 'complete', 'System creation completed'),\n($1, 'design', 'pending', 'Design system architecture'),\n($1, 'build', 'pending', 'Build working components'),\n($1, 'automate', 'pending', 'Add triggers and schedules'),\n($1, 'review', 'pending', 'Schedule review cycle')\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [\n $json.body.system\_id\n] }}"

}

},

"id": "insert-lifecycle-steps",

"name": "Insert Lifecycle Steps",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO routines (name, system\_id, day\_of\_week, description, status)\nSELECT \n $1 || ' - ' || day,\n $2,\n day,\n 'Auto-generated routine for ' || $1,\n 'active'\nFROM unnest(ARRAY['Monday', 'Friday']) AS day\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [\n $json.body.name,\n $json.body.system\_id\n] }}"

}

},

"id": "create-default-routines",

"name": "Create Default Routines",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [650, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE systems \nSET current\_stage = 'design',\n metadata = jsonb\_set(\n COALESCE(metadata, '{}'::jsonb),\n '{spawned\_at}',\n to\_jsonb(NOW())\n )\nWHERE id = $1\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.body.system\_id] }}"

}

},

"id": "advance-to-design",

"name": "Advance Stage to Design",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [850, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nVALUES ($1, 'system\_spawned', jsonb\_build\_object(\n 'name', $2,\n 'category', $3,\n 'timestamp', NOW()\n))\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [\n $json.body.system\_id,\n $json.body.name,\n $json.body.category\n] }}"

}

},

"id": "log-spawn-event",

"name": "Log System Spawn Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [1050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"chatId": "={{ $env.TELEGRAM\_CHAT\_ID }}",

"text": "=🎉 \*New System Created\*\n\n📋 \*Name:\* {{ $json.body.name }}\n🗂️ \*Category:\* {{ $json.body.category }}\n🎯 \*Purpose:\* {{ $json.body.purpose }}\n\n✅ Lifecycle steps initialized\n📅 Default routines created\n🚀 Stage advanced to: \*Design\*\n\n\*Next Step:\* Define your system architecture and data flows.",

"additionalFields": {

"parseMode": "Markdown"

}

},

"id": "send-telegram-notification",

"name": "Send Telegram Notification",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [1250, 300],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": true,\n \"system\_id\": $json.body.system\_id,\n \"message\": \"System spawned successfully\",\n \"next\_stage\": \"design\"\n} }}",

"options": {

"responseCode": 201

}

},

"id": "webhook-response",

"name": "Webhook Response",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1,

"position": [1450, 300]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.body.system\_id }}",

"operation": "isNotEmpty"

}

]

}

},

"id": "validate-input",

"name": "Validate System ID",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [450, 150]

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n \"success\": false,\n \"error\": \"Missing required field: system\_id\"\n} }}",

"options": {

"responseCode": 400

}

},

"id": "error-response",

"name": "Error Response",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1,

"position": [450, 50]

}

],

"connections": {

"Webhook - New System Created": {

"main": [

[

{

"node": "Validate System ID",

"type": "main",

"index": 0

}

]

]

},

"Validate System ID": {

"main": [

[

{

"node": "Insert Lifecycle Steps",

"type": "main",

"index": 0

}

],

[

{

"node": "Error Response",

"type": "main",

"index": 0

}

]

]

},

"Insert Lifecycle Steps": {

"main": [

[

{

"node": "Create Default Routines",

"type": "main",

"index": 0

}

]

]

},

"Create Default Routines": {

"main": [

[

{

"node": "Advance Stage to Design",

"type": "main",

"index": 0

}

]

]

},

"Advance Stage to Design": {

"main": [

[

{

"node": "Log System Spawn Event",

"type": "main",

"index": 0

}

]

]

},

"Log System Spawn Event": {

"main": [

[

{

"node": "Send Telegram Notification",

"type": "main",

"index": 0

}

]

]

},

"Send Telegram Notification": {

"main": [

[

{

"node": "Webhook Response",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"timezone": "America/Denver"

},

"tags": []

}

{

"name": "SBS Telegram Bot",

"nodes": [

{

"parameters": {

"updates": [

"message"

]

},

"id": "telegram-trigger",

"name": "Telegram Trigger",

"type": "n8n-nodes-base.telegramTrigger",

"typeVersion": 1,

"position": [250, 400],

"webhookId": "sbs-telegram-bot",

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.message.text }}",

"operation": "startsWith",

"value2": "/complete"

}

]

}

},

"id": "check-complete",

"name": "Check Complete",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [450, 200]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.message.text }}",

"operation": "startsWith",

"value2": "/skip"

}

]

}

},

"id": "check-skip",

"name": "Check Skip",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [450, 300]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.message.text }}",

"operation": "startsWith",

"value2": "/advance"

}

]

}

},

"id": "check-advance",

"name": "Check Advance",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [450, 450]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.message.text }}",

"operation": "startsWith",

"value2": "/status"

}

]

}

},

"id": "check-status",

"name": "Check Status",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [450, 600]

},

{

"parameters": {

"conditions": {

"string": [

{

"value1": "={{ $json.message.text }}",

"operation": "startsWith",

"value2": "/help"

}

]

}

},

"id": "check-help",

"name": "Check Help",

"type": "n8n-nodes-base.if",

"typeVersion": 1,

"position": [450, 750]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nSELECT \n system\_id,\n 'routine\_completed',\n jsonb\_build\_object(\n 'routine\_id', $1,\n 'completed\_at', NOW(),\n 'completed\_by', 'telegram\_bot'\n )\nFROM routines\nWHERE id = $1\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.message.text.split(' ')[1]] }}"

}

},

"id": "complete-routine",

"name": "Complete Routine",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [650, 150],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"chatId": "={{ $json.message.chat.id }}",

"text": "=✅ \*Routine Completed!\*\n\nGreat work! Your routine has been marked as complete.\n\n🎯 Keep up the momentum!",

"additionalFields": {

"parseMode": "Markdown",

"replyToMessageId": "={{ $json.message.message\_id }}"

}

},

"id": "complete-response",

"name": "Send Complete Response",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [850, 150],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO system\_logs (system\_id, event, details)\nSELECT \n system\_id,\n 'routine\_skipped',\n jsonb\_build\_object(\n 'routine\_id', $1,\n 'skipped\_at', NOW(),\n 'reason', 'user\_request'\n )\nFROM routines\nWHERE id = $1\nRETURNING \*;",

"additionalFields": {

"queryParameters": "={{ [$json.message.text.split(' ')[1]] }}"

}

},

"id": "skip-routine",

"name": "Skip Routine",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [650, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"chatId": "={{ $json.message.chat.id }}",

"text": "=⏭️ \*Routine Skipped\*\n\nNo problem! This routine has been skipped for today.\n\n💡 You'll get a reminder next time it's scheduled.",

"additionalFields": {

"parseMode": "Markdown",

"replyToMessageId": "={{ $json.message.message\_id }}"

}

},

"id": "skip-response",

"name": "Send Skip Response",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [850, 300],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"method": "POST",

"url": "={{ $env.N8N\_WEBHOOK\_BASE\_URL }}/webhook/sbs-system-update",

"sendBody": true,

"bodyParameters": {

"parameters": [

{

"name": "system\_id",

"value": "={{ $json.message.text.split(' ')[1] }}"

},

{

"name": "action",

"value": "advance\_stage"

}

]

},

"options": {}

},

"id": "advance-system",

"name": "Advance System Stage",

"type": "n8n-nodes-base.httpRequest",

"typeVersion": 3,

"position": [650, 450]

},

{

"parameters": {

"chatId": "={{ $json.message.chat.id }}",

"text": "=🚀 \*System Stage Advanced!\*\n\nYour system has been moved to the next stage in its lifecycle.\n\nCheck the orchestrator for next steps!",

"additionalFields": {

"parseMode": "Markdown",

"replyToMessageId": "={{ $json.message.message\_id }}"

}

},

"id": "advance-response",

"name": "Send Advance Response",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [850, 450],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT \n s.id,\n s.name,\n s.category,\n s.current\_stage,\n COUNT(DISTINCT ss.id) FILTER (WHERE ss.status = 'complete') as completed\_steps,\n COUNT(DISTINCT ss.id) as total\_steps,\n COUNT(DISTINCT r.id) as active\_routines,\n MAX(sl.created\_at) as last\_activity\nFROM systems s\nLEFT JOIN system\_steps ss ON s.id = ss.system\_id\nLEFT JOIN routines r ON s.id = r.system\_id AND r.status = 'active'\nLEFT JOIN system\_logs sl ON s.id = sl.system\_id\nWHERE s.current\_stage != 'complete'\nGROUP BY s.id, s.name, s.category, s.current\_stage\nORDER BY last\_activity DESC\nLIMIT 10;",

"additionalFields": {}

},

"id": "get-system-status",

"name": "Get System Status",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2,

"position": [650, 600],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL SBS"

}

}

},

{

"parameters": {

"jsCode": "const systems = $input.all();\nlet message = '📊 \*SBS System Status\*\\n\\n';\n\nif (systems.length === 0) {\n message += 'No active systems found.\\n\\nCreate your first system to get started!';\n} else {\n systems.forEach((item, index) => {\n const s = item.json;\n const progress = Math.round((s.completed\_steps / s.total\_steps) \* 100);\n message += `${index + 1}. \*${s.name}\*\\n`;\n message += ` 📁 ${s.category}\\n`;\n message += ` ⚙️ Stage: ${s.current\_stage}\\n`;\n message += ` 📈 Progress: ${progress}% (${s.completed\_steps}/${s.total\_steps})\\n`;\n message += ` 🔄 Active Routines: ${s.active\_routines}\\n`;\n message += `\\n`;\n });\n}\n\nreturn [{ json: { message } }];"

},

"id": "format-status-message",

"name": "Format Status Message",

"type": "n8n-nodes-base.code",

"typeVersion": 2,

"position": [850, 600]

},

{

"parameters": {

"chatId": "={{ $('Telegram Trigger').item.json.message.chat.id }}",

"text": "={{ $json.message }}",

"additionalFields": {

"parseMode": "Markdown",

"replyToMessageId": "={{ $('Telegram Trigger').item.json.message.message\_id }}"

}

},

"id": "status-response",

"name": "Send Status Response",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [1050, 600],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"chatId": "={{ $json.message.chat.id }}",

"text": "=🤖 \*SBS Telegram Bot Commands\*\n\n\*Routine Management:\*\n`/complete [routine\_id]` - Mark routine as complete\n`/skip [routine\_id]` - Skip today's routine\n\n\*System Management:\*\n`/advance [system\_id]` - Move system to next stage\n`/status` - View all active systems\n\n\*Information:\*\n`/help` - Show this help message\n\n\*Quick Tip:\* Use the routine\_id from daily reminders to quickly complete or skip tasks!",

"additionalFields": {

"parseMode": "Markdown",

"replyToMessageId": "={{ $json.message.message\_id }}"

}

},

"id": "help-response",

"name": "Send Help Response",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [650, 750],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

},

{

"parameters": {

"chatId": "={{ $json.message.chat.id }}",

"text": "=❓ \*Unknown Command\*\n\nI didn't recognize that command.\n\nUse `/help` to see available commands.",

"additionalFields": {

"parseMode": "Markdown",

"replyToMessageId": "={{ $json.message.message\_id }}"

}

},

"id": "default-response",

"name": "Send Default Response",

"type": "n8n-nodes-base.telegram",

"typeVersion": 1,

"position": [650, 900],

"credentials": {

"telegramApi": {

"id": "2",

"name": "Telegram Bot"

}

}

}

],

"connections": {

"Telegram Trigger": {

"main": [

[

{

"node": "Check Complete",

"type": "main",

"index": 0

}

]

]

},

"Check Complete": {

"main": [

[

{

"node": "Complete Routine",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Skip",

"type": "main",

"index": 0

}

]

]

},

"Check Skip": {

"main": [

[

{

"node": "Skip Routine",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Advance",

"type": "main",

"index": 0

}

]

]

},

"Check Advance": {

"main": [

[

{

"node": "Advance System Stage",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Status",

"type": "main",

"index": 0

}

]

]

},

"Check Status": {

"main": [

[

{

"node": "Get System Status",

"type": "main",

"index": 0

}

],

[

{

"node": "Check Help",

"type": "main",

"index": 0

}

]

]

},

"Check Help": {

"main": [

[

{

"node": "Send Help Response",

"type": "main",

"index": 0

}

],

[

{

"node": "Send Default Response",

"type": "main",

"index": 0

}

]

]

},

"Complete Routine": {

"main": [

[

{

"node": "Send Complete Response",

"type": "main",

"index": 0

}

]

]

},

"Skip Routine": {

"main": [

[

{

"node": "Send Skip Response",

"type": "main",

"index": 0

}

]

]

},

"Advance System Stage": {

"main": [

[

{

"node": "Send Advance Response",

"type": "main",

"index": 0

}

]

]

},

"Get System Status": {

"main": [

[

{

"node": "Format Status Message",

"type": "main",

"index": 0

}

]

]

},

"Format Status Message": {

"main": [

[

{

"node": "Send Status Response",

"type": "main",

"index": 0

}

]

]

}

},

"active": true,

"settings": {

"timezone": "America/Denver"

},

"tags": []

}

{

"name": "SHOP\_CHECK",

"nodes": [

{

"parameters": {

"httpMethod": "POST",

"path": "shop/purchase",

"responseMode": "responseNode",

"options": {}

},

"id": "webhook\_purchase",

"name": "Webhook - Purchase Request",

"type": "n8n-nodes-base.webhook",

"typeVersion": 2,

"position": [250, 300],

"webhookId": "shop-purchase"

},

{

"parameters": {

"assignments": {

"assignments": [

{

"id": "character\_id",

"name": "characterId",

"value": "={{ $json.body.character\_id }}",

"type": "number"

},

{

"id": "item\_id",

"name": "itemId",

"value": "={{ $json.body.item\_id }}",

"type": "number"

},

{

"id": "quantity",

"name": "quantity",

"value": "={{ $json.body.quantity || 1 }}",

"type": "number"

}

]

}

},

"id": "parse\_request",

"name": "Parse Request",

"type": "n8n-nodes-base.set",

"typeVersion": 3.3,

"position": [450, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT i.id, i.name, i.cost, i.item\_type, i.rarity, i.description, i.effect FROM items i WHERE i.id = $1",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $json.itemId }}"

}

]

}

}

},

"id": "fetch\_item",

"name": "Fetch Item Details",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "SELECT c.id, c.user\_id, c.coins, u.username FROM characters c JOIN users u ON c.user\_id = u.id WHERE c.id = $1",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Parse Request').item.json.characterId }}"

}

]

}

}

},

"id": "fetch\_character",

"name": "Fetch Character",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [650, 450],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"assignments": {

"assignments": [

{

"id": "total\_cost",

"name": "totalCost",

"value": "={{ $('Fetch Item Details').item.json.cost \* $('Parse Request').item.json.quantity }}",

"type": "number"

},

{

"id": "user\_coins",

"name": "userCoins",

"value": "={{ $('Fetch Character').item.json.coins }}",

"type": "number"

},

{

"id": "can\_afford",

"name": "canAfford",

"value": "={{ $('Fetch Character').item.json.coins >= ($('Fetch Item Details').item.json.cost \* $('Parse Request').item.json.quantity) }}",

"type": "boolean"

},

{

"id": "item\_data",

"name": "itemData",

"value": "={{ $('Fetch Item Details').item.json }}",

"type": "object"

},

{

"id": "character\_data",

"name": "characterData",

"value": "={{ $('Fetch Character').item.json }}",

"type": "object"

}

]

}

},

"id": "validate\_purchase",

"name": "Validate Purchase",

"type": "n8n-nodes-base.set",

"typeVersion": 3.3,

"position": [850, 375]

},

{

"parameters": {

"conditions": {

"options": {

"caseSensitive": true,

"leftValue": "",

"typeValidation": "strict"

},

"conditions": [

{

"id": "can\_afford\_check",

"leftValue": "={{ $json.canAfford }}",

"rightValue": true,

"operator": {

"type": "boolean",

"operation": "equals"

}

}

],

"combinator": "and"

},

"options": {}

},

"id": "if\_can\_afford",

"name": "Can Afford?",

"type": "n8n-nodes-base.if",

"typeVersion": 2,

"position": [1050, 375]

},

{

"parameters": {

"operation": "executeQuery",

"query": "UPDATE characters SET coins = coins - $1 WHERE id = $2 RETURNING coins",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Validate Purchase').item.json.totalCost }}"

},

{

"parameter": "={{ $('Parse Request').item.json.characterId }}"

}

]

}

}

},

"id": "deduct\_coins",

"name": "Deduct Coins",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO inventory (character\_id, item\_id, quantity) VALUES ($1, $2, $3) ON CONFLICT (character\_id, item\_id) DO UPDATE SET quantity = inventory.quantity + $3 RETURNING \*",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Parse Request').item.json.characterId }}"

},

{

"parameter": "={{ $('Parse Request').item.json.itemId }}"

},

{

"parameter": "={{ $('Parse Request').item.json.quantity }}"

}

]

}

}

},

"id": "add\_to\_inventory",

"name": "Add to Inventory",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1450, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO transactions (character\_id, type, amount, item\_id, description) VALUES ($1, 'spend', $2, $3, $4) RETURNING \*",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Parse Request').item.json.characterId }}"

},

{

"parameter": "={{ $('Validate Purchase').item.json.totalCost }}"

},

{

"parameter": "={{ $('Parse Request').item.json.itemId }}"

},

{

"parameter": "=Purchased {{ $('Parse Request').item.json.quantity }}x {{ $('Validate Purchase').item.json.itemData.name }}"

}

]

}

}

},

"id": "log\_transaction",

"name": "Log Transaction",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1650, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO events (character\_id, event\_type, coins\_change, description) VALUES ($1, 'shop\_purchase', $2, $3)",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Parse Request').item.json.characterId }}"

},

{

"parameter": "={{ -$('Validate Purchase').item.json.totalCost }}"

},

{

"parameter": "=Purchased {{ $('Validate Purchase').item.json.itemData.name }} (x{{ $('Parse Request').item.json.quantity }}) for {{ $('Validate Purchase').item.json.totalCost }} coins"

}

]

}

}

},

"id": "log\_event",

"name": "Log Event",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1850, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('user', $1, 'item', $2, 'shop\_purchase', $3, 'success', 'info', 'api')",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Validate Purchase').item.json.characterData.user\_id }}"

},

{

"parameter": "={{ $('Parse Request').item.json.itemId }}"

},

{

"parameter": "={{ JSON.stringify({itemName: $('Validate Purchase').item.json.itemData.name, quantity: $('Parse Request').item.json.quantity, cost: $('Validate Purchase').item.json.totalCost, remainingCoins: $('Deduct Coins').item.json.coins}) }}"

}

]

}

}

},

"id": "log\_system",

"name": "Log System",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [2050, 300],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n success: true,\n message: 'Purchase successful!',\n item: $('Validate Purchase').item.json.itemData,\n quantity: $('Parse Request').item.json.quantity,\n totalCost: $('Validate Purchase').item.json.totalCost,\n remainingCoins: $('Deduct Coins').item.json.coins,\n inventory: $('Add to Inventory').item.json\n} }}",

"options": {

"responseCode": 200

}

},

"id": "success\_response",

"name": "Success Response",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [2250, 300]

},

{

"parameters": {

"operation": "executeQuery",

"query": "INSERT INTO systems\_log (actor\_type, actor\_id, target\_type, target\_id, action, detail, outcome, severity, source) VALUES ('user', $1, 'item', $2, 'shop\_purchase\_failed', $3, 'failure', 'warning', 'api')",

"options": {

"queryParameters": {

"parameters": [

{

"parameter": "={{ $('Validate Purchase').item.json.characterData.user\_id }}"

},

{

"parameter": "={{ $('Parse Request').item.json.itemId }}"

},

{

"parameter": "={{ JSON.stringify({reason: 'insufficient\_coins', required: $('Validate Purchase').item.json.totalCost, available: $('Validate Purchase').item.json.userCoins}) }}"

}

]

}

}

},

"id": "log\_failure",

"name": "Log Failure",

"type": "n8n-nodes-base.postgres",

"typeVersion": 2.4,

"position": [1250, 450],

"credentials": {

"postgres": {

"id": "1",

"name": "PostgreSQL account"

}

}

},

{

"parameters": {

"respondWith": "json",

"responseBody": "={{ {\n success: false,\n error: 'Insufficient coins',\n required: $('Validate Purchase').item.json.totalCost,\n available: $('Validate Purchase').item.json.userCoins,\n shortfall: $('Validate Purchase').item.json.totalCost - $('Validate Purchase').item.json.userCoins\n} }}",

"options": {

"responseCode": 400

}

},

"id": "error\_response",

"name": "Error Response",

"type": "n8n-nodes-base.respondToWebhook",

"typeVersion": 1.1,

"position": [1450, 450]

}

],

"connections": {

"Webhook - Purchase Request": {

"main": [

[

{

"node": "Parse Request",

"type": "main",

"index": 0

}

]

]

},

"Parse Request": {

"main": [

[

{

"node": "Fetch Item Details",

"type": "main",

"index": 0

},

{

"node": "Fetch Character",

"type": "main",

"index": 0

}

]

]

},

"Fetch Item Details": {

"main": [

[

{

"node": "Validate Purchase",

"type": "main",

"index": 0

}

]

]

},

"Fetch Character": {

"main": [

[

{

"node": "Validate Purchase",

"type": "main",

"index": 0

}

]

]

},

"Validate Purchase": {

"main": [

[

{

"node": "Can Afford?",

"type": "main",

"index": 0

}

]

]

},

"Can Afford?": {

"main": [

[

{

"node": "Deduct Coins",

"type": "main",

"index": 0

}

],

[

{

"node": "Log Failure",

"type": "main",

"index": 0

}

]

]

},

"Deduct Coins": {

"main": [

[

{

"node": "Add to Inventory",

"type": "main",

"index": 0

}

]

]

},

"Add to Inventory": {

"main": [

[

{

"node": "Log Transaction",

"type": "main",

"index": 0

}

]

]

},

"Log Transaction": {

"main": [

[

{

"node": "Log Event",

"type": "main",

"index": 0

}

]

]

},

"Log Event": {

"main": [

[

{

"node": "Log System",

"type": "main",

"index": 0

}

]

]

},

"Log System": {

"main": [

[

{

"node": "Success Response",

"type": "main",

"index": 0

}

]

]

},

"Log Failure": {

"main": [

[

{

"node": "Error Response",

"type": "main",

"index": 0

}

]

]

}

},

"settings": {

"executionOrder": "v1"

}

}