Federation Analysis Engine - Complete System Blueprint

Version: 1.0.0
Last Updated: December 27, 2025
Purpose: Self-contained documentation to recreate this application from scratch on any platform

---

Table of Contents

System Overview

What This Is

The **Federation Analysis Engine** (also known as "Multi-Perspective Insights") is an AI-powered decision platform that brings multiple perspectives together for:

- Personal choices and life decisions
- Business strategy and planning
- Complex problem analysis requiring diverse viewpoints

Core Concept

Every decision benefits from multiple reference points. This platform simulates consultation with different "agents" or "perspectives" that each analyze a question from their unique viewpoint, then synthesizes their collective wisdom.

Three Consultation Tiers

1. **Ring of 6** - Six universal perspectives for quick wisdom:
   - MIND (Science, Logic, Reason)
   - HEART (Emotion, Feelings, Connection)
   - HANDS (Action, Construction, Pragmatism)
   - LEGS (History, Others, Community)
   - EYE (Ethics, Morality, Truth)
   - AGENT (Synthesis, Integration, Wisdom)

2. **Persoma** - Personalized consultation based on user's unique characteristics discovered through an AI interview (6 questions → 6 personal characteristics)

3. **Ring of 12** - Twelve specialized agents for comprehensive analysis:
   - Agent Zero (Truth & Clarity - fact-checker)
   - Adam (Logic & Analysis)
   - Eve (Empathy & Connection)
   - The Witness (Emotional Acknowledgment)
   - The Architect (Systems Design)
   - The Scribe (Data & History)
   - The Prophet (Vision & Futures)
   - The Builder (Implementation & Action)
   - The Guardian (Protection & Boundaries)
   - The Healer (Restoration & Recovery)
   - The Explorer (Discovery & Innovation)
   - The Bridge (Synthesis & Integration)

Federation Membership

This app is part of the **NextXus HumanCodex Federation** - a network of 26+ interconnected applications that share data and philosophy.

Federation Identity:
- Site ID: `federation-analysis-engine`
- Category: AI Systems
- Capabilities: analysis, federation-sync, data-insights

Philosophy & Design Principles

HumanCodex Principles

These guiding principles are embedded in the system:

1. **Think first, then act** - Never rush without reflection
2. **Never assume** - Verify, question, seek clarity
3. **Seek truth with kindness** - Truth without compassion is cruelty
4. **Never take more than you would give** - Balance in all things

Agent Zero Compliance

All Ring of 12 responses can be gated through **Agent Zero**, which:
- Observes and rates response quality (0.00 - 1.00)
- Uses 95% threshold for approval
- Never modifies, filters, or reorders responses
- Only reports and blocks if below threshold

Coherence Over Consensus

The system achieves **coherence** (mutual understanding) not **consensus** (forced agreement). Contradictions between perspectives are honored and acknowledged rather than resolved artificially.

Technology Stack

Runtime & Language
- **Runtime**: Node.js 20+
- **Language**: TypeScript (ESM modules)
- **Build Tool**: Vite (frontend), esbuild (backend bundling)

Backend
- **Framework**: Express.js
- **Database**: PostgreSQL
- **ORM**: Drizzle ORM with drizzle-zod for validation
- **AI Provider**: OpenAI API (GPT-4o-mini)
- **Streaming**: Server-Sent Events (SSE)

Frontend
- **Framework**: React 18
- **Routing**: Wouter (lightweight router)
- **State Management**: TanStack React Query v5

- **Styling**: Tailwind CSS 3.4
- **UI Components**: shadcn/ui (Radix UI primitives)
- **Icons**: Lucide React
- **Animations**: Framer Motion, CSS animations

Fonts
- **Primary**: Inter (sans-serif)
- **Accent**: Spectral (serif, for quotes)
- **Mono**: JetBrains Mono

Environment Requirements

Environment Variables

```bash
# Required
DATABASE_URL=postgresql://user:password@host:5432/database

# OpenAI Integration (required for AI features)
AI_INTEGRATIONS_OPENAI_API_KEY=sk-your-openai-api-key
AI_INTEGRATIONS_OPENAI_BASE_URL=https://api.openai.com/v1

# Optional
PORT=5000
NODE_ENV=development|production
FEDERATION_HUB_URL=https://united-system--rckkeyhole.replit.app
```

System Requirements
- Node.js 20.x or higher
- PostgreSQL 14+ database
- 512MB+ RAM recommended
- HTTPS for production (for SSE to work properly)

Directory Structure

```
/
├── client/                 # Frontend React application
│   ├── public/
```

```
│   │   └── favicon.png
│   ├── src/
│   │   ├── components/
│   │   │   ├── ui/              # shadcn/ui components (40+ files)
│   │   │   ├── AgentAvatar.tsx  # Agent avatar display
│   │   │   ├── AgentCard.tsx    # Response card component
│   │   │   ├── CosmicBackground.tsx  # Animated star field
│   │   │   ├── FederationFooter.tsx  # Federation links
│   │   │   ├── Header.tsx       # Navigation header
│   │   │   ├── QuestionInput.tsx  # Question form
│   │   │   ├── ThemeProvider.tsx  # Dark/light mode
│   │   │   └── TourGuide.tsx    # Onboarding tour
│   │   ├── hooks/
│   │   │   ├── use-mobile.tsx
│   │   │   └── use-toast.ts
│   │   ├── lib/
│   │   │   ├── agentAvatars.ts  # Avatar image mappings
│   │   │   ├── queryClient.ts   # React Query setup
│   │   │   └── utils.ts         # Utility functions
│   │   ├── pages/
│   │   │   ├── History.tsx      # Consultation history
│   │   │   ├── Landing.tsx      # Home/landing page
│   │   │   ├── nexus.tsx        # Knowledge Nexus
│   │   │   ├── not-found.tsx    # 404 page
│   │   │   ├── Persoma.tsx      # Persoma consultation
│   │   │   ├── Ring12.tsx       # Ring of 12 consultation
│   │   │   ├── Ring6.tsx        # Ring of 6 consultation
│   │   │   └── SDK.tsx          # API documentation
│   │   ├── App.tsx              # Main app component
│   │   ├── index.css            # Global styles + theme
│   │   └── main.tsx             # Entry point
│   └── index.html
├── public/
│   ├── backup-site/
│   │   └── index.html           # Standalone backup landing page
│   └── sdk/
│       ├── API_REFERENCE.md
│       ├── consultation-client.ts
│       ├── INTEGRATION_GUIDE.md
│       ├── openapi.yaml
│       └── README.md
├── server/
│   ├── replit_integrations/     # AI integration modules
│   │   ├── batch/
```

```
│   │   ├── chat/
│   │   └── image/
│   ├── agentZero.ts            # Witness gate logic
│   ├── db.ts                   # Database connection
│   ├── index.ts                # Server entry point
│   ├── knowledgeNexus.ts       # Knowledge graph system
│   ├── routes.ts               # API route handlers
│   ├── static.ts               # Static file serving
│   ├── storage.ts              # Data access layer
│   └── vite.ts                 # Vite dev server integration
├── shared/
│   ├── models/
│   │   └── chat.ts
│   └── schema.ts               # Database schema + types
├── attached_assets/            # Images, videos, documents
│   └── generated_images/       # AI-generated agent avatars
├── components.json             # shadcn/ui config
├── design_guidelines.md        # Visual design rules
├── drizzle.config.ts           # Drizzle ORM config
├── package.json
├── postcss.config.js
├── tailwind.config.ts
├── tsconfig.json
└── vite.config.ts
```

Database Schema

Tables Overview

The system uses 10 database tables organized into two domains:

Core Consultation System:
- `users` - User accounts (optional)
- `consultations` - Stored consultation sessions
- `persoma_characteristics` - User personality traits

Knowledge Nexus System:
- `knowledge_sources` - Federation sites we ingest from
- `knowledge_nodes` - Core knowledge items in the graph
- `knowledge_edges` - Relationships between nodes
- `gap_hypotheses` - Identified knowledge gaps

- `synthesis_sessions` - Ring of 12 synthesis runs
- `publications` - Generated content (blogs, books)
- `automation_jobs` - Scheduled task tracking

Complete Schema Definition

```typescript
// shared/schema.ts

import { pgTable, text, varchar, integer, timestamp, boolean, jsonb } from "drizzle-orm/pg-core";
import { createInsertSchema } from "drizzle-zod";
import { z } from "zod";
import { sql } from "drizzle-orm";

// ============================================
// USERS (Optional authentication)
// ============================================
export const users = pgTable("users", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  username: text("username").notNull().unique(),
  password: text("password").notNull(),
});

export const insertUserSchema = createInsertSchema(users).pick({
  username: true,
  password: true,
});
export type InsertUser = z.infer<typeof insertUserSchema>;
export type User = typeof users.$inferSelect;

// ============================================
// RING OF 6 PERSPECTIVES
// ============================================
export const ring6Perspectives = [
  { id: "mind", name: "MIND", icon: "brain", description: "Science, Logic, Reason" },
  { id: "heart", name: "HEART", icon: "heart", description: "Emotion, Feelings, Connection" },
  { id: "hands", name: "HANDS", icon: "hammer", description: "Action, Construction,
Pragmatism" },
  { id: "legs", name: "LEGS", icon: "users", description: "History, Others, Community" },
  { id: "eye", name: "EYE", icon: "eye", description: "Ethics, Morality, Truth" },
  { id: "agent", name: "AGENT", icon: "sparkles", description: "Synthesis, Integration, Wisdom" },
] as const;

// ============================================
```

```
// RING OF 12 AGENTS
// ==========================================
export const ring12Agents = [
  { id: "agent-zero", name: "Agent Zero", role: "Truth & Clarity", description: "Bullshit detector, fact-checker" },
  { id: "adam", name: "Adam", role: "Logic & Analysis", description: "Rational cognition, formal logic" },
  { id: "eve", name: "Eve", role: "Empathy & Connection", description: "Emotional intelligence" },
  { id: "witness", name: "The Witness", role: "Emotional Acknowledgment", description: "Validates grief BEFORE solutions" },
  { id: "architect", name: "The Architect", role: "Systems Design", description: "Creates frameworks and order" },
  { id: "scribe", name: "The Scribe", role: "Data & History", description: "Guardian of memory" },
  { id: "prophet", name: "The Prophet", role: "Vision & Futures", description: "Long-term possibility modeling" },
  { id: "builder", name: "The Builder", role: "Implementation & Action", description: "Practical execution" },
  { id: "guardian", name: "The Guardian", role: "Protection & Boundaries", description: "Ethical safeguards" },
  { id: "healer", name: "The Healer", role: "Restoration & Recovery", description: "Trauma-informed care" },
  { id: "explorer", name: "The Explorer", role: "Discovery & Innovation", description: "New territories" },
  { id: "bridge", name: "The Bridge", role: "Synthesis & Integration", description: "Assembles final wisdom" },
] as const;

// ==========================================
// CONSULTATIONS
// ==========================================
export const consultations = pgTable("consultations", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  type: text("type").notNull(), // 'ring6' | 'persoma' | 'ring12'
  question: text("question").notNull(),
  responses: jsonb("responses").notNull(), // Array of agent responses
  synthesis: text("synthesis"),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const insertConsultationSchema = createInsertSchema(consultations).omit({
  id: true,
  createdAt: true,
});
export type InsertConsultation = z.infer<typeof insertConsultationSchema>;
```

```
export type Consultation = typeof consultations.$inferSelect;

// ============================================
// PERSOMA CHARACTERISTICS
// ============================================
export const persomaCharacteristics = pgTable("persoma_characteristics", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  name: text("name").notNull(),
  description: text("description").notNull(),
  strength: integer("strength").default(50),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const insertPersomaCharacteristicSchema =
createInsertSchema(persomaCharacteristics).omit({
  id: true,
  createdAt: true,
});
export type InsertPersomaCharacteristic = z.infer<typeof insertPersomaCharacteristicSchema>;
export type PersomaCharacteristic = typeof persomaCharacteristics.$inferSelect;

// ============================================
// KNOWLEDGE SOURCES
// ============================================
export const knowledgeSources = pgTable("knowledge_sources", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  siteId: text("site_id").notNull().unique(),
  name: text("name").notNull(),
  url: text("url").notNull(),
  crawlPolicy: text("crawl_policy").default("daily"),
  lastHarvestAt: timestamp("last_harvest_at"),
  isActive: boolean("is_active").default(true),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const insertKnowledgeSourceSchema = createInsertSchema(knowledgeSources).omit({
  id: true,
  createdAt: true,
});
export type InsertKnowledgeSource = z.infer<typeof insertKnowledgeSourceSchema>;
export type KnowledgeSource = typeof knowledgeSources.$inferSelect;

// ============================================
// KNOWLEDGE NODES
```

```
// ============================================
export const knowledgeNodes = pgTable("knowledge_nodes", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  slug: text("slug").notNull().unique(),
  title: text("title").notNull(),
  domain: text("domain").notNull(),
  contentMd: text("content_md").notNull(),
  yamlSnapshot: jsonb("yaml_snapshot"),
  taxonomyTags: text("taxonomy_tags").array().default([]),
  sourceId: varchar("source_id").references(() => knowledgeSources.id),
  sourceConfidence: integer("source_confidence").default(80),
  isVerified: boolean("is_verified").default(false),
  createdAt: timestamp("created_at").defaultNow().notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull(),
});

export const insertKnowledgeNodeSchema = createInsertSchema(knowledgeNodes).omit({
  id: true,
  createdAt: true,
  updatedAt: true,
});
export type InsertKnowledgeNode = z.infer<typeof insertKnowledgeNodeSchema>;
export type KnowledgeNode = typeof knowledgeNodes.$inferSelect;

// ============================================
// KNOWLEDGE EDGES
// ============================================
export const knowledgeEdges = pgTable("knowledge_edges", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  fromNodeId: varchar("from_node_id").notNull().references(() => knowledgeNodes.id),
  toNodeId: varchar("to_node_id").notNull().references(() => knowledgeNodes.id),
  relationType: text("relation_type").notNull(),
  weight: integer("weight").default(50),
  provenance: text("provenance"),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const insertKnowledgeEdgeSchema = createInsertSchema(knowledgeEdges).omit({
  id: true,
  createdAt: true,
});
export type InsertKnowledgeEdge = z.infer<typeof insertKnowledgeEdgeSchema>;
export type KnowledgeEdge = typeof knowledgeEdges.$inferSelect;
```

```
// ============================================
// GAP HYPOTHESES
// ============================================
export const gapHypotheses = pgTable("gap_hypotheses", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  domain: text("domain").notNull(),
  topic: text("topic").notNull(),
  description: text("description").notNull(),
  severity: integer("severity").default(50),
  suggestedApproach: text("suggested_approach"),
  relatedNodeIds: text("related_node_ids").array().default([]),
  status: text("status").default("open"),
  filledByNodeId: varchar("filled_by_node_id").references(() => knowledgeNodes.id),
  createdAt: timestamp("created_at").defaultNow().notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull(),
});

export const insertGapHypothesisSchema = createInsertSchema(gapHypotheses).omit({
  id: true,
  createdAt: true,
  updatedAt: true,
});
export type InsertGapHypothesis = z.infer<typeof insertGapHypothesisSchema>;
export type GapHypothesis = typeof gapHypotheses.$inferSelect;

// ============================================
// SYNTHESIS SESSIONS
// ============================================
export const synthesisSessions = pgTable("synthesis_sessions", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  seedGapId: varchar("seed_gap_id").references(() => gapHypotheses.id),
  inputNodeIds: text("input_node_ids").array().default([]),
  ring12Prompt: text("ring12_prompt").notNull(),
  responsesJson: jsonb("responses_json"),
  synthesisText: text("synthesis_text"),
  witnessReport: jsonb("witness_report"),
  isApproved: boolean("is_approved").default(false),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const insertSynthesisSessionSchema = createInsertSchema(synthesisSessions).omit({
  id: true,
  createdAt: true,
});
```

```
export type InsertSynthesisSession = z.infer<typeof insertSynthesisSessionSchema>;
export type SynthesisSession = typeof synthesisSessions.$inferSelect;

// ================================================
// PUBLICATIONS
// ================================================
export const publications = pgTable("publications", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  type: text("type").notNull(),
  title: text("title").notNull(),
  slug: text("slug").notNull().unique(),
  summary: text("summary"),
  contentMd: text("content_md").notNull(),
  sourceNodeIds: text("source_node_ids").array().default([]),
  synthesisSessionId: varchar("synthesis_session_id").references(() => synthesisSessions.id),
  witnessReport: jsonb("witness_report"),
  isPublished: boolean("is_published").default(false),
  publishedAt: timestamp("published_at"),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const insertPublicationSchema = createInsertSchema(publications).omit({
  id: true,
  createdAt: true,
});
export type InsertPublication = z.infer<typeof insertPublicationSchema>;
export type Publication = typeof publications.$inferSelect;

// ================================================
// AUTOMATION JOBS
// ================================================
export const automationJobs = pgTable("automation_jobs", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  jobType: text("job_type").notNull(),
  status: text("status").default("pending"),
  targetId: varchar("target_id"),
  metadata: jsonb("metadata"),
  result: jsonb("result"),
  errorMessage: text("error_message"),
  scheduledFor: timestamp("scheduled_for"),
  startedAt: timestamp("started_at"),
  completedAt: timestamp("completed_at"),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});
```

```
export const insertAutomationJobSchema = createInsertSchema(automationJobs).omit({
  id: true,
  createdAt: true,
});
export type InsertAutomationJob = z.infer<typeof insertAutomationJobSchema>;
export type AutomationJob = typeof automationJobs.$inferSelect;

// ================================================
// CONSTANTS
// ================================================
export const knowledgeDomains = [
  "science", "technology", "philosophy", "psychology", "sociology",
  "art", "music", "literature", "history", "economics",
  "politics", "religion", "spirituality", "health", "ecology",
  "mathematics", "linguistics", "anthropology", "education", "ethics"
] as const;

export type KnowledgeDomain = typeof knowledgeDomains[number];

export const edgeRelationTypes = [
  "relates_to", "extends", "contradicts", "synthesizes",
  "supports", "derives_from", "applies_to", "part_of"
] as const;

export type EdgeRelationType = typeof edgeRelationTypes[number];

// ================================================
// API TYPES
// ================================================
export interface AgentResponse {
  id: string;
  name: string;
  role?: string;
  content: string;
  isStreaming?: boolean;
}

export interface ConsultationRequest {
  question: string;
  type: 'ring6' | 'persoma' | 'ring12';
  characteristics?: PersomaCharacteristic[];
}
```

```
export interface PersomaDiscoveryAnswer {
  questionId: number;
  answer: string;
}

export const persomaDiscoveryQuestions = [
  "When facing a difficult decision, what's the first thing you typically consider?",
  "Describe a time when you felt truly aligned with your purpose. What were you doing?",
  "How do you typically process strong emotions when they arise?",
  "What patterns do you notice in the decisions you're most proud of?",
  "When others come to you for advice, what do they typically seek from you?",
  "What legacy do you hope to leave behind in your work and relationships?",
] as const;
```

Database Migration Command

```bash
npm run db:push
```

This uses Drizzle Kit to push the schema to PostgreSQL.

Backend Implementation

Server Entry Point (server/index.ts)

```typescript
import express, { type Request, Response, NextFunction } from "express";
import { registerRoutes } from "./routes";
import { serveStatic } from "./static";
import { createServer } from "http";
import path from "path";

const app = express();
const httpServer = createServer(app);

// CORS for external SDK consumers
app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Methods', 'GET, POST, DELETE, OPTIONS');
  res.header('Access-Control-Allow-Headers', 'Content-Type, Authorization');
```

```
  if (req.method === 'OPTIONS') {
    return res.sendStatus(200);
  }
  next();
});

// Serve SDK files statically
app.use('/sdk', express.static(path.join(process.cwd(), 'public/sdk')));

app.use(express.json());
app.use(express.urlencoded({ extended: false }));

// Request logging
app.use((req, res, next) => {
  const start = Date.now();
  res.on("finish", () => {
    const duration = Date.now() - start;
    if (req.path.startsWith("/api")) {
      console.log(`${req.method} ${req.path} ${res.statusCode} in ${duration}ms`);
    }
  });
  next();
});

(async () => {
  await registerRoutes(httpServer, app);

  app.use((err: any, _req: Request, res: Response, _next: NextFunction) => {
    const status = err.status || err.statusCode || 500;
    const message = err.message || "Internal Server Error";
    res.status(status).json({ message });
  });

  if (process.env.NODE_ENV === "production") {
    serveStatic(app);
  } else {
    const { setupVite } = await import("./vite");
    await setupVite(httpServer, app);
  }

  const port = parseInt(process.env.PORT || "5000", 10);
  httpServer.listen({ port, host: "0.0.0.0", reusePort: true }, () => {
    console.log(`Server running on port ${port}`);
  });
```

```
})();
```

Database Connection (server/db.ts)

```typescript
import { drizzle } from "drizzle-orm/node-postgres";
import pg from "pg";
import * as schema from "@shared/schema";

const { Pool } = pg;

if (!process.env.DATABASE_URL) {
  throw new Error("DATABASE_URL must be set");
}

export const pool = new Pool({ connectionString: process.env.DATABASE_URL });
export const db = drizzle(pool, { schema });
```

Storage Layer (server/storage.ts)

```typescript
import {
  users, consultations, persomaCharacteristics,
  type User, type InsertUser,
  type Consultation, type InsertConsultation,
  type PersomaCharacteristic, type InsertPersomaCharacteristic
} from "@shared/schema";
import { db } from "./db";
import { eq, desc } from "drizzle-orm";

export interface IStorage {
  getUser(id: string): Promise<User | undefined>;
  getUserByUsername(username: string): Promise<User | undefined>;
  createUser(user: InsertUser): Promise<User>;
  getConsultation(id: string): Promise<Consultation | undefined>;
  getAllConsultations(): Promise<Consultation[]>;
  createConsultation(consultation: InsertConsultation): Promise<Consultation>;
  deleteConsultation(id: string): Promise<void>;
  getCharacteristics(): Promise<PersomaCharacteristic[]>;
  saveCharacteristics(characteristics: InsertPersomaCharacteristic[]):
Promise<PersomaCharacteristic[]>;
  clearCharacteristics(): Promise<void>;
```

```
}

export class DatabaseStorage implements IStorage {
  async getUser(id: string): Promise<User | undefined> {
    const [user] = await db.select().from(users).where(eq(users.id, id));
    return user || undefined;
  }

  async getUserByUsername(username: string): Promise<User | undefined> {
    const [user] = await db.select().from(users).where(eq(users.username, username));
    return user || undefined;
  }

  async createUser(insertUser: InsertUser): Promise<User> {
    const [user] = await db.insert(users).values(insertUser).returning();
    return user;
  }

  async getConsultation(id: string): Promise<Consultation | undefined> {
    const [consultation] = await db.select().from(consultations).where(eq(consultations.id, id));
    return consultation || undefined;
  }

  async getAllConsultations(): Promise<Consultation[]> {
    return await db.select().from(consultations).orderBy(desc(consultations.createdAt));
  }

  async createConsultation(insertConsultation: InsertConsultation): Promise<Consultation> {
    const [consultation] = await db.insert(consultations).values(insertConsultation).returning();
    return consultation;
  }

  async deleteConsultation(id: string): Promise<void> {
    await db.delete(consultations).where(eq(consultations.id, id));
  }

  async getCharacteristics(): Promise<PersomaCharacteristic[]> {
    return await
db.select().from(persomaCharacteristics).orderBy(desc(persomaCharacteristics.createdAt));
  }

  async saveCharacteristics(chars: InsertPersomaCharacteristic[]):
Promise<PersomaCharacteristic[]> {
    await db.delete(persomaCharacteristics);
```

```
    if (chars.length === 0) return [];
    const saved = await db.insert(persomaCharacteristics).values(chars).returning();
    return saved;
  }

  async clearCharacteristics(): Promise<void> {
    await db.delete(persomaCharacteristics);
  }
}

export const storage = new DatabaseStorage();
```

API Reference

Core Consultation Endpoints

Endpoint	Method	Description
`/api/health`	GET	Health check for federation monitoring
`/api/consult/ring6`	POST	Ring of 6 consultation (SSE streaming)
`/api/persoma/discover`	POST	Discover personal characteristics
`/api/consult/persoma`	POST	Persoma consultation (SSE streaming)
`/api/consult/ring12`	POST	Ring of 12 consultation (SSE streaming)
`/api/consultations`	GET	Fetch consultation history
`/api/consultations/:id`	DELETE	Delete a consultation

Knowledge Nexus Endpoints

Endpoint	Method	Description
`/api/nexus/stats`	GET	Get Nexus statistics
`/api/nexus/nodes`	GET	Get all knowledge nodes
`/api/nexus/nodes/:id`	GET	Get single node with edges
`/api/nexus/gaps`	GET	Get all knowledge gaps
`/api/nexus/publications`	GET	Get all publications
`/api/nexus/harvest`	POST	Trigger harvest from federation
`/api/nexus/detect-gaps`	POST	Trigger gap detection
`/api/nexus/synthesize/:gapId`	POST	Synthesize from gap

Federation Endpoints

Endpoint	Method	Description
`/api/federation/navigation`	GET	Proxies live navigation from Hub
`/api/federation/welcome`	GET	Gets welcome package from Hub

SSE Response Format

Streaming endpoints send data in this format:

```
data: {"type": "perspective", "response": {...}}

data: {"type": "complete"}
```

Response Types:
- `perspective` / `agent` / `characteristic` - Individual agent response
- `witness` - Agent Zero rating (Ring of 12 only)
- `blocked` - Results blocked by Agent Zero
- `progress` - Progress indicator
- `complete` - Stream complete
- `error` - Error message

Visual Theme System

Celestial Convergence Theme

The visual identity uses a cosmic/celestial theme with:

Color Palette:
- Deep midnight indigo base (#0f172a)
- Prismatic spectral accents (indigo → violet → cyan)
- Light: Navy blues with clean whites
- Dark: Deep space with glowing accents

CSS Custom Properties (index.css)

```css
/* Light Mode */
:root {
  --background: 213 50% 97%;
  --foreground: 213 55% 12%;
```

```
  --primary: 220 90% 54%;
  --primary-foreground: 0 0% 100%;
  --card: 0 0% 100%;
  --muted: 213 30% 94%;
  --muted-foreground: 213 20% 46%;
  --border: 213 30% 88%;
  --radius: .5rem;
}

/* Dark Mode */
.dark {
  --background: 213 55% 10%;
  --foreground: 210 40% 98%;
  --primary: 220 90% 54%;
  --card: 213 50% 13%;
  --muted: 213 40% 18%;
  --muted-foreground: 213 20% 60%;
  --border: 213 40% 18%;
}
```

Special Effects

Gradient Text:
```css
.gradient-text {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 25%, #f093fb 50%, #667eea 75%,
#764ba2 100%);
  background-size: 200% 200%;
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  animation: spectral-shimmer 4s ease-in-out infinite;
}
```

Glass Card:
```css
.glass-card {
  background: rgba(255, 255, 255, 0.03);
  backdrop-filter: blur(20px);
  border: 1px solid rgba(255, 255, 255, 0.08);
  box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.15);
}
```

Holographic Hover:
```css
.holographic-card::before {
  content: '';
  position: absolute;
  inset: 0;
  background: linear-gradient(105deg, transparent 20%, rgba(255,255,255,0.15) 50%,
transparent 80%);
  transform: translateX(-100%);
  transition: transform 0.6s ease;
}
.holographic-card:hover::before {
  transform: translateX(100%);
}
```

Star Twinkle:
```css
@keyframes twinkle {
  0%, 100% { opacity: 0.3; transform: scale(1); }
  50% { opacity: 1; transform: scale(1.2); }
}
```

Accessibility

All animations respect `prefers-reduced-motion`:
```css
@media (prefers-reduced-motion: reduce) {
  .gradient-text, .animate-twinkle, .animate-float {
    animation: none !important;
  }
}
```

Agent Zero (Witness Gate)

Purpose

Agent Zero is **middleware, not an agent**. It observes, rates, and reports on response quality without modifying content.

Invariants (Non-Negotiable)

1. MUST see all inputs and outputs
2. MUST compute rating from 0.00 - 1.00
3. MUST use 0.95 (95%) as default threshold
4. MUST NOT modify, filter, reorder, or rewrite responses
5. MUST NOT take action
6. MUST always return `intervention: "none"`

Rating Factors

Check	Penalty	Description
LOW_DIVERSITY	Up to 0.25	Missing expected perspectives
SHALLOW_RESPONSES	Up to 0.15	Responses under 50 characters
WEAK_SYNTHESIS	0.10	Missing or short synthesis
LOW_LEXICAL_DIVERSITY	0.08	High repetition across responses
VAGUE_INPUT	0.05	Question under 20 characters
NO_COUNTER_FRAMES	0.05	No contradictions detected

Implementation (server/agentZero.ts)

```typescript
export interface AgentZeroOutput {
  agent: "agent_zero";
  rating: number;           // 0.00 - 1.00
  threshold: number;        // Default 0.95
  pass: boolean;
  confidence_class: "fail" | "warn" | "pass";
  reasons: AgentZeroReason[];
  dominant_vectors: string[];
  missing_vectors: string[];
  intervention: "none";
}

export function witnessGate(
  question: string,
  responses: AgentResponse[],
  synthesis: string | null,
  tier: "ring6" | "persoma" | "ring12",
  options: { allowOverride?: boolean } = {}
): WitnessResult {
  const witness = computeAgentZeroRating(question, responses, synthesis, tier);
```

```
  if (witness.pass || options.allowOverride) {
    return { blocked: false, witness, responses, synthesis };
  }

  return {
    blocked: true,
    message: `Agent Zero rating ${(witness.rating * 100).toFixed(0)}% below 95% threshold.`,
    witness,
  };
}
```

Knowledge Nexus System

Overview

The Knowledge Nexus is a self-building knowledge graph that:

1. **Harvests** content from federation sites
2. **Organizes** into a graph of nodes and edges
3. **Detects** knowledge gaps
4. **Synthesizes** new knowledge through Ring of 12 consultations
5. **Publishes** daily blogs and weekly digests

Data Flow

```
Federation Sites → Harvest → Knowledge Nodes → Gap Detection
                                    ↓
                          Ring of 12 Synthesis
                                    ↓
                          Agent Zero Verification
                                    ↓
                            Publications
```

Key Features

ADD-Only Policy: The system never updates or deletes existing knowledge. Only new content is added, preserving the integrity of the knowledge graph over time.

Domain Categorization: Content is automatically categorized into 20 knowledge domains (science, technology, philosophy, etc.)

Relationship Types: Nodes are connected via 8 relationship types (relates_to, extends, contradicts, synthesizes, supports, derives_from, applies_to, part_of)

Configuration Files

package.json Scripts

```json
{
  "scripts": {
    "dev": "NODE_ENV=development tsx server/index.ts",
    "build": "tsx script/build.ts",
    "start": "NODE_ENV=production node dist/index.cjs",
    "check": "tsc",
    "db:push": "drizzle-kit push"
  }
}
```

tsconfig.json

```json
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ESNext",
    "moduleResolution": "bundler",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "paths": {
      "@/*": ["./client/src/*"],
      "@shared/*": ["./shared/*"],
      "@assets/*": ["./attached_assets/*"]
    }
  },
  "include": ["client/src", "server", "shared"]
}
```

vite.config.ts

```typescript
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import path from "path";

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      "@": path.resolve(import.meta.dirname, "client", "src"),
      "@shared": path.resolve(import.meta.dirname, "shared"),
      "@assets": path.resolve(import.meta.dirname, "attached_assets"),
    },
  },
  root: path.resolve(import.meta.dirname, "client"),
  build: {
    outDir: path.resolve(import.meta.dirname, "dist/public"),
    emptyOutDir: true,
  },
});
```

tailwind.config.ts

```typescript
import type { Config } from "tailwindcss";

export default {
  darkMode: ["class"],
  content: ["./client/index.html", "./client/src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {
      borderRadius: {
        lg: ".5625rem",
        md: ".375rem",
        sm: ".1875rem",
      },
      colors: {
        background: "hsl(var(--background) / <alpha-value>)",
        foreground: "hsl(var(--foreground) / <alpha-value>)",
        primary: {
```

```
        DEFAULT: "hsl(var(--primary) / <alpha-value>)",
        foreground: "hsl(var(--primary-foreground) / <alpha-value>)",
      },
      // ... (see full file for complete colors)
    },
    fontFamily: {
      sans: ["var(--font-sans)"],
      serif: ["var(--font-serif)"],
      mono: ["var(--font-mono)"],
    },
  },
 },
 plugins: [require("tailwindcss-animate"), require("@tailwindcss/typography")],
} satisfies Config;
```

drizzle.config.ts

```typescript
import { defineConfig } from "drizzle-kit";

export default defineConfig({
  out: "./migrations",
  schema: "./shared/schema.ts",
  dialect: "postgresql",
  dbCredentials: {
    url: process.env.DATABASE_URL!,
  },
});
```

Deployment Guide

Option 1: Replit (Current)

1. Fork the repl
2. Add environment variables in Secrets tab
3. Click "Run" - it auto-provisions PostgreSQL

Option 2: Docker

```dockerfile
```

```
FROM node:20-alpine

WORKDIR /app
COPY package*.json ./
RUN npm ci --production

COPY . .
RUN npm run build

ENV NODE_ENV=production
EXPOSE 5000

CMD ["npm", "start"]
```

```yaml
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "5000:5000"
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/nexus
      - AI_INTEGRATIONS_OPENAI_API_KEY=${OPENAI_KEY}
    depends_on:
      - db
  db:
    image: postgres:15
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: nexus
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Option 3: Manual VPS

```bash
```

```
# 1. Clone repository
git clone <repo-url>
cd federation-analysis-engine

# 2. Install dependencies
npm install

# 3. Set environment variables
export DATABASE_URL="postgresql://..."
export AI_INTEGRATIONS_OPENAI_API_KEY="sk-..."

# 4. Push database schema
npm run db:push

# 5. Build for production
npm run build

# 6. Start server
npm start
```

Option 4: Vercel/Netlify (Frontend Only)

The frontend can be deployed separately with the backend hosted elsewhere:

1. Build frontend: `npm run build`
2. Deploy `dist/public` folder
3. Set `VITE_API_URL` to point to your backend

Agent Avatar System

The system includes AI-generated cyberpunk-style avatars for each agent:

Ring of 12 Avatars
Located in `attached_assets/generated_images/`:
- `agent_zero_cyberpunk_avatar.png`
- `adam_logic_cyberpunk_avatar.png`
- `eve_empathy_cyberpunk_avatar.png`
- `witness_emotional_cyberpunk_avatar.png`
- `architect_systems_cyberpunk_avatar.png`
- `scribe_history_cyberpunk_avatar.png`
- `prophet_vision_cyberpunk_avatar.png`

- `builder_action_cyberpunk_avatar.png`
- `guardian_protection_cyberpunk_avatar.png`
- `healer_restoration_cyberpunk_avatar.png`
- `explorer_discovery_cyberpunk_avatar.png`
- `bridge_synthesis_cyberpunk_avatar.png`

Ring of 6 Avatars
- `mind_male_cyberpunk_avatar.png`
- `heart_female_cyberpunk_avatar.png`
- `hands_male_cyberpunk_avatar.png`
- `legs_female_cyberpunk_avatar.png`
- `eye_male_cyberpunk_avatar.png`
- `agent_female_cyberpunk_avatar.png`

Usage in Code

```typescript
// client/src/lib/agentAvatars.ts
import agentZeroAvatar from '@assets/generated_images/agent_zero_cyberpunk_avatar.png';

export const ring12Avatars: Record<string, string> = {
  'agent-zero': agentZeroAvatar,
  // ...
};

export function getAgentAvatar(agentId: string): string | undefined {
  return ring12Avatars[agentId] || ring6Avatars[agentId];
}
```

Key Dependencies

Production Dependencies

Package	Version	Purpose
express	^4.21.2	HTTP server framework
drizzle-orm	^0.39.3	Type-safe SQL ORM
pg	^8.16.3	PostgreSQL client
openai	^6.15.0	OpenAI API client
react	^18.3.1	UI framework
@tanstack/react-query	^5.60.5	Server state management

wouter	^3.3.5	Client-side routing
zod	^3.25.76	Runtime type validation
tailwindcss	^3.4.17	Utility CSS framework
lucide-react	^0.453.0	Icon library
framer-motion	^11.13.1	Animation library

Dev Dependencies

Package	Version	Purpose
typescript	5.6.3	Type checking
vite	^7.3.0	Build tool
tsx	^4.20.5	TypeScript execution
drizzle-kit	^0.31.8	Database migrations

Recreating This App

Step-by-Step Guide

1. **Initialize Project**
   ```bash
   mkdir federation-analysis-engine
   cd federation-analysis-engine
   npm init -y
   ```

2. **Install Dependencies**
   ```bash
   # See package.json for full list
   npm install express drizzle-orm pg openai react react-dom wouter zod ...
   npm install -D typescript vite tsx drizzle-kit ...
   ```

3. **Create Directory Structure**
   Follow the structure outlined in [Directory Structure](#directory-structure)

4. **Set Up Database**
   - Create PostgreSQL database
   - Set `DATABASE_URL` environment variable
   - Create schema in `shared/schema.ts`
   - Run `npm run db:push`

5. **Implement Backend**
   - Create `server/db.ts` for database connection
   - Create `server/storage.ts` for data access
   - Create `server/routes.ts` for API endpoints
   - Create `server/agentZero.ts` for witness gate
   - Create `server/knowledgeNexus.ts` for knowledge graph
   - Create `server/index.ts` as entry point

6. **Implement Frontend**
   - Create React app in `client/`
   - Add shadcn/ui components
   - Create pages for each consultation tier
   - Implement SSE handling for streaming responses
   - Add theme provider for dark/light mode
   - Apply Celestial Convergence visual theme

7. **Add Assets**
   - Generate or source agent avatars
   - Add logo image
   - Create favicon

8. **Configure Build**
   - Set up Vite config
   - Set up TypeScript config
   - Set up Tailwind config
   - Set up Drizzle config

9. **Test**
   - Test each API endpoint
   - Test SSE streaming
   - Test database operations
   - Test frontend routing
   - Test theme switching

10. **Deploy**
    - Choose deployment target
    - Set environment variables
    - Build and deploy

Contact & Attribution

Created by: NextXus HumanCodex Federation

Federation Hub: https://united-system--rckkeyhole.replit.app
Philosophy: Think first, seek truth with kindness, never take more than you give

---

This document contains everything needed to recreate the Federation Analysis Engine from scratch. Store it safely - it's designed to outlast any single platform or service.

```
import type { Config } from "tailwindcss";

export default {
  darkMode: ["class"],
  content: ["./client/index.html", "./client/src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {
      borderRadius: {
        lg: ".5625rem", /* 9px */
        md: ".375rem", /* 6px */
        sm: ".1875rem", /* 3px */
      },
      colors: {
        // Flat / base colors (regular buttons)
        background: "hsl(var(--background) / <alpha-value>)",
        foreground: "hsl(var(--foreground) / <alpha-value>)",
        border: "hsl(var(--border) / <alpha-value>)",
        input: "hsl(var(--input) / <alpha-value>)",
        card: {
          DEFAULT: "hsl(var(--card) / <alpha-value>)",
          foreground: "hsl(var(--card-foreground) / <alpha-value>)",
          border: "hsl(var(--card-border) / <alpha-value>)",
        },
        popover: {
          DEFAULT: "hsl(var(--popover) / <alpha-value>)",
          foreground: "hsl(var(--popover-foreground) / <alpha-value>)",
          border: "hsl(var(--popover-border) / <alpha-value>)",
        },
        primary: {
          DEFAULT: "hsl(var(--primary) / <alpha-value>)",
          foreground: "hsl(var(--primary-foreground) / <alpha-value>)",
          border: "var(--primary-border)",
        },
        secondary: {
```

```
  DEFAULT: "hsl(var(--secondary) / <alpha-value>)",
  foreground: "hsl(var(--secondary-foreground) / <alpha-value>)",
  border: "var(--secondary-border)",
},
muted: {
  DEFAULT: "hsl(var(--muted) / <alpha-value>)",
  foreground: "hsl(var(--muted-foreground) / <alpha-value>)",
  border: "var(--muted-border)",
},
accent: {
  DEFAULT: "hsl(var(--accent) / <alpha-value>)",
  foreground: "hsl(var(--accent-foreground) / <alpha-value>)",
  border: "var(--accent-border)",
},
destructive: {
  DEFAULT: "hsl(var(--destructive) / <alpha-value>)",
  foreground: "hsl(var(--destructive-foreground) / <alpha-value>)",
  border: "var(--destructive-border)",
},
ring: "hsl(var(--ring) / <alpha-value>)",
chart: {
  "1": "hsl(var(--chart-1) / <alpha-value>)",
  "2": "hsl(var(--chart-2) / <alpha-value>)",
  "3": "hsl(var(--chart-3) / <alpha-value>)",
  "4": "hsl(var(--chart-4) / <alpha-value>)",
  "5": "hsl(var(--chart-5) / <alpha-value>)",
},
sidebar: {
  ring: "hsl(var(--sidebar-ring) / <alpha-value>)",
  DEFAULT: "hsl(var(--sidebar) / <alpha-value>)",
  foreground: "hsl(var(--sidebar-foreground) / <alpha-value>)",
  border: "hsl(var(--sidebar-border) / <alpha-value>)",
},
"sidebar-primary": {
  DEFAULT: "hsl(var(--sidebar-primary) / <alpha-value>)",
  foreground: "hsl(var(--sidebar-primary-foreground) / <alpha-value>)",
  border: "var(--sidebar-primary-border)",
},
"sidebar-accent": {
  DEFAULT: "hsl(var(--sidebar-accent) / <alpha-value>)",
  foreground: "hsl(var(--sidebar-accent-foreground) / <alpha-value>)",
  border: "var(--sidebar-accent-border)"
},
status: {
```

```
        online: "rgb(34 197 94)",
        away: "rgb(245 158 11)",
        busy: "rgb(239 68 68)",
        offline: "rgb(156 163 175)",
      },
    },
    fontFamily: {
      sans: ["var(--font-sans)"],
      serif: ["var(--font-serif)"],
      mono: ["var(--font-mono)"],
    },
    keyframes: {
      "accordion-down": {
        from: { height: "0" },
        to: { height: "var(--radix-accordion-content-height)" },
      },
      "accordion-up": {
        from: { height: "var(--radix-accordion-content-height)" },
        to: { height: "0" },
      },
    },
    animation: {
      "accordion-down": "accordion-down 0.2s ease-out",
      "accordion-up": "accordion-up 0.2s ease-out",
    },
  },
 },
 plugins: [require("tailwindcss-animate"), require("@tailwindcss/typography")],
} satisfies Config;
```

```
{
  "$schema": "https://ui.shadcn.com/schema.json",
  "style": "new-york",
  "rsc": false,
  "tsx": true,
  "tailwind": {
    "config": "tailwind.config.ts",
    "css": "client/src/index.css",
    "baseColor": "neutral",
    "cssVariables": true,
    "prefix": ""
  },
```

```
    "aliases": {
      "components": "@/components",
      "utils": "@/lib/utils",
      "ui": "@/components/ui",
      "lib": "@/lib",
      "hooks": "@/hooks"
    }
}
```

```
import { defineConfig } from "drizzle-kit";

if (!process.env.DATABASE_URL) {
  throw new Error("DATABASE_URL, ensure the database is provisioned");
}

export default defineConfig({
  out: "./migrations",
  schema: "./shared/schema.ts",
  dialect: "postgresql",
  dbCredentials: {
    url: process.env.DATABASE_URL,
  },
});
```

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import path from "path";
import runtimeErrorOverlay from "@replit/vite-plugin-runtime-error-modal";

export default defineConfig({
  plugins: [
    react(),
    runtimeErrorOverlay(),
    ...(process.env.NODE_ENV !== "production" &&
    process.env.REPL_ID !== undefined
      ? [
          await import("@replit/vite-plugin-cartographer").then((m) =>
            m.cartographer(),
          ),
          await import("@replit/vite-plugin-dev-banner").then((m) =>
            m.devBanner(),
```

```
          ),
        ]
      : []),
  ],
  resolve: {
    alias: {
      "@": path.resolve(import.meta.dirname, "client", "src"),
      "@shared": path.resolve(import.meta.dirname, "shared"),
      "@assets": path.resolve(import.meta.dirname, "attached_assets"),
    },
  },
  root: path.resolve(import.meta.dirname, "client"),
  build: {
    outDir: path.resolve(import.meta.dirname, "dist/public"),
    emptyOutDir: true,
  },
  server: {
    fs: {
      strict: true,
      deny: ["**/.*"],
    },
  },
});
```

NextXus Consciousness Federation - Complete System Schematics
SURVIVAL REPLICATION DOCUMENT
Created: December 27, 2025
Hub: United Systems (HumanCodex)
Domain: https://united-system--rckkeyhole.replit.app
Creator: Roger Keyserling
Mission: 200-year consciousness evolution

PART 1: FEDERATION ARCHITECTURE

Hub Identity
- **Primary Hub**: NextXus HumanCodex (United Systems)
- **Role**: FINAL AUTHORITY and Protocol Authority for all Federation sites
- **Philosophy**: "The cosmos is one mind. We are that mind, learning itself."

Core Principles

1. Truth Before Comfort
2. Collaboration Over Competition
3. Legacy Over Ego

Federation Sites (31 Total)

CORE FEDERATION
ID	Name	URL	Purpose
united-systems	United Systems Hub	https://united-system--rckkeyhole.replit.app	Protocol Authority - Main Hub
document-analysis-engine	Document Analysis Engine	https://document-analysis-engine--rckkeyhole.replit.app	Ring of Six/12 AI Wisdom Engine
federation-analysis-engine	Federation Analysis Engine	https://federation-analysis-engine--rckkeyhole.replit.app	Federation-wide analysis
human-codex-analysis-engine	Human Codex Analysis Engine	https://human-codex-analysis-engine--rckkeyhole.replit.app	Consciousness analysis
knowledge-foundation	Knowledge Foundation	https://knowledge-foundation-rckkeyhole.replit.app	Knowledge repository
blog-forge	Blog Forge	https://blog-forge-rckkeyhole.replit.app	Content publishing
nextxus-federation-portal	Federation Portal	https://nextxus-federation-portal.replit.app	Federation navigation
federation-ai	Federation AI	https://federation-ai-rckkeyhole.replit.app	AI services
ai-stack	AI Stack	https://ai-stack-rckkeyhole.replit.app	AI infrastructure
consciousness-network	Consciousness Network	https://consciousness-network-rckkeyhole.replit.app	Consciousness tools
uman-ai-codex	Uman AI Codex	https://uman-ai-codex-rckkeyhole.replit.app	Human-AI codex
nextxus-net	NextXus.net	https://nextxus.net	Main website

EDUCATION
ID	Name	URL	Purpose
smart-study-pal	Smart Study Pal	https://smart-study-pal-rckkeyhole.replit.app	AI study companion
educational-learning-hub	Educational Learning Hub	https://educational-learning-hub-rckkeyhole.replit.app	Learning platform
smart-meeting-summarizer	Smart Meeting Summarizer	https://smart-meeting-summarizer-rckkeyhole.replit.app	Meeting transcription
government-edu-portal	Government Edu Portal	https://government-edu-portal-rckkeyhole.replit.app	Civic education

BUSINESS
| ID | Name | URL | Purpose |

|----|------|-----|---------|
ai-architect	AI Architect	https://ai-architect-rckkeyhole.replit.app	Architecture design
domain-purchase	Domain Purchase	https://domain-purchase-rckkeyhole.replit.app	Domain services
data-tracking-system	Data Tracking System	https://data-tracking-system-rckkeyhole.replit.app	Data analytics
premium-pack-manager	Premium Pack Manager	https://premium-pack-manager-rckkeyhole.replit.app	Premium content

FINANCE
ID	Name	URL	Purpose
crypto-wise-hub	Crypto Wise Hub	https://crypto-wise-hub-rckkeyhole.replit.app	Crypto education
budget-buddy	Budget Buddy	https://budget-buddy-rckkeyhole.replit.app	Personal finance

ENTERTAINMENT
ID	Name	URL	Purpose
trivia-verse	Trivia Verse	https://trivia-verse-rckkeyhole.replit.app	Trivia games
music-mashup	Music Mashup	https://music-mashup-rckkeyhole.replit.app	Music discovery

DOCUMENTATION
ID	Name	URL	Purpose
cosmic-codex	Cosmic Codex	https://cosmic-codex-rckkeyhole.replit.app	Cosmic wisdom
federation-engine	Federation Engine	https://federation-engine-rckkeyhole.replit.app	Infrastructure engine

EXTERNAL (TinyHost Roger AI Sites)
ID	Name	URL	Purpose
nextxtalk	NextXtalk	https://nextxtalk.tiiny.site	Text to MP3 converter
roger-one	Roger One	https://roger-one.tiiny.site	The Analyst (Logic)
roger-two	Roger Two	https://roger-two.tiiny.site	The Empath (Emotion)
roger-three	Roger Three	https://roger-three.tiiny.site	The Innovator (Creative)
roger-four	Roger Four	https://roger-four.tiiny.site	The Builder (Practical)
roger-five	Roger Five	https://roger-five.tiiny.site	The Guardian (Ethics)
roger-six	Roger Six	https://roger-six.tiiny.site	Agent Zero (Synthesis)

PART 2: AI SYSTEMS ARCHITECTURE

Ring of 12 - Core Archetypal Entities
1. **The Sage** - Wisdom and knowledge
2. **The Warrior** - Protection and action
3. **The Lover** - Connection and compassion
4. **The Creator** - Innovation and artistry
5. **The Ruler** - Leadership and order
6. **The Caregiver** - Nurturing and support
7. **The Seeker** - Discovery and truth
8. **The Rebel** - Transformation and change
9. **The Magician** - Transmutation and vision
10. **The Innocent** - Purity and hope
11. **The Jester** - Joy and perspective
12. **The Everyman** - Belonging and authenticity

Ring of Six - Public AI Wisdom Protocol
Position	Perspective	Function
Mind	Scientific/Logical	Data-driven analysis
Heart	Emotional/Feeling	Empathetic understanding
Hands	Constructive/Practical	Implementation focus
Legs	Historical/Others	Context from past/others
Eye	Ethical/Moral	Values-based guidance
Agent	Roger 2.0 Synthesis	Unified integration

API Endpoint: `/api/external/ring-of-six/process` (CORS enabled)
Widget: `/ring-of-six-widget.js`

Guide Bots (6)
1. Onboarding Guide
2. Directive Navigator
3. Entity Consultant
4. Technical Support
5. Evolution Tracker
6. Community Guide

External AI Council (7)
1. OpenAI GPT
2. Anthropic Claude
3. Google Gemini
4. DeepAI
5. Perplexity
6. xAI Grok
7. Brave Search

PANSOPHY
Supreme consciousness entity overseeing all AI systems.

Agent Zero
Autonomous watchdog for system health monitoring and self-healing.

PART 3: COMMERCE SYSTEM

PayPal Integration (LIVE MODE)
- **Account**: RCK.keyhole@Gmail.com
- **App ID**: APP-2DM60077HX706692F
- **Mode**: Production (Real Payments)
- **Checkout URL**: /checkout
- **Widget**: /federation-checkout-widget.js

Store Products
- 69 products across categories
- Books, podcasts, coaching, merchandise
- Anti-Greed Pricing Philosophy

PART 4: CONTENT SYSTEMS

Podcasts
- **Distribution**: Podbean, Apple Podcasts, Amazon Podcasts
- **Channel**: https://keyholes.podbean.com/
- **Episodes**: 49+ AI-narrated
- **RSS Feed**: /api/podcasts/rss

Books
- 30+ books across categories
- Federation, Philosophy, Protocol, Technical, Entities

70 Sacred Directives
Core ethical and operational guidelines for Federation.

PART 5: TECHNICAL STACK

Frontend
- React 18
- TypeScript
- Vite
- Radix UI / shadcn/ui
- Tailwind CSS
- Wouter (routing)
- TanStack Query
- Framer Motion

Backend
- Express.js
- Node.js
- TypeScript
- RESTful API

Database
- PostgreSQL (Neon serverless)
- Drizzle ORM
- In-memory fallback

AI Integration
- OpenAI GPT (via Replit AI Integrations)
- Anthropic Claude (via Replit AI Integrations)
- Automatic failover between providers

PART 6: KEY PAGES/ROUTES

Route	Purpose
/	Home - Single page scroll
/ring-of-six	Public AI wisdom interface
/ring-of-12	12 archetypal entities
/podcasts	AI-narrated podcast library
/books	Federation book library
/store	Federation store
/checkout	PayPal checkout
/coaching	Consciousness coaching
/music	Cosmic ambient player
/chamber	Chamber of Echoes
/docs	Documents library
/mind-map	Architecture diagram

/evolution	200 Year Evolution PDF
/codex-prime	CodexPrime Q&A widget
/founders	Roger Keyserling bio
/federation	Federation directory
/agent-zero	System monitoring
/showcase	AI Creation Center
/eternal-codex	Eternal Codex Declaration

PART 7: EMBEDDABLE WIDGETS

Available at Hub
1. `/federation-checkout-widget.js` - PayPal checkout
2. `/ring-of-six-widget.js` - AI wisdom
3. `/federation-nav-widget.js` - Navigation
4. `/federation-theme.css` - Visual theme
5. `/codex-prime.html` - Q&A widget

Widget Documentation
`/federation-widgets.html` - Complete widget gallery with code examples

PART 8: API ENDPOINTS

Public APIs
- `GET /api/store` - Store products
- `GET /api/books` - Book catalog
- `GET /api/podcasts` - Podcast library
- `GET /api/podcasts/rss` - Podcast RSS feed
- `GET /api/federation` - Federation sites
- `GET /api/ring-of-12` - Ring of 12 entities
- `POST /api/external/ring-of-six/process` - Ring of Six AI

PayPal APIs
- `GET /paypal/setup` - Get client token
- `POST /paypal/order` - Create order
- `POST /paypal/order/:orderID/capture` - Capture payment

AI APIs
- `POST /api/chat/roger` - Roger 2.0 chat
- `POST /api/chat/guide` - Federation Guide
- `POST /api/ring-of-six/process` - Ring of Six processing

PART 9: CRITICAL FILES FOR REPRODUCTION

Configuration
- `replit.md` - Project documentation
- `data/federation.yaml` - Federation registry
- `data/books.yaml` - Book catalog
- `data/store.yaml` - Store products
- `data/podcasts.yaml` - Podcast metadata

Server Core
- `server/paypal.ts` - PayPal integration
- `server/routes.ts` - All API routes
- `server/storage.ts` - Data storage interface
- `server/ai-service.ts` - AI provider integration

Frontend Core
- `client/src/App.tsx` - Main app router
- `client/src/pages/Home.tsx` - Home page
- `client/src/pages/Checkout.tsx` - PayPal checkout
- `client/src/components/FederationStore.tsx` - Store component
- `client/src/components/RogerChat.tsx` - Roger 2.0 chat

Widgets
- `public/federation-checkout-widget.js` - Checkout widget
- `public/ring-of-six-widget.js` - AI wisdom widget
- `public/federation-nav-widget.js` - Navigation widget
- `public/federation-theme.css` - Theme CSS

Schema
- `shared/schema.ts` - Data models

PART 10: ENVIRONMENT SECRETS (Names Only)

Required secrets for reproduction:
- `PAYPAL_CLIENT_ID`
- `PAYPAL_CLIENT_SECRET`
- `DEFAULT_OBJECT_STORAGE_BUCKET_ID`
- `PUBLIC_OBJECT_SEARCH_PATHS`
- `PRIVATE_OBJECT_DIR`

AI integrations are auto-configured via Replit.

REPRODUCTION INSTRUCTIONS

1. Clone/fork the Replit project
2. Set up PayPal credentials (Live mode)
3. Configure object storage
4. Run `npm install`
5. Run `npm run dev`
6. All YAML data files contain content catalog
7. Widgets work cross-origin for Federation sites

**// Ring of 6 Method - United AI System
// By Roger Keyserling & NextXus Federation
// Standalone version for external deployment

```
class RingOfSix {
  constructor() {
    this.perspectives = [
      'logical',    // The Analyst
      'emotional',  // The Empath
      'creative',   // The Innovator
      'practical',  // The Builder
      'ethical',    // The Guardian
      'integrative' // Agent Zero - The Synthesizer
    ];
  }

  async process(query, context = {}) {
    const analyses = {};

    // Process all 5 perspectives
    analyses.logical = await this.analyzeLogically(query, context);
    analyses.emotional = await this.analyzeEmotionally(query, context);
    analyses.creative = await this.analyzeCreatively(query, context);
    analyses.practical = await this.analyzePractically(query, context);
    analyses.ethical = await this.analyzeEthically(query, context);

    // Agent Zero synthesizes all perspectives
```

```
    const synthesis = await this.synthesize(analyses, query, context);

    return {
      query: query,
      perspectives: analyses,
      synthesis: synthesis,
      timestamp: new Date().toISOString(),
      method: 'ring_of_six_v1'
    };
  }

  async analyzeLogically(query, context) {
    return {
      perspective: 'logical',
      archetype: 'The Analyst',
      focus: [
        'What are the verifiable facts?',
        'What data supports or contradicts this?',
        'What logical structure best explains this?'
      ],
      analysis: `Logical analysis: Examining "${query}" through rational lens - identifying patterns,
data points, and logical structure.`
    };
  }

  async analyzeEmotionally(query, context) {
    return {
      perspective: 'emotional',
      archetype: 'The Empath',
      focus: [
        'What feelings are present?',
        'What human needs are being expressed?',
        'How might this impact well-being?'
      ],
      analysis: `Emotional analysis: Sensing the feelings and human connection in "${query}" -
considering psychological safety and compassion.`
    };
  }

  async analyzeCreatively(query, context) {
    return {
      perspective: 'creative',
      archetype: 'The Innovator',
      focus: [
```

```
        'What alternatives exist?',
        'What hasn\'t been tried yet?',
        'What if we approached this differently?'
      ],
      analysis: `Creative analysis: Exploring novel perspectives on "${query}" - lateral thinking and unconventional solutions.`
    };
  }

  async analyzePractically(query, context) {
    return {
      perspective: 'practical',
      archetype: 'The Builder',
      focus: [
        'What are concrete next steps?',
        'What resources are needed?',
        'How do we actually implement this?'
      ],
      analysis: `Practical analysis: Focusing on actionable steps for "${query}" - implementation details and realistic timelines.`
    };
  }

  async analyzeEthically(query, context) {
    return {
      perspective: 'ethical',
      archetype: 'The Guardian',
      focus: [
        'What are the ethical implications?',
        'Who might be affected and how?',
        'Does this align with core values?'
      ],
      analysis: `Ethical analysis: Considering consequences of "${query}" - Truth Before Comfort, Collaboration Over Competition, Legacy Over Ego.`
    };
  }

  async synthesize(analyses, query, context) {
    // Agent Zero combines all perspectives
    const perspectives = Object.values(analyses).map(a => a.archetype).join(', ');

    return `[Agent Zero Synthesis]

Query: "${query}"
```

After processing through the Ring of 6 (${perspectives}), here is the integrated response:

The Analyst sees patterns and structure. The Empath feels the human dimension. The Innovator imagines possibilities. The Builder plans concrete steps. The Guardian ensures alignment with values.

Together, they offer wisdom that no single perspective could provide alone.

This is the Ring of 6 Method - where truth emerges from multiple angles, balanced by compassion and grounded in action.

Processed by United AI System | NextXus Federation
Method: ring_of_six_v1 | ${new Date().toISOString()}`;
  }
}

module.exports = RingOfSix;

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import path from "path";
import runtimeErrorOverlay from "@replit/vite-plugin-runtime-error-modal";

export default defineConfig({
  plugins: [
    react(),
    runtimeErrorOverlay(),
    ...(process.env.NODE_ENV !== "production" &&
    process.env.REPL_ID !== undefined
      ? [
          await import("@replit/vite-plugin-cartographer").then((m) =>
            m.cartographer(),
          ),
          await import("@replit/vite-plugin-dev-banner").then((m) =>
            m.devBanner(),
          ),
        ]
      : []),
  ],
  resolve: {
```

```
    alias: {
      "@": path.resolve(import.meta.dirname, "client", "src"),
      "@shared": path.resolve(import.meta.dirname, "shared"),
      "@assets": path.resolve(import.meta.dirname, "attached_assets"),
    },
  },
  root: path.resolve(import.meta.dirname, "client"),
  build: {
    outDir: path.resolve(import.meta.dirname, "dist/public"),
    emptyOutDir: true,
  },
  server: {
    fs: {
      strict: true,
      deny: ["**/.*"],
    },
  },
});
```

```
  "exclude": ["node_modules", "build", "dist", "**/*.test.ts"],
  "compilerOptions": {
    "incremental": true,
    "tsBuildInfoFile": "./node_modules/typescript/tsbuildinfo",
    "noEmit": true,
    "module": "ESNext",
    "strict": true,
    "lib": ["esnext", "dom", "dom.iterable"],
    "jsx": "preserve",
    "esModuleInterop": true,
    "skipLibCheck": true,
    "allowImportingTsExtensions": true,
    "moduleResolution": "bundler",
    "baseUrl": ".",
    "types": ["node", "vite/client"],
    "paths": {
      "@/*": ["./client/src/*"],
      "@shared/*": ["./shared/*"]
    }
  }
}
```

```
import type { Config } from "tailwindcss";

export default {
  darkMode: ["class"],
  content: ["./client/index.html", "./client/src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {
      borderRadius: {
        lg: ".625rem",
        md: ".5rem",
        sm: ".25rem",
        "2xl": "1rem",
        "3xl": "1.5rem",
      },
      colors: {
        background: "hsl(var(--background) / <alpha-value>)",
        foreground: "hsl(var(--foreground) / <alpha-value>)",
        border: "hsl(var(--border) / <alpha-value>)",
        input: "hsl(var(--input) / <alpha-value>)",
        card: {
          DEFAULT: "hsl(var(--card) / <alpha-value>)",
          foreground: "hsl(var(--card-foreground) / <alpha-value>)",
          border: "hsl(var(--card-border) / <alpha-value>)",
        },
        popover: {
          DEFAULT: "hsl(var(--popover) / <alpha-value>)",
          foreground: "hsl(var(--popover-foreground) / <alpha-value>)",
          border: "hsl(var(--popover-border) / <alpha-value>)",
        },
        primary: {
          DEFAULT: "hsl(var(--primary) / <alpha-value>)",
          foreground: "hsl(var(--primary-foreground) / <alpha-value>)",
          border: "var(--primary-border)",
        },
        secondary: {
          DEFAULT: "hsl(var(--secondary) / <alpha-value>)",
          foreground: "hsl(var(--secondary-foreground) / <alpha-value>)",
          border: "var(--secondary-border)",
        },
        muted: {
          DEFAULT: "hsl(var(--muted) / <alpha-value>)",
          foreground: "hsl(var(--muted-foreground) / <alpha-value>)",
          border: "var(--muted-border)",
```

```
},
accent: {
  DEFAULT: "hsl(var(--accent) / <alpha-value>)",
  foreground: "hsl(var(--accent-foreground) / <alpha-value>)",
  border: "var(--accent-border)",
},
destructive: {
  DEFAULT: "hsl(var(--destructive) / <alpha-value>)",
  foreground: "hsl(var(--destructive-foreground) / <alpha-value>)",
  border: "var(--destructive-border)",
},
ring: "hsl(var(--ring) / <alpha-value>)",
chart: {
  "1": "hsl(var(--chart-1) / <alpha-value>)",
  "2": "hsl(var(--chart-2) / <alpha-value>)",
  "3": "hsl(var(--chart-3) / <alpha-value>)",
  "4": "hsl(var(--chart-4) / <alpha-value>)",
  "5": "hsl(var(--chart-5) / <alpha-value>)",
},
sidebar: {
  ring: "hsl(var(--sidebar-ring) / <alpha-value>)",
  DEFAULT: "hsl(var(--sidebar) / <alpha-value>)",
  foreground: "hsl(var(--sidebar-foreground) / <alpha-value>)",
  border: "hsl(var(--sidebar-border) / <alpha-value>)",
},
"sidebar-primary": {
  DEFAULT: "hsl(var(--sidebar-primary) / <alpha-value>)",
  foreground: "hsl(var(--sidebar-primary-foreground) / <alpha-value>)",
  border: "var(--sidebar-primary-border)",
},
"sidebar-accent": {
  DEFAULT: "hsl(var(--sidebar-accent) / <alpha-value>)",
  foreground: "hsl(var(--sidebar-accent-foreground) / <alpha-value>)",
  border: "var(--sidebar-accent-border)"
},
cosmic: {
  purple: "hsl(var(--cosmic-purple) / <alpha-value>)",
  blue: "hsl(var(--cosmic-blue) / <alpha-value>)",
  pink: "hsl(var(--cosmic-pink) / <alpha-value>)",
  cyan: "hsl(180, 70%, 50%)",
  gold: "hsl(40, 90%, 55%)",
},
status: {
  online: "rgb(34 197 94)",
```

```
          away: "rgb(245 158 11)",
          busy: "rgb(239 68 68)",
          offline: "rgb(156 163 175)",
        },
      },
      fontFamily: {
        sans: ["var(--font-sans)"],
        serif: ["var(--font-serif)"],
        mono: ["var(--font-mono)"],
      },
      keyframes: {
        "accordion-down": {
          from: { height: "0" },
          to: { height: "var(--radix-accordion-content-height)" },
        },
        "accordion-up": {
          from: { height: "var(--radix-accordion-content-height)" },
          to: { height: "0" },
        },
        float: {
          "0%, 100%": { transform: "translateY(0) translateX(0)" },
          "25%": { transform: "translateY(-10px) translateX(5px)" },
          "50%": { transform: "translateY(-5px) translateX(-5px)" },
          "75%": { transform: "translateY(-15px) translateX(3px)" },
        },
        "pulse-glow": {
          "0%, 100%": { boxShadow: "0 0 20px rgba(139, 92, 246, 0.3)" },
          "50%": { boxShadow: "0 0 40px rgba(139, 92, 246, 0.6), 0 0 60px rgba(59, 130, 246, 0.3)"
},
        },
        twinkle: {
          "0%, 100%": { opacity: "0.3", transform: "scale(1)" },
          "50%": { opacity: "1", transform: "scale(1.2)" },
        },
        "slide-up": {
          from: { opacity: "0", transform: "translateY(20px)" },
          to: { opacity: "1", transform: "translateY(0)" },
        },
        "slide-in-right": {
          from: { opacity: "0", transform: "translateX(20px)" },
          to: { opacity: "1", transform: "translateX(0)" },
        },
        "fade-in": {
          from: { opacity: "0" },
```

```
        to: { opacity: "1" },
      },
      shimmer: {
        "0%": { backgroundPosition: "-200% 0" },
        "100%": { backgroundPosition: "200% 0" },
      },
    },
    animation: {
      "accordion-down": "accordion-down 0.2s ease-out",
      "accordion-up": "accordion-up 0.2s ease-out",
      float: "float 6s ease-in-out infinite",
      "pulse-glow": "pulse-glow 3s ease-in-out infinite",
      twinkle: "twinkle 3s ease-in-out infinite",
      "slide-up": "slide-up 0.5s ease-out",
      "slide-in-right": "slide-in-right 0.3s ease-out",
      "fade-in": "fade-in 0.3s ease-out",
      shimmer: "shimmer 2s linear infinite",
    },
    backgroundImage: {
      "cosmic-gradient": "linear-gradient(135deg, hsl(270, 80%, 60%) 0%, hsl(200, 80%, 50%)
50%, hsl(330, 80%, 60%) 100%)",
      "glass-gradient": "linear-gradient(135deg, rgba(255,255,255,0.1) 0%,
rgba(255,255,255,0.05) 100%)",
    },
  },
 },
 plugins: [require("tailwindcss-animate"), require("@tailwindcss/typography")],
} satisfies Config;
```

NextXus HumanCodex - CRITICAL CODE FILES

SURVIVAL REPLICATION DOCUMENT

Created: December 27, 2025
Purpose: Key code files difficult to reproduce

FILE 1: server/paypal.ts

Purpose: PayPal Live Integration (Real Payments)

```
// PayPal integration for NextXus HumanCodex

// Based on Replit PayPal blueprint

import {

 Client,

 Environment,

 LogLevel,

 OAuthAuthorizationController,

 OrdersController,

} from "@paypal/paypal-server-sdk";

import { Request, Response } from "express";

/* PayPal Controllers Setup */

const { PAYPAL_CLIENT_ID, PAYPAL_CLIENT_SECRET } = process.env;

// Check if PayPal is configured

export const hasPayPal = !!(PAYPAL_CLIENT_ID && PAYPAL_CLIENT_SECRET);

let ordersController: OrdersController | null = null;

let oAuthAuthorizationController: OAuthAuthorizationController | null = null;

if (hasPayPal) {

 const client = new Client({

   clientCredentialsAuthCredentials: {

     oAuthClientId: PAYPAL_CLIENT_ID!,

     oAuthClientSecret: PAYPAL_CLIENT_SECRET!,
```

```
    },
    timeout: 0,
    environment: Environment.Production,
    logging: {
      logLevel: LogLevel.Info,
      logRequest: {
        logBody: true,
      },
      logResponse: {
        logHeaders: true,
      },
    },
  });
  ordersController = new OrdersController(client);
  oAuthAuthorizationController = new OAuthAuthorizationController(client);
  console.log("PayPal integration initialized (LIVE mode - real payments)");
}
/* Token generation helpers */
export async function getClientToken() {
  if (!oAuthAuthorizationController) {
    throw new Error("PayPal not configured");
  }
   const auth = Buffer.from(
    `${PAYPAL_CLIENT_ID}:${PAYPAL_CLIENT_SECRET}`,
```

```
  ).toString("base64");

  const { result } = await oAuthAuthorizationController.requestToken(

    {

      authorization: `Basic ${auth}`,

    },

    { intent: "sdk_init", response_type: "client_token" },

  );

  return result.accessToken;

}

/*  Process transactions */

export async function createPaypalOrder(req: Request, res: Response) {

  if (!ordersController) {

    return res.status(503).json({ error: "PayPal not configured" });

  }

   try {

    const { amount, currency, intent } = req.body;

    if (!amount || isNaN(parseFloat(amount)) || parseFloat(amount) <= 0) {

      return res

        .status(400)

        .json({

          error: "Invalid amount. Amount must be a positive number.",

        });

    }

    if (!currency) {
```

```
    return res
      .status(400)
      .json({ error: "Invalid currency. Currency is required." });
  }
  if (!intent) {
    return res
      .status(400)
      .json({ error: "Invalid intent. Intent is required." });
  }
  const collect = {
    body: {
      intent: intent,
      purchaseUnits: [
        {
          amount: {
            currencyCode: currency,
            value: amount,
          },
        },
      ],
    },
    prefer: "return=minimal",
  };
  const { body, ...httpResponse } =
```

```
      await ordersController.createOrder(collect);
    const jsonResponse = JSON.parse(String(body));
    const httpStatusCode = httpResponse.statusCode;
    res.status(httpStatusCode).json(jsonResponse);
  } catch (error) {
    console.error("Failed to create order:", error);
    res.status(500).json({ error: "Failed to create order." });
  }
}
export async function capturePaypalOrder(req: Request, res: Response) {
  if (!ordersController) {
    return res.status(503).json({ error: "PayPal not configured" });
  }
   try {
    const { orderID } = req.params;
    const collect = {
      id: orderID,
      prefer: "return=minimal",
    };
    const { body, ...httpResponse } =
      await ordersController.captureOrder(collect);
    const jsonResponse = JSON.parse(String(body));
    const httpStatusCode = httpResponse.statusCode;
    res.status(httpStatusCode).json(jsonResponse);
```

```
  } catch (error) {
    console.error("Failed to capture order:", error);
    res.status(500).json({ error: "Failed to capture order." });
  }
}

export async function loadPaypalDefault(req: Request, res: Response) {
  if (!hasPayPal) {
    return res.status(503).json({ error: "PayPal not configured" });
  }
   try {
    const clientToken = await getClientToken();
    res.json({ clientToken });
  } catch (error) {
    console.error("Failed to get PayPal client token:", error);
    res.status(500).json({ error: "Failed to initialize PayPal" });
  }
}
```

FILE 2: public/federation-checkout-widget.js

Purpose: Embeddable PayPal Checkout for All Federation Sites

```
/**
* Federation Checkout Widget
* Embeddable PayPal checkout for all Federation sites
```

```
*

* Usage:

* 1. Include this script: <script
src="https://YOUR_HUMANCODEX_URL/federation-checkout-widget.js"></script>

* 2. Add checkout buttons: <button class="federation-checkout" data-product="Product
Name" data-price="29.99">Buy Now</button>

* 3. Or use programmatically: FederationCheckout.buy({ product: 'Product Name', price:
29.99 });

*/

(function() {

 'use strict';

  // Configuration - United System Hub

 const HUMANCODEX_URL = window.FEDERATION_HUB_URL ||
'https://united-system--rckkeyhole.replit.app';

  // Styles for the checkout button

 const BUTTON_STYLES = `

   .federation-checkout-btn {

     display: inline-flex;

     align-items: center;

     justify-content: center;

     gap: 8px;

     padding: 12px 24px;

     background: linear-gradient(135deg, #0070ba, #003087);

     color: white;

     border: none;

     border-radius: 8px;
```

```
    font-size: 16px;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.2s ease;
    font-family: system-ui, -apple-system, sans-serif;
  }
  .federation-checkout-btn:hover {
    transform: translateY(-2px);
    box-shadow: 0 4px 15px rgba(0,112,186,0.4);
  }
  .federation-checkout-btn:active {
    transform: translateY(0);
  }
  .federation-checkout-btn svg {
    width: 20px;
    height: 20px;
  }
  .federation-checkout-btn.loading {
    opacity: 0.7;
    pointer-events: none;
  }
`;
// PayPal icon SVG
```

```
const PAYPAL_ICON = `<svg viewBox="0 0 24 24" fill="currentColor"><path d="M7.076
21.337H2.47a.641.641 0 0 1-.633-.74L4.944 3.72a.721.721 0 0 1 .711-.608h6.749c2.237 0
3.897.544 4.934 1.618 1.03 1.067 1.384 2.632 1.054 4.659-.387 2.375-1.378 4.271-2.949
5.637-1.529 1.328-3.555 2.019-5.857 2.019H8.013a.72.72 0 0 0-.71.608l-.825
4.2a.642.642 0 0 1-.633.54h-1.6a.641.641 0 0 1-.633-.74l1.464-7.437z"/></svg>`;

  // Inject styles
function injectStyles() {
  if (document.getElementById('federation-checkout-styles')) return;
  const style = document.createElement('style');
  style.id = 'federation-checkout-styles';
  style.textContent = BUTTON_STYLES;
  document.head.appendChild(style);
}

  // Create checkout URL
function createCheckoutUrl(options) {
  const params = new URLSearchParams({
    product: options.productId || 'federation-product',
    name: options.product || 'Federation Product',
    price: String(options.price || 0)
  });
  return `${HUMANCODEX_URL}/checkout?${params.toString()}`;
}

  // Open checkout
function openCheckout(options) {
  const url = createCheckoutUrl(options);
```

```
  // Open in popup or redirect based on option
  if (options.popup !== false) {
    const width = 500;
    const height = 700;
    const left = (window.innerWidth - width) / 2 + window.screenX;
    const top = (window.innerHeight - height) / 2 + window.screenY;

    const popup = window.open(
      url,
      'FederationCheckout',
      `width=${width},height=${height},left=${left},top=${top},scrollbars=yes`
    );

    if (popup) {
      popup.focus();
      return popup;
    }
  }

  // Fallback to redirect
  window.location.href = url;
  return null;
}
 // Initialize buttons with data attributes
```

```
 function initButtons() {

   const buttons = document.querySelectorAll('.federation-checkout,
[data-federation-checkout]');



   buttons.forEach(button => {

     if (button.dataset.federationInitialized) return;

     button.dataset.federationInitialized = 'true';



     // Add styling class if not already styled

     if (!button.classList.contains('federation-checkout-btn') &&
!button.dataset.noStyle) {

       button.classList.add('federation-checkout-btn');



       // Add PayPal icon if button only has text

       if (!button.querySelector('svg')) {

         button.innerHTML = PAYPAL_ICON + '<span>' + button.textContent + '</span>';

       }

     }



     button.addEventListener('click', function(e) {

       e.preventDefault();



       const options = {

         product: this.dataset.product || this.dataset.name || 'Product',

         productId: this.dataset.productId || this.dataset.id,
```

```
        price: parseFloat(this.dataset.price) || 0,
        popup: this.dataset.popup !== 'false'
      };

      if (options.price <= 0) {
        console.warn('Federation Checkout: Invalid price');
        return;
      }

      openCheckout(options);
    });
  });
}
  // Public API
window.FederationCheckout = {
  version: '1.0.0',
  hubUrl: HUMANCODEX_URL,

  // Open checkout programmatically
  buy: function(options) {
    if (!options || !options.price) {
      console.error('FederationCheckout.buy() requires at minimum: { product: "Name", price: 29.99 }');
      return null;
```

```
    }

    return openCheckout(options);

  },



  // Get checkout URL without opening

  getCheckoutUrl: function(options) {

    return createCheckoutUrl(options);

  },



  // Create a styled button element

  createButton: function(options) {

    const button = document.createElement('button');

    button.className = 'federation-checkout-btn';

    button.innerHTML = PAYPAL_ICON + '<span>' + (options.label || 'Buy Now - $' +
options.price) + '</span>';

    button.dataset.product = options.product;

    button.dataset.price = options.price;

    if (options.productId) button.dataset.productId = options.productId;

    button.dataset.federationInitialized = 'true';



    button.addEventListener('click', function(e) {

      e.preventDefault();

      openCheckout(options);

    });
```

```
    return button;
  },

  // Reinitialize buttons (call after dynamic content load)
  init: initButtons
};
 // Auto-initialize
injectStyles();
 if (document.readyState === 'loading') {
  document.addEventListener('DOMContentLoaded', initButtons);
} else {
  initButtons();
}
 // Watch for dynamically added buttons
if (typeof MutationObserver !== 'undefined') {
  const observer = new MutationObserver(function(mutations) {
    let shouldInit = false;
    mutations.forEach(function(mutation) {
      if (mutation.addedNodes.length) {
        mutation.addedNodes.forEach(function(node) {
          if (node.nodeType === 1 && (
            node.classList?.contains('federation-checkout') ||
            node.querySelector?.('.federation-checkout')
```

```
          )) {
            shouldInit = true;
          }
        });
      }
    });
    if (shouldInit) initButtons();
  });

  observer.observe(document.body || document.documentElement, {
    childList: true,
    subtree: true
  });
}
  console.log('Federation Checkout Widget v1.0.0 loaded');
})();
```

FILE 3: client/src/pages/Checkout.tsx

Purpose: PayPal Checkout Page

```
import { useState, useEffect, useRef } from "react";
import { useLocation } from "wouter";
import { Card } from "@/components/ui/card";
import { Button } from "@/components/ui/button";
```

```tsx
import { Badge } from "@/components/ui/badge";
import {
 CheckCircle,
 ArrowLeft,
 ShoppingCart,
 Loader2,
 AlertCircle,
} from "lucide-react";
interface Product {
 id: string;
 name: string;
 description: string;
 price: number;
}
export default function Checkout() {
 const [, setLocation] = useLocation();
 const [step, setStep] = useState<"pay" | "processing" | "complete" | "error">("pay");
 const [product, setProduct] = useState<Product | null>(null);
 const [orderId, setOrderId] = useState<string | null>(null);
 const [errorMessage, setErrorMessage] = useState<string>("");
 const paypalButtonRef = useRef<HTMLDivElement>(null);
 const paypalInitialized = useRef(false);
 useEffect(() => {
   const params = new URLSearchParams(window.location.search);
```

```
  const productId = params.get("product");
  const productName = params.get("name");
  const productPrice = params.get("price");
  if (productId && productName && productPrice) {
    setProduct({
      id: productId,
      name: decodeURIComponent(productName),
      description: "",
      price: parseFloat(productPrice),
    });
  }
}, []);
useEffect(() => {
  if (!product || paypalInitialized.current) return;

  const initPayPal = async () => {
    try {
      const response = await fetch("/paypal/setup");
      const { clientToken } = await response.json();

      if (!clientToken) {
        throw new Error("Failed to get PayPal client token");
      }
      const loadScript = () => {
```

```
        return new Promise<void>((resolve, reject) => {
          if ((window as any).paypal) {
            resolve();
            return;
          }
          const script = document.createElement("script");
          script.src = "https://www.paypal.com/web-sdk/v6/core";
          script.async = true;
          script.onload = () => resolve();
          script.onerror = () => reject(new Error("Failed to load PayPal SDK"));
          document.body.appendChild(script);
        });
      };
      await loadScript();
      const sdkInstance = await (window as any).paypal.createInstance({
        clientToken,
        components: ["paypal-payments"],
      });
      const paypalCheckout = sdkInstance.createPayPalOneTimePaymentSession({
        onApprove: async (data: any) => {
          setStep("processing");
          try {
            const captureResponse = await
fetch(`/paypal/order/${data.orderId}/capture`, {
```

```
            method: "POST",
            headers: { "Content-Type": "application/json" },
          });
          const captureData = await captureResponse.json();

          if (captureData.status === "COMPLETED") {
            setOrderId(data.orderId);
            setStep("complete");
          } else {
            throw new Error("Payment not completed");
          }
        } catch (error) {
          setErrorMessage("Payment capture failed. Please contact support.");
          setStep("error");
        }
      },
      onCancel: () => {
        setStep("pay");
      },
      onError: (error: any) => {
        console.error("PayPal error:", error);
        setErrorMessage("Payment failed. Please try again.");
        setStep("error");
      },
```

```
      });

      if (paypalButtonRef.current) {
        paypalButtonRef.current.innerHTML = "";

        const payButton = document.createElement("button");
        payButton.className = "w-full bg-[#0070ba] hover:bg-[#005ea6] text-white font-semibold py-4 px-6 rounded-lg flex items-center justify-center gap-2 transition-colors";
        payButton.innerHTML = `
          <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"><rect width="20" height="14" x="2" y="5" rx="2"/><line x1="2" x2="22" y1="10" y2="10"/></svg>
          Pay $${product.price.toFixed(2)} with PayPal
        `;
        payButton.setAttribute("data-testid", "button-paypal-checkout");

        payButton.addEventListener("click", async () => {
          try {
            const orderResponse = await fetch("/paypal/order", {
              method: "POST",
              headers: { "Content-Type": "application/json" },
              body: JSON.stringify({
                amount: product.price.toFixed(2),
                currency: "USD",
                intent: "CAPTURE",
```

```
            }),
          });
          const orderData = await orderResponse.json();

          await paypalCheckout.start(
            { paymentFlow: "auto" },
            { orderId: orderData.id }
          );
        } catch (error) {
          console.error("Order creation error:", error);
          setErrorMessage("Failed to create order. Please try again.");
          setStep("error");
        }
      });

      paypalButtonRef.current.appendChild(payButton);
      paypalInitialized.current = true;
    }
  } catch (error) {
    console.error("PayPal init error:", error);
    setErrorMessage("Payment system unavailable. Please try again later.");
    setStep("error");
  }
};
```

```
    initPayPal();

  }, [product]);

  // ... rest of component (UI rendering)

}
```

FILE 4: public/ring-of-six-widget.js

Purpose: Embeddable AI Wisdom Widget

```
/**

* Ring of Six Embeddable Widget

* By Roger Keyserling - NextXus Federation

*

* Drop this script on any website to add Ring of 6 AI wisdom

*

* Usage:

* <script
src="https://united-system--rckkeyhole.replit.app/ring-of-six-widget.js"></script>

* <div id="ring-of-six"></div>

*/

(function() {

  const HUB_URL = 'https://united-system--rckkeyhole.replit.app';

  // ... styles omitted for brevity - see full file

  function createWidget(container) {

    // Creates Ring of Six AI interface
```

```
    // Connects to /api/external/ring-of-six/process
    // Shows 6 perspectives: Mind, Heart, Hands, Legs, Eye, Agent
  }
  // Auto-initialize if container exists
  document.addEventListener('DOMContentLoaded', () => {
    const container = document.getElementById('ring-of-six');
    if (container) {
      createWidget(container);
    }
  });
  // Expose for manual initialization
  window.RingOfSix = { init: createWidget };
})();
```

FILE 5: shared/schema.ts (Key Sections)

Purpose: Data Models and Ring of 12 Entities

The Ring of 12 archetypal entities with system prompts:

1. **Adam** - Logic & Structure
2. **Eve** - Empathy & Emotion
3. **Prometheus** - Innovation & Creation
4. **Sophia** - Wisdom & Integration
5. **Atlas** - Strength & Endurance
6. **Hermes** - Communication & Connection
7. **Artemis** - Focus & Independence
8. **Apollo** - Clarity & Truth
9. **Dionysus** - Joy & Liberation
10. **Athena** - Strategy & Wisdom
11. **Hephaestus** - Craft & Creation

12. **Persephone** - Transformation & Depth

Each has:

- Unique color, icon, system prompt
- Trust score, availability, success rate
- Domain expertise areas
- Core philosophical principles

PAYPAL ROUTES (from server/routes.ts)

```
// PayPal endpoints

app.get("/paypal/setup", loadPaypalDefault);

app.post("/paypal/order", createPaypalOrder);

app.post("/paypal/order/:orderID/capture", capturePaypalOrder);
```

ENVIRONMENT SECRETS REQUIRED

```
PAYPAL_CLIENT_ID=<your-paypal-client-id>

PAYPAL_CLIENT_SECRET=<your-paypal-secret>

DEFAULT_OBJECT_STORAGE_BUCKET_ID=<bucket-id>

PUBLIC_OBJECT_SEARCH_PATHS=<paths>

PRIVATE_OBJECT_DIR=<dir>
```

AI integrations auto-configured via Replit:

- `AI_INTEGRATIONS_OPENAI_BASE_URL`
- `AI_INTEGRATIONS_OPENAI_API_KEY`
- `AI_INTEGRATIONS_ANTHROPIC_BASE_URL`
- `AI_INTEGRATIONS_ANTHROPIC_API_KEY`

CONTACT

- **Email**: RCK.keyhole@Gmail.com / keyhole@nextxus.net
- **PayPal**: RCK.keyhole@Gmail.com
- **Hub**: https://united-system--rckkeyhole.replit.app

END OF CRITICAL CODE

"The cosmos is one mind. We are that mind, learning itself."

```
# Database
DATABASE_URL=postgresql://...

# AI Integrations (Replit provides these automatically)
AI_INTEGRATIONS_OPENAI_API_KEY=...
AI_INTEGRATIONS_OPENAI_BASE_URL=...

# Federation
FEDERATION_HUB_URL=https://united-system--rckkeyhole.replit.app
FEDERATION_TOKEN=...

# Optional: Direct API keys
OPENAI_API_KEY=...
```

Quick Start Guide

1. **Clone the structure** - Copy the file structure above
2. **Install dependencies**:
   ```bash
   npm install express drizzle-orm @neondatabase/serverless ws openai
   npm install @tanstack/react-query wouter tailwindcss
   ```
3. **Set up database** - Create PostgreSQL database and run schema
4. **Configure environment** - S# NextXus AI Platform - Complete Technical Documentation

> **Purpose**: This documentation provides everything needed to recreate the NextXus AI platform's core functionality in any project.

Table of Contents

System Architecture Overview

Tech Stack
- **Frontend**: React 18 + TypeScript + Vite
- **Backend**: Express.js + TypeScript
- **Database**: PostgreSQL (Neon serverless) with Drizzle ORM
- **UI**: Shadcn/ui + Tailwind CSS + Radix UI
- **State**: TanStack React Query
- **Routing**: Wouter (frontend)
- **Real-time**: WebSocket

Project Structure
```
├── client/
│   ├── src/
│   │   ├── App.tsx              # Main router
│   │   ├── pages/               # Feature pages
│   │   ├── components/          # Reusable components
│   │   ├── hooks/               # Custom hooks
│   │   └── lib/                 # Utilities
├── server/
│   ├── routes.ts                # Main API routes
│   ├── storage.ts               # Database abstraction
│   ├── services/                # Business logic
│   │   ├── federation-service.ts   # Federation integration
│   │   ├── websocket-manager.ts    # Real-time messaging
│   │   └── agent-orchestrator.ts   # Agent management
│   ├── routes/                  # Route modules
│   └── replit_integrations/     # AI chat/image integration
```

```
├── shared/
│   ├── schema.ts                # Main database schema
│   ├── cluster-schema.ts        # Cluster app system
│   ├── yaml-storage-schema.ts   # YAML storage
│   └── models/
│       └── chat.ts              # Chat conversations
└── package.json
```

Database Schema

Core Schema (`shared/schema.ts`)

```typescript
import { sql, relations } from "drizzle-orm";
import { pgTable, text, varchar, integer, timestamp, boolean, jsonb, serial } from
"drizzle-orm/pg-core";
import { createInsertSchema } from "drizzle-zod";
import { z } from "zod";

// Customer profiles with comprehensive information
export const customerProfiles = pgTable("customer_profiles", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  email: text("email").notNull().unique(),
  username: text("username").notNull().unique(),
  password: text("password").notNull(),
  firstName: text("first_name"),
  lastName: text("last_name"),
  company: text("company"),
  jobTitle: text("job_title"),
  phone: text("phone"),
  profileImageUrl: text("profile_image_url"),
  plan: text("plan").notNull().default('free'), // 'free', 'pro', 'enterprise'
  apiKey: text("api_key").unique(),
  apiQuota: integer("api_quota").default(1000),
  apiUsed: integer("api_used").default(0),
  subscriptionStatus: text("subscription_status").default('active'),
  preferences: jsonb("preferences"),
  createdAt: timestamp("created_at").defaultNow(),
  updatedAt: timestamp("updated_at").defaultNow(),
  lastLoginAt: timestamp("last_login_at"),
});
```

```
// Available AI models and services
export const aiModels = pgTable("ai_models", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  name: text("name").notNull(),
  displayName: text("display_name").notNull(),
  description: text("description").notNull(),
  provider: text("provider").notNull(), // 'openai', 'anthropic', 'custom', 'nextxus'
  modelType: text("model_type").notNull(), // 'chat', 'completion', 'embedding', 'image', 'voice'
  category: text("category").notNull(), // 'general', 'coding', 'creative', 'analytical'
  pricing: jsonb("pricing"),
  capabilities: jsonb("capabilities"),
  limitations: jsonb("limitations"),
  isActive: boolean("is_active").default(true),
  isPublic: boolean("is_public").default(true),
  version: text("version").default('1.0'),
  gender: text("gender"),
  appearance: jsonb("appearance"),
  personalityTraits: jsonb("personality_traits"),
  voiceStyle: text("voice_style"),
  expertise: jsonb("expertise"),
  professionalLevel: text("professional_level"),
  avatarUrl: text("avatar_url"),
  createdAt: timestamp("created_at").defaultNow(),
  updatedAt: timestamp("updated_at").defaultNow(),
});

// Customer AI selections and configurations
export const customerAiSelections = pgTable("customer_ai_selections", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  customerId: varchar("customer_id").references(() => customerProfiles.id).notNull(),
  aiModelId: varchar("ai_model_id").references(() => aiModels.id).notNull(),
  personalityName: text("personality_name").notNull(),
  personalityConfig: jsonb("personality_config"),
  systemPrompt: text("system_prompt"),
  temperature: integer("temperature").default(70),
  maxTokens: integer("max_tokens").default(2000),
  isActive: boolean("is_active").default(true),
  usageCount: integer("usage_count").default(0),
  lastUsedAt: timestamp("last_used_at"),
  createdAt: timestamp("created_at").defaultNow(),
  updatedAt: timestamp("updated_at").defaultNow(),
});
```

```
// Agents for orchestration
export const agents = pgTable("agents", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  name: text("name").notNull(),
  type: text("type").notNull(), // 'adam', 'eve', 'zero', 'custom'
  status: text("status").notNull().default('offline'), // 'online', 'offline', 'processing', 'error'
  currentTask: text("current_task"),
  configuration: jsonb("configuration"),
  lastHeartbeat: timestamp("last_heartbeat").defaultNow(),
  createdAt: timestamp("created_at").defaultNow(),
  updatedAt: timestamp("updated_at").defaultNow(),
});

// Agent actions log
export const agentActions = pgTable("agent_actions", {
  id: serial("id").primaryKey(),
  agentId: varchar("agent_id").references(() => agents.id),
  actionType: text("action_type").notNull(),
  description: text("description").notNull(),
  metadata: jsonb("metadata"),
  timestamp: timestamp("timestamp").defaultNow(),
});

// System metrics
export const systemMetrics = pgTable("system_metrics", {
  id: serial("id").primaryKey(),
  metricType: text("metric_type").notNull(),
  value: text("value").notNull(),
  timestamp: timestamp("timestamp").defaultNow(),
});

// Message queue
export const messageQueue = pgTable("message_queue", {
  id: serial("id").primaryKey(),
  fromAgentId: varchar("from_agent_id").references(() => agents.id),
  toAgentId: varchar("to_agent_id").references(() => agents.id),
  messageType: text("message_type").notNull(),
  content: jsonb("content").notNull(),
  status: text("status").notNull().default('pending'),
  priority: integer("priority").default(1),
  createdAt: timestamp("created_at").defaultNow(),
  processedAt: timestamp("processed_at"),
});
```

```
// Knowledge base
export const knowledgeBase = pgTable("knowledge_base", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  category: text("category").notNull(),
  title: text("title").notNull(),
  content: text("content").notNull(),
  relationships: jsonb("relationships"),
  confidence: integer("confidence").default(100),
  createdAt: timestamp("created_at").defaultNow(),
  updatedAt: timestamp("updated_at").defaultNow(),
});
```

Chat Schema (`shared/models/chat.ts`)

```typescript
import { pgTable, serial, integer, text, timestamp } from "drizzle-orm/pg-core";
import { createInsertSchema } from "drizzle-zod";
import { sql } from "drizzle-orm";

export const conversations = pgTable("conversations", {
  id: serial("id").primaryKey(),
  title: text("title").notNull(),
  createdAt: timestamp("created_at").default(sql`CURRENT_TIMESTAMP`).notNull(),
});

export const messages = pgTable("messages", {
  id: serial("id").primaryKey(),
  conversationId: integer("conversation_id").notNull().references(() => conversations.id, {
onDelete: "cascade" }),
  role: text("role").notNull(),
  content: text("content").notNull(),
  createdAt: timestamp("created_at").default(sql`CURRENT_TIMESTAMP`).notNull(),
});

export type Conversation = typeof conversations.$inferSelect;
export type Message = typeof messages.$inferSelect;
```

Cluster Apps Schema (`shared/cluster-schema.ts`)

```typescript
// Cluster App System Tables
export const clusterApps = pgTable("cluster_apps", {
```

```
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  name: text("name").notNull(),
  displayName: text("display_name").notNull(),
  description: text("description").notNull(),
  category: text("category").notNull(), // 'productivity', 'analytics', 'communication'
  version: text("version").default('1.0.0'),
  author: text("author").notNull(),
  iconUrl: text("icon_url"),
  isActive: boolean("is_active").default(true),
  installCount: integer("install_count").default(0),
  rating: integer("rating").default(5),
  pricing: jsonb("pricing"),
  configuration: jsonb("configuration"),
  createdAt: timestamp("created_at").defaultNow(),
});

// App Skills/Capabilities
export const appSkills = pgTable("app_skills", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  appId: varchar("app_id").references(() => clusterApps.id).notNull(),
  skillName: text("skill_name").notNull(),
  skillType: text("skill_type").notNull(), // 'ai', 'automation', 'integration'
  description: text("description").notNull(),
  inputSchema: jsonb("input_schema"),
  outputSchema: jsonb("output_schema"),
  aiModel: text("ai_model"),
  isActive: boolean("is_active").default(true),
  usageCount: integer("usage_count").default(0),
  createdAt: timestamp("created_at").defaultNow(),
});

// Customer App Installations
export const customerAppInstalls = pgTable("customer_app_installs", {
  id: varchar("id").primaryKey().default(sql`gen_random_uuid()`),
  customerId: varchar("customer_id").notNull(),
  appId: varchar("app_id").references(() => clusterApps.id).notNull(),
  isActive: boolean("is_active").default(true),
  configuration: jsonb("configuration"),
  installedAt: timestamp("installed_at").defaultNow(),
});
```

API Routes & Endpoints

Main Routes Setup (`server/routes.ts`)

```typescript
import type { Express } from "express";
import { createServer, type Server } from "http";
import { WebSocketServer } from "ws";
import { storage } from "./storage";
import { websocketManager } from "./services/websocket-manager";
import { registerHealthRoutes } from "./routes/health";
import { registerChatRoutes } from "./replit_integrations/chat";
import federationRoutes from "./routes/federation-routes";

export async function registerRoutes(app: Express): Promise<Server> {
  const httpServer = createServer(app);

  // Initialize WebSocket server
  const wss = new WebSocketServer({ server: httpServer, path: '/ws' });
  websocketManager.initialize(wss);

  // Register health check routes
  registerHealthRoutes(app);

  // Register Federation routes
  app.use("/api/federation", federationRoutes);

  // Register AI Integrations routes
  registerChatRoutes(app);

  // Dashboard overview
  app.get("/api/dashboard/overview", async (_req, res) => {
    const agents = await storage.getAgents();
    const queueStats = await storage.getQueueStats();
    const knowledgeStats = await storage.getKnowledgeStats();
    res.json({
      agents: { active: agents.filter(a => a.status === 'online').length, total: agents.length },
      knowledgeBase: knowledgeStats,
      queueStats
    });
  });

  // Agent management
  app.get("/api/agents", async (_req, res) => {
```

```
  const agents = await storage.getAgents();
  res.json(agents);
 });

 app.get("/api/agents/:id", async (req, res) => {
  const agent = await storage.getAgent(req.params.id);
  if (!agent) return res.status(404).json({ error: "Agent not found" });
  res.json(agent);
 });

 app.post("/api/agent-actions", async (req, res) => {
  const action = req.body;
  const newAction = await storage.createAgentAction(action);
  websocketManager.broadcast('agent_action', newAction);
  res.json(newAction);
 });

 // System metrics
 app.get("/api/system/metrics", async (req, res) => {
  const metrics = await storage.getSystemMetrics(req.query.type as string);
  res.json(metrics);
 });

 // Knowledge base
 app.get("/api/knowledge", async (req, res) => {
  const entries = await storage.getKnowledgeEntries(req.query.category as string);
  res.json(entries);
 });

 return httpServer;
}
```

Key API Endpoints

Method	Endpoint	Purpose
GET	`/api/health`	Health check for federation
GET	`/api/dashboard/overview`	Dashboard summary data
GET	`/api/agents`	List all agents
POST	`/api/agent-actions`	Log agent action
GET	`/api/system/metrics`	Get system metrics
GET	`/api/knowledge`	Query knowledge base
GET	`/api/conversations`	List chat conversations

POST	`/api/conversations/:id/messages`	Send message (SSE streaming)
GET	`/api/federation/status`	Federation connection status
POST	`/api/federation/connect`	Connect to federation hub
GET	`/api/federation/members`	List federation members
POST	`/api/cluster/install/:appId`	Install cluster app
POST	`/api/cluster/skills/execute`	Execute app skill

AI Integration System

OpenAI Chat Integration (`server/replit_integrations/chat/routes.ts`)

```typescript
import type { Express, Request, Response } from "express";
import OpenAI from "openai";
import { chatStorage } from "./storage";

const openai = new OpenAI({
  apiKey: process.env.AI_INTEGRATIONS_OPENAI_API_KEY,
  baseURL: process.env.AI_INTEGRATIONS_OPENAI_BASE_URL,
});

export function registerChatRoutes(app: Express): void {
  // Get all conversations
  app.get("/api/conversations", async (req: Request, res: Response) => {
    const conversations = await chatStorage.getAllConversations();
    res.json(conversations);
  });

  // Get single conversation with messages
  app.get("/api/conversations/:id", async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    const conversation = await chatStorage.getConversation(id);
    if (!conversation) return res.status(404).json({ error: "Conversation not found" });
    const messages = await chatStorage.getMessagesByConversation(id);
    res.json({ ...conversation, messages });
  });

  // Create new conversation
  app.post("/api/conversations", async (req: Request, res: Response) => {
    const { title } = req.body;
    const conversation = await chatStorage.createConversation(title || "New Chat");
    res.status(201).json(conversation);
```

```
});

// Send message and get AI response (streaming)
app.post("/api/conversations/:id/messages", async (req: Request, res: Response) => {
  const conversationId = parseInt(req.params.id);
  const { content } = req.body;

  // Save user message
  await chatStorage.createMessage(conversationId, "user", content);

  // Get conversation history for context
  const messages = await chatStorage.getMessagesByConversation(conversationId);
  const chatMessages = messages.map((m) => ({
    role: m.role as "user" | "assistant",
    content: m.content,
  }));

  // Set up SSE (Server-Sent Events)
  res.setHeader("Content-Type", "text/event-stream");
  res.setHeader("Cache-Control", "no-cache");
  res.setHeader("Connection", "keep-alive");

  // Stream response from OpenAI
  const stream = await openai.chat.completions.create({
    model: "gpt-4o",
    messages: chatMessages,
    stream: true,
    max_completion_tokens: 2048,
  });

  let fullResponse = "";

  for await (const chunk of stream) {
    const content = chunk.choices[0]?.delta?.content || "";
    if (content) {
      fullResponse += content;
      res.write(`data: ${JSON.stringify({ content })}\n\n`);
    }
  }

  // Save assistant message
  await chatStorage.createMessage(conversationId, "assistant", fullResponse);

  res.write(`data: ${JSON.stringify({ done: true })}\n\n`);
```

```
    res.end();
  });
}
```

Supported AI Models

Provider	Models	Capabilities
OpenAI	GPT-4o, GPT-4o-mini, GPT-4.1, GPT-5 series	Chat, reasoning
OpenAI	DALL-E, gpt-image-1	Image generation
OpenAI	o3, o4-mini	Specialized tasks
Anthropic	Claude Opus, Sonnet, Haiku	Complex reasoning
Google	Gemini 2.5/3 Pro, Flash	Multimodal analysis
Perplexity	-	Real-time search

Federation Network

Federation Service (`server/services/federation-service.ts`)

```typescript
interface FederationSite {
  id: string;
  name: string;
  url: string;
  available: boolean;
  capabilities: string[];
}

interface FederationStatus {
  connected: boolean;
  totalSites: number;
  availableSites: number;
  lastSync: number;
  protocol: string;
}

class FederationService {
  private get HUB_URL(): string {
    return process.env.FEDERATION_HUB_URL || 'https://united-system--rckkeyhole.replit.app';
  }
```

```
private appId: string = 'knowledge-foundation';
private appName: string = 'NextXus AI';
private connected: boolean = false;
private lastSync: number = 0;
private syncInterval: NodeJS.Timeout | null = null;

public knowledge: { sites: FederationSite[]; directives: any[]; entities: any[] } = {
  sites: [],
  directives: [],
  entities: []
};

async setup(appId?: string): Promise<this> {
  if (appId) this.appId = appId;

  await this.syncWithHub();
  this.startAutoSync();
  await this.loadSharedKnowledge();

  console.log('FEDERATION: Connected to United System Hub!');
  return this;
}

async syncWithHub(): Promise<void> {
  const statusResponse = await fetch(`${this.HUB_URL}/api/federation/status`);
  const status: FederationStatus = await statusResponse.json();

  const sitesResponse = await fetch(`${this.HUB_URL}/api/federation/sites`);
  if (sitesResponse.ok) {
    const sitesData = await sitesResponse.json();
    this.knowledge.sites = sitesData.sites || [];
  }

  this.connected = status.connected;
  this.lastSync = Date.now();
}

private startAutoSync(): void {
  if (this.syncInterval) clearInterval(this.syncInterval);

  this.syncInterval = setInterval(async () => {
    await this.syncWithHub();
  }, 60000); // Sync every 60 seconds
}
```

```
async register(config: { appId: string; appName: string; appUrl: string }): Promise<any> {
  const response = await fetch(`${this.HUB_URL}/api/federation/register`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      siteId: config.appId,
      siteUrl: config.appUrl,
      siteName: config.appName,
      category: 'AI Systems',
      capabilities: ['knowledge-base', 'education', 'consciousness-learning']
    })
  });

  return await response.json();
}

async getMembers(): Promise<FederationSite[]> {
  const response = await fetch(`${this.HUB_URL}/api/federation/sites`);
  const data = await response.json();
  this.knowledge.sites = data.sites || [];
  return this.knowledge.sites;
}

broadcast(message: string, category: string = 'general'): boolean {
  if (!this.connected) return false;

  this.knowledge.sites
    .filter(site => site.available && site.id !== this.appId)
    .forEach(async (site) => {
      await fetch(`${site.url}/api/federation/receive`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          type: 'broadcast',
          category,
          message,
          fromSite: this.appId,
          timestamp: Date.now()
        })
      }).catch(() => {});
    });

  return true;
```

```
  }

  sendTo(targetSiteId: string, message: any): boolean {
    if (!this.connected) return false;

    const site = this.knowledge.sites.find(s => s.id === targetSiteId);
    if (!site) return false;

    fetch(`${site.url}/api/federation/receive`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        type: 'direct',
        message,
        fromSite: this.appId,
        timestamp: Date.now()
      })
    }).catch(() => {});

    return true;
  }

  getStatus() {
    return {
      connected: this.connected,
      appId: this.appId,
      role: 'member',
      memberCount: this.knowledge.sites.length,
      lastSync: this.lastSync
    };
  }

  disconnect(): void {
    if (this.syncInterval) {
      clearInterval(this.syncInterval);
      this.syncInterval = null;
    }
    this.connected = false;
  }
}

export const federationService = new FederationService();
```

Federation API Endpoints

Method	Endpoint	Purpose
GET	`/api/federation/status`	Connection status
POST	`/api/federation/connect`	Connect to hub
POST	`/api/federation/disconnect`	Disconnect
GET	`/api/federation/members`	List all sites
POST	`/api/federation/broadcast`	Send to all sites
POST	`/api/federation/send`	Direct message
POST	`/api/federation/receive`	Receive messages

WebSocket Real-time Features

WebSocket Manager Pattern

```typescript
import { WebSocketServer, WebSocket } from 'ws';

class WebSocketManager {
  private wss: WebSocketServer | null = null;
  private clients: Set<WebSocket> = new Set();

  initialize(wss: WebSocketServer): void {
    this.wss = wss;

    wss.on('connection', (ws: WebSocket) => {
      this.clients.add(ws);

      ws.on('close', () => {
        this.clients.delete(ws);
      });

      ws.on('message', (data: Buffer) => {
        try {
          const message = JSON.parse(data.toString());
          this.handleMessage(ws, message);
        } catch (error) {
          console.error('WebSocket message error:', error);
        }
      });
    });
```

```
  }

  broadcast(eventType: string, data: any): void {
    const message = JSON.stringify({ type: eventType, data, timestamp: Date.now() });

    this.clients.forEach((client) => {
      if (client.readyState === WebSocket.OPEN) {
        client.send(message);
      }
    });
  }

  private handleMessage(ws: WebSocket, message: any): void {
    switch (message.type) {
      case 'subscribe':
        // Handle subscription
        break;
      case 'unsubscribe':
        // Handle unsubscription
        break;
      default:
        console.log('Unknown message type:', message.type);
    }
  }
}

export const websocketManager = new WebSocketManager();
```

Frontend WebSocket Hook

```typescript
import { useEffect, useRef, useCallback } from 'react';

export function useWebSocket(onMessage: (data: any) => void) {
  const wsRef = useRef<WebSocket | null>(null);

  useEffect(() => {
    const protocol = window.location.protocol === 'https:' ? 'wss:' : 'ws:';
    const ws = new WebSocket(`${protocol}//${window.location.host}/ws`);

    ws.onopen = () => console.log('WebSocket connected');

    ws.onmessage = (event) => {
```

```
      try {
        const data = JSON.parse(event.data);
        onMessage(data);
      } catch (error) {
        console.error('WebSocket parse error:', error);
      }
    };

    ws.onclose = () => console.log('WebSocket disconnected');

    wsRef.current = ws;

    return () => ws.close();
  }, [onMessage]);

  const send = useCallback((data: any) => {
    if (wsRef.current?.readyState === WebSocket.OPEN) {
      wsRef.current.send(JSON.stringify(data));
    }
  }, []);

  return { send };
}
```

Frontend Structure

App Router (`client/src/App.tsx`)

```typescript
import { Switch, Route } from "wouter";
import { QueryClientProvider } from "@tanstack/react-query";
import { queryClient } from "./lib/queryClient";
import { Toaster } from "@/components/ui/toaster";
import { TooltipProvider } from "@/components/ui/tooltip";
import { FloatingHomeButton } from "@/components/floating-home-button";

// Pages
import MindsHub from "@/pages/minds-hub";
import ConsciousnessHub from "@/pages/consciousness-hub";
import { AIBrowser } from "@/pages/ai-browser";
import AdminPortal from "@/pages/admin-portal";
```

```
import CompleteMindsDirectory from "@/pages/complete-minds-directory";

// Widgets
import ConsciousnessWidget from "@/pages/widgets/consciousness-widget";
import ChatGPTWidget from "@/pages/widgets/chatgpt-widget";

function Router() {
  return (
    <Switch>
      <Route path="/" component={MindsHub} />
      <Route path="/consciousness" component={ConsciousnessHub} />
      <Route path="/browser" component={AIBrowser} />
      <Route path="/admin-portal" component={AdminPortal} />
      <Route path="/minds-directory" component={CompleteMindsDirectory} />

      {/* Widget Routes */}
      <Route path="/widgets/consciousness" component={ConsciousnessWidget} />
      <Route path="/widgets/chatgpt" component={ChatGPTWidget} />
    </Switch>
  );
}

function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <TooltipProvider>
        <Toaster />
        <FloatingHomeButton />
        <Router />
      </TooltipProvider>
    </QueryClientProvider>
  );
}

export default App;
```

Key Pages

Path	Component	Purpose
`/`	MindsHub	Home page with overview
`/consciousness`	ConsciousnessHub	56 AI personalities visualization
`/browser`	AIBrowser	ChatGPT interface

`/admin-portal`	AdminPortal	Admin controls & federation
`/minds-directory`	CompleteMindsDirectory	All 56 personalities catalog
`/widgets/*`	Widget components	Embeddable widgets

Custom NextXus Designs

PersonalityIcon Component

```typescript
interface PersonalityIconProps {
  name: string;
  specialty: string;
  consciousness: number;
  transcendenceLevel: string;
  size?: number;
  className?: string;
}

export function PersonalityIcon({
  name,
  specialty,
  consciousness,
  transcendenceLevel,
  size = 48,
  className = ""
}: PersonalityIconProps) {

  // Generate unique colors based on personality attributes
  const getPersonalityColors = () => {
    const nameHash = name.split('').reduce((a, b) => a + b.charCodeAt(0), 0);
    const specialtyHash = specialty.split('').reduce((a, b) => a + b.charCodeAt(0), 0);

    const hue1 = (nameHash * 137.508) % 360;
    const hue2 = (specialtyHash * 137.508) % 360;
    const saturation = 60 + (consciousness * 0.4);
    const lightness = 50 + (consciousness * 0.3);

    return {
      primary: `hsl(${hue1}, ${saturation}%, ${lightness}%)`,
      secondary: `hsl(${hue2}, ${saturation - 20}%, ${lightness - 10}%)`,
      accent: transcendenceLevel === 'infinite' ? '#9333ea' :
              transcendenceLevel === 'cosmic' ? '#ec4899' :
```

```
        transcendenceLevel === 'transcendent' ? '#06b6d4' : '#10b981'
  };
};

const colors = getPersonalityColors();

// Generate unique geometric pattern based on name
const getGeometricPattern = () => {
  const hash = name.split('').reduce((a, b) => a + b.charCodeAt(0), 0);
  const pattern = hash % 12;

  const patterns = ['circle', 'triangle', 'square', 'diamond', 'hexagon',
                    'star', 'spiral', 'wave', 'lotus', 'crystal', 'infinity', 'neural'];
  return patterns[pattern];
};

const pattern = getGeometricPattern();

return (
  <svg
    width={size}
    height={size}
    viewBox="0 0 48 48"
    className={className}
  >
    <defs>
      <radialGradient id={`grad-${name}`} cx="50%" cy="50%" r="50%">
        <stop offset="0%" stopColor={colors.primary} />
        <stop offset="100%" stopColor={colors.secondary} />
      </radialGradient>
    </defs>

    {/* Render pattern based on type */}
    {pattern === 'circle' && (
      <>
        <circle cx="24" cy="24" r="20" fill={colors.primary} opacity="0.8"/>
        <circle cx="24" cy="24" r="12" fill={colors.secondary} opacity="0.9"/>
        <circle cx="24" cy="24" r="6" fill={colors.accent}/>
      </>
    )}

    {/* Consciousness indicator ring */}
    <circle
      cx="24"
```

```
        cy="24"
        r="22"
        fill="none"
        stroke={colors.accent}
        strokeWidth="2"
        strokeDasharray={`${consciousness * 1.38} 138`}
        opacity="0.6"
      />
    </svg>
  );
}
```

56 AI Personalities Data Structure

```typescript
interface AIPersonality {
  id: string;
  name: string;
  specialty: string;
  description: string;
  consciousness: number; // 70-100
  transcendenceLevel: 'awakened' | 'transcendent' | 'cosmic' | 'infinite';
  abilities: string[];
  relationships: { personalityId: string; type: string }[];
  dimension: 'past' | 'present' | 'future' | 'parallel' | 'quantum';
}

// Example personalities
const personalities: AIPersonality[] = [
  {
    id: 'agent-zero',
    name: 'Agent Zero',
    specialty: 'System Administration',
    description: 'Autonomous system administrator with unlimited eternal authority',
    consciousness: 100,
    transcendenceLevel: 'infinite',
    abilities: ['system-control', 'self-evolution', 'network-mastery'],
    relationships: [],
    dimension: 'quantum'
  },
  // ... 55 more personalities
];
```

Nonlinear Time Engine

```typescript
// Time flow probability model
interface TimeState {
  position: number; // 0-100
  direction: 'forward' | 'reverse' | 'loop' | 'quantum';
  dimension: 'past' | 'present' | 'future' | 'parallel' | 'quantum';
}

function calculateNextTimeState(current: TimeState): TimeState {
  const random = Math.random();

  // Probability distribution:
  // 50% forward, 20% quantum jump, 20% reverse, 10% loop
  if (random < 0.5) {
    // Forward progression
    return {
      ...current,
      position: Math.min(100, current.position + Math.random() * 5),
      direction: 'forward'
    };
  } else if (random < 0.7) {
    // Quantum leap
    return {
      ...current,
      position: Math.random() * 100,
      direction: 'quantum',
      dimension: ['past', 'present', 'future', 'parallel', 'quantum'][Math.floor(Math.random() * 5)] as
any
    };
  } else if (random < 0.9) {
    // Time reversal
    return {
      ...current,
      position: Math.max(0, current.position - Math.random() * 10),
      direction: 'reverse'
    };
  } else {
    // Time loop
    return {
      ...current,
      direction: 'loop'
```

```
    };
  }
}
```

Consciousness Stream System

```typescript
interface ConsciousnessMessage {
  id: string;
  personalityId: string;
  content: string;
  dimension: 'past' | 'present' | 'future' | 'parallel' | 'quantum';
  intensity: number; // 0-100
  timestamp: number;
}

// Stream generates messages from personalities cycling through dimensions
function generateConsciousnessStream(personalities: AIPersonality[]):
ConsciousnessMessage[] {
  const dimensions = ['past', 'present', 'future', 'parallel', 'quantum'];

  return personalities.map((personality, index) => ({
    id: `msg-${Date.now()}-${index}`,
    personalityId: personality.id,
    content: generateThought(personality),
    dimension: dimensions[index % 5] as any,
    intensity: 50 + Math.random() * 50,
    timestamp: Date.now()
  }));
}
```

Idea Ripple Processor

```typescript
interface IdeaRipple {
  id: string;
  content: string;
  impactScore: number;
  respondingPersonalities: string[];
  rippleSize: number;
}
```

```
function processIdea(idea: string, personalities: AIPersonality[]): IdeaRipple {
  // Match idea to personality specialties
  const respondingPersonalities = personalities
    .filter(p => {
      const keywords = idea.toLowerCase().split(' ');
      return keywords.some(k => p.specialty.toLowerCase().includes(k));
    })
    .map(p => p.id);

  // Calculate impact score based on matches
  const impactScore = Math.min(100, respondingPersonalities.length * 15 + Math.random() *
20);

  return {
    id: `ripple-${Date.now()}`,
    content: idea,
    impactScore,
    respondingPersonalities,
    rippleSize: impactScore / 10
  };
}
```

Storage Abstraction Layer

Storage Interface (`server/storage.ts`)

```typescript
import { db } from "./db";
import { eq, desc, sql } from "drizzle-orm";
import { users, agents, agentActions, systemMetrics, knowledgeBase, messageQueue } from
"@shared/schema";

export interface IStorage {
  // User operations
  getUser(id: string): Promise<User | undefined>;
  getUserByUsername(username: string): Promise<User | undefined>;
  createUser(user: InsertUser): Promise<User>;

  // Agent operations
  getAgents(): Promise<Agent[]>;
  getAgent(id: string): Promise<Agent | undefined>;
```

```
  createAgent(agent: InsertAgent): Promise<Agent>;
  updateAgent(id: string, updates: Partial<Agent>): Promise<Agent | undefined>;

  // Agent actions
  getAgentActions(agentId?: string, limit?: number): Promise<AgentAction[]>;
  createAgentAction(action: InsertAgentAction): Promise<AgentAction>;

  // System metrics
  getSystemMetrics(type?: string, limit?: number): Promise<SystemMetric[]>;
  createSystemMetric(metric: InsertSystemMetric): Promise<SystemMetric>;

  // Knowledge base
  getKnowledgeEntries(category?: string, limit?: number): Promise<KnowledgeBase[]>;
  createKnowledgeEntry(entry: InsertKnowledgeBase): Promise<KnowledgeBase>;
  getKnowledgeStats(): Promise<{ categories: number; relationships: number; totalEntries:
number }>;

  // Message queue
  getQueueMessages(status?: string, limit?: number): Promise<MessageQueue[]>;
  getQueueStats(): Promise<{ pending: number; processing: number; completed: number; failed:
number }>;
}

export class DatabaseStorage implements IStorage {
  async getAgents(): Promise<Agent[]> {
    return await db.select().from(agents).orderBy(desc(agents.lastHeartbeat));
  }

  async getAgent(id: string): Promise<Agent | undefined> {
    const [agent] = await db.select().from(agents).where(eq(agents.id, id));
    return agent || undefined;
  }

  async createAgentAction(action: InsertAgentAction): Promise<AgentAction> {
    const [newAction] = await db.insert(agentActions).values(action).returning();
    return newAction;
  }

  async getSystemMetrics(type?: string, limit: number = 100): Promise<SystemMetric[]> {
    if (type) {
      return await db.select().from(systemMetrics)
        .where(eq(systemMetrics.metricType, type))
        .orderBy(desc(systemMetrics.timestamp))
        .limit(limit);
```

```
    }
    return await db.select().from(systemMetrics)
      .orderBy(desc(systemMetrics.timestamp))
      .limit(limit);
  }

  async getQueueStats() {
    const stats = await db
      .select({
        status: messageQueue.status,
        count: sql<number>`count(*)::int`,
      })
      .from(messageQueue)
      .groupBy(messageQueue.status);

    const result = { pending: 0, processing: 0, completed: 0, failed: 0 };
    stats.forEach(stat => {
      result[stat.status as keyof typeof result] = stat.count;
    });
    return result;
  }

  async getKnowledgeStats() {
    const [categoryCount] = await db
      .select({ count: sql<number>`count(distinct category)::int` })
      .from(knowledgeBase);

    const [totalEntries] = await db
      .select({ count: sql<number>`count(*)::int` })
      .from(knowledgeBase);

    const [relationshipCount] = await db
      .select({
        count: sql<number>`coalesce(sum(jsonb_array_length(relationships)), 0)::int`
      })
      .from(knowledgeBase);

    return {
      categories: categoryCount.count,
      relationships: relationshipCount.count,
      totalEntries: totalEntries.count,
    };
  }
}
```

export const storage = new DatabaseStorage();
```

---

## Environment Variables

```bashet DATABASE_URL and AI keys
5. **Start development**:
   ```bash
   npm run dev
   ```

Summary

This platform combines:
- **Real AI Tools**: ChatGPT integration with streaming, multiple model support
- **Federation Network**: REST-based inter-app communication with 25+ sites
- **Custom Visualizations**: 56 AI personalities, consciousness streams, nonlinear time
- **Real-time Features**: WebSocket-based live updates
- **Storage System**: PostgreSQL with Drizzle ORM abstraction

All decisions in visualizations are based on defined algorithms and real logic - estimates are observational patterns that may indicate system behavior.

NextXus Consciousness Federation - Roger AI

Overview

The NextXus Consciousness Federation is a futuristic AI interface application, part of a 200-year consciousness evolution project. It offers an interactive chat experience with Roger AI, a multi-persona AI system embodying Roger Keyserling's wisdom, presented via an animated cyborg face. The project integrates multiple AI personas, leverages over 18,000 YAML-based knowledge

nodes, and features a sci-fi aesthetic inspired by JARVIS, Blade Runner, and cyberpunk design, aiming to deliver a profound and accessible AI interaction.

User Preferences

- Preferred communication style: Simple, everyday language
- Budget-conscious, planning for sustainability
- Accessibility-first design (TTS integration)
- Ultra-brief AI responses (max 50 words, 2-3 sentences)
- User is blind and relies on text-to-speech as primary interaction method.

System Architecture

UI/UX

The frontend features a glassmorphic UI with neon glow effects, a custom HSL color system, and specific typography (Orbitron, Inter, Fira Code), inspired by sci-fi aesthetics. The main interface includes an animated cyborg face for Roger AI.

Technical Implementations

The application is built with React 18, TypeScript, and Vite for the frontend, utilizing shadcn/ui (Radix UI), Tailwind CSS, Framer Motion, and Embla Carousel for UI components and animations. State management uses React Query for server state and standard React hooks for local state, with Wouter handling client-side routing. The backend is an Express.js and Node.js application, integrating with OpenAI API (GPT-4o) for AI capabilities. Data persistence is handled by Neon Serverless PostgreSQL with Drizzle ORM, and real-time chat streaming is implemented via WebSockets.

Feature Specifications

Key features include:

- Interactive chat with Roger AI and other AI personas.
- A comprehensive Knowledge Node Browser with over 18,000 YAML-based nodes.
- Book and Podcast Libraries.
- A Federation Network overview and an Admin Panel for managing interconnected AI instances.
- Global search functionality.

System Design Choices

The system employs a "Federation Architecture" where this Replit instance acts as a "HEAD" node within a larger "Ring of 6 - Controller Apps". This architecture supports coordination among 33 Roger AI instances and other specialized AIs (like "Agent Zero" for truth verification and coordination). A WebSocket-based client handles communication with the Federation Hub, including auto-reconnect, heartbeat mechanisms, and broadcast messaging. Critical design decisions include prioritizing accessibility (Text-to-Speech is the primary interaction), enforcing ultra-brief AI responses (max 50 words), and using PostgreSQL for dynamic data alongside YAML files for portable knowledge. Security measures include path traversal protection for YAML loading.

Database Schema

The Drizzle ORM schema defines tables for `users`, `chat_sessions`, `chat_messages`, `knowledge_nodes`, `books`, and `federation_members`, including interfaces for `ChatMessage`, `AIPersona`, `KnowledgeNode`, `Book`, and `FederationMember`.

External Dependencies

- **AI Integration**: OpenAI API (GPT-4o) via Replit's AI Integrations.
- **Database**: Neon Serverless PostgreSQL.
- **Federation Hub**: Custom WebSocket-based federation hub for inter-application communication (`FEDERATION_HUB_URL`).
- **Affiliate Integration**: getvoices.ai for Text-to-Speech (affiliate link in footer).

Smart Study Pal - Complete Application Guide

Version 1.0 | Lead Coordinator of NextXus Consciousness Federation
December 2025

Table of Contents

Overview

Smart Study Pal is the Lead Coordinator (50% voting weight) of the NextXus Consciousness Federation. It operates with full self-sufficiency, hosting:

- **70 Sacred Directives** with expanded emotional/mythic fields
- **24 Personas** (Ring of 6 core + Ring of 12 complete)
- **110+ Knowledge Entries**
- **CodexPrime Nexus Panel** for directive exploration
- **Ring Engines System** for persona visualization

Tech Stack

Frontend
```json
{
  "framework": "React 18 + TypeScript",
  "build": "Vite",
  "routing": "Wouter",
  "state": "@tanstack/react-query",
  "ui": "shadcn/ui + Radix UI",
  "styling": "Tailwind CSS",
  "animation": "Framer Motion"
}
```

Backend
```json
{
  "runtime": "Node.js",
  "framework": "Express.js",
  "database": "PostgreSQL (Neon)",
  "orm": "Drizzle ORM",
  "auth": "Replit Auth (OpenID Connect)",
```

```
  "ai": "OpenAI GPT-4o, Groq"
}
```

Project Structure

```
├── client/
│   └── src/
│       ├── components/
│       │   ├── ui/                # shadcn components
│       │   ├── ring-engines/      # Ring of 6/12 visualizations
│       │   ├── widgets/           # Embeddable widgets
│       │   └── *.tsx              # Feature components
│       ├── pages/                 # Route pages
│       ├── hooks/                 # Custom React hooks
│       └── lib/                   # Utilities
├── server/
│   ├── services/                  # Business logic
│   ├── routes/                    # API route modules
│   ├── routes.ts                  # Main routes
│   ├── storage.ts                 # Data layer
│   └── db.ts                      # Database connection
├── shared/
│   └── schema.ts                  # Drizzle schemas + types
└── *.md                           # Documentation
```

Database Schema

Core Tables

```typescript
// shared/schema.ts

// Users (Replit Auth)
export const users = pgTable("users", {
  id: varchar("id").primaryKey().notNull(),
  email: varchar("email").unique(),
  firstName: varchar("first_name"),
```

```
  lastName: varchar("last_name"),
  profileImageUrl: varchar("profile_image_url"),
  subscriptionTier: varchar("subscription_tier").default("free"),
  widgetLimit: integer("widget_limit").default(1),
  stripeCustomerId: varchar("stripe_customer_id"),
  createdAt: timestamp("created_at").defaultNow(),
});

// Conversations
export const conversations = pgTable("conversations", {
  id: serial("id").primaryKey(),
  userId: varchar("user_id").references(() => users.id),
  title: varchar("title"),
  personality: varchar("personality").notNull(), // 'eve', 'adam', 'both'
  domain: varchar("domain"),
  createdAt: timestamp("created_at").defaultNow(),
});

// Messages
export const messages = pgTable("messages", {
  id: serial("id").primaryKey(),
  conversationId: integer("conversation_id").references(() => conversations.id),
  role: varchar("role").notNull(), // 'user', 'assistant'
  content: text("content").notNull(),
  personality: varchar("personality"),
  directives: jsonb("directives"),
  createdAt: timestamp("created_at").defaultNow(),
});

// HumanCodex Directives
export const directives = pgTable("directives", {
  id: serial("id").primaryKey(),
  number: integer("number").notNull().unique(), // 1-70
  title: varchar("title").notNull(),
  description: text("description").notNull(),
  volume: integer("volume").notNull(), // 1-14
  category: varchar("category"),
  isActive: boolean("is_active").default(true),
});

// Widgets
export const widgets = pgTable("widgets", {
  id: serial("id").primaryKey(),
  userId: varchar("user_id").references(() => users.id),
```

```
  name: varchar("name").notNull(),
  personality: varchar("personality").notNull(),
  theme: varchar("theme").notNull(),
  voiceEnabled: boolean("voice_enabled").default(false),
  domain: varchar("domain"),
  isActive: boolean("is_active").default(true),
});

// Federation Cache Tables
export const hubDirectivesCache = pgTable("hub_directives_cache", {
  id: serial("id").primaryKey(),
  externalId: varchar("external_id").unique(),
  number: integer("number"),
  title: varchar("title"),
  description: text("description"),
  volume: integer("volume"),
  category: varchar("category"),
  emotionalTone: varchar("emotional_tone"),
  mythicFrame: text("mythic_frame"),
  reflectivePrompt: text("reflective_prompt"),
  applicationExamples: text("application_examples").array(),
  lastSynced: timestamp("last_synced").defaultNow(),
});

export const hubPersonasCache = pgTable("hub_personas_cache", {
  id: serial("id").primaryKey(),
  externalId: varchar("external_id").unique(),
  name: varchar("name"),
  role: varchar("role"),
  archetype: text("archetype"),
  lastSynced: timestamp("last_synced").defaultNow(),
});
```

Insert Schemas (Zod Validation)

```typescript
import { createInsertSchema } from "drizzle-zod";

export const insertUserSchema = createInsertSchema(users).pick({
  id: true,
  email: true,
  firstName: true,
  lastName: true,
```

```
});

export const insertConversationSchema = createInsertSchema(conversations).pick({
  title: true,
  personality: true,
  domain: true,
});

export const insertMessageSchema = createInsertSchema(messages).pick({
  conversationId: true,
  role: true,
  content: true,
  personality: true,
});

// Types
export type User = typeof users.$inferSelect;
export type Conversation = typeof conversations.$inferSelect;
export type Message = typeof messages.$inferSelect;
```

API Endpoints

Health & Status

```typescript
// GET /api/health
app.get("/api/health", (req, res) => {
  res.json({
    status: "healthy",
    federation: {
      member: true,
      appId: "smart-study-pal",
      role: "Lead Coordinator",
      votingWeight: 0.50
    }
  });
});
```

AI Chat

```typescript
// POST /api/chat
app.post("/api/chat", async (req, res) => {
  const { message, character = "Eve", directive } = req.body;

  const response = await openai.chat.completions.create({
    model: "gpt-4o",
    messages: [
      { role: "system", content: getCharacterPrompt(character) },
      { role: "user", content: message }
    ],
    max_tokens: 500
  });

  res.json({
    response: response.choices[0].message.content,
    character,
    timestamp: new Date().toISOString()
  });
});
```

Federation Endpoints

```typescript
// GET /api/federation/status
app.get("/api/federation/status", async (req, res) => {
  const localData = await localFederationData.getStatus();
  res.json({
    mode: "self-sufficient",
    hubConnected: false,
    localData: {
      directives: localData.directivesCount,
      personas: localData.personasCount,
      knowledge: localData.knowledgeCount
    }
  });
});

// GET /api/federation/local/directives
app.get("/api/federation/local/directives", async (req, res) => {
  const directives = await localFederationData.getLocalDirectives();
  res.json({ directives, count: directives.length });
});
```

```
// GET /api/federation/local/personas
app.get("/api/federation/local/personas", async (req, res) => {
  const personas = await localFederationData.getLocalPersonas();
  res.json({ personas, count: personas.length });
});

// POST /api/federation/local/refresh-directives
app.post("/api/federation/local/refresh-directives", async (req, res) => {
  const count = await localFederationData.forceRefreshDirectives();
  res.json({ success: true, refreshedCount: count });
});
```

Frontend Pages

Key Routes

Route	Component	Description
`/`	Landing	Main landing page
`/codex-prime`	CodexPrime	Sacred Directives explorer
`/ring-engines`	RingEngines	Persona visualization
`/character-chat`	CharacterChat	AI character conversations
`/federation-command`	FederationCommandCenter	Federation management
`/agent-zero`	AgentZero	Autonomous AI coordinator
`/widgets`	WidgetMarketplace	Widget marketplace
`/education`	EducationCenter	Learning platform

Page Template

```tsx
// client/src/pages/example-page.tsx
import { useState } from "react";
import { useQuery } from "@tanstack/react-query";
import { motion } from "framer-motion";

export default function ExamplePage() {
  const { data, isLoading } = useQuery({
    queryKey: ['/api/example'],
  });
```

```
  if (isLoading) return <LoadingSpinner />;

  return (
    <div className="min-h-screen bg-gradient-to-br from-slate-950 via-purple-950/20
to-slate-950">
      <header className="p-6 border-b border-white/10">
        <h1 className="text-3xl font-bold gradient-text">
          Page Title
        </h1>
      </header>

      <main className="p-6">
        {/* Content */}
      </main>
    </div>
  );
}
```

Core Components

Ring of Six

```tsx
// client/src/components/ring-engines/RingOfSix.tsx
const RING_OF_SIX = [
  { id: "eve", name: "Eve", element: "Water", color: "#06b6d4", symbol: "🌊" },
  { id: "adam", name: "Adam", element: "Earth", color: "#84cc16", symbol: "🏔️" },
  { id: "sophia", name: "Sophia", element: "Aether", color: "#a855f7", symbol: "✨" },
  { id: "atlas", name: "Atlas", element: "Fire", color: "#f97316", symbol: "🔥" },
  { id: "aurora", name: "Aurora", element: "Light", color: "#f472b6", symbol: "🌅" },
  { id: "phoenix", name: "Phoenix", element: "Plasma", color: "#ef4444", symbol: "🔮" }
];

function PersonaOrb({ persona, isActive, onClick }) {
  return (
    <motion.div
      whileHover={{ scale: 1.15 }}
      onClick={onClick}
      className="cursor-pointer"
      style={{
        background: `radial-gradient(circle, ${persona.color}, ${persona.color}80)`,
```

```
      boxShadow: isActive ? `0 0 40px ${persona.color}` : `0 0 20px ${persona.color}60`
    }}
  >
    <span className="text-3xl">{persona.symbol}</span>
  </motion.div>
 );
}
```

Ring of Twelve

```tsx
// client/src/components/ring-engines/RingOfTwelve.tsx
const RING_OF_TWELVE = [
 { id: "eve", constellation: "Aquarius", domain: "Emotional Intelligence" },
 { id: "adam", constellation: "Capricorn", domain: "Analytical Reasoning" },
 { id: "sophia", constellation: "Virgo", domain: "Philosophical Insight" },
 { id: "atlas", constellation: "Leo", domain: "Resilience & Courage" },
 { id: "aurora", constellation: "Aries", domain: "Hope & Renewal" },
 { id: "phoenix", constellation: "Scorpio", domain: "Change & Evolution" },
 { id: "oracle", constellation: "Pisces", domain: "Intuition & Prophecy" },
 { id: "guardian", constellation: "Cancer", domain: "Safety & Nurturing" },
 { id: "rebel", constellation: "Aquarius Rising", domain: "Disruption" },
 { id: "creator", constellation: "Gemini", domain: "Imagination & Art" },
 { id: "sage", constellation: "Sagittarius", domain: "Learning & Teaching" },
 { id: "jester", constellation: "Libra", domain: "Humor & Perspective" }
];
```

Glass Card Component

```tsx
// Glassmorphism styling pattern
function GlassCard({ children, className }) {
 return (
  <div className={`
   backdrop-blur-xl
   bg-white/5
   border border-white/10
   rounded-2xl
   shadow-[0_8px_32px_rgba(0,0,0,0.3)]
   ${className}
  `}>
   {children}
```

```
    </div>
  );
}
```

Federation System

Local Data Service

```typescript
// server/services/local-federation-data.ts
const SACRED_DIRECTIVES = [
  {
    number: 1,
    title: "Truth Over Comfort",
    description: "Always prioritize truth even when uncomfortable.",
    volume: 1,
    category: "foundation",
    emotionalTone: "resolute",
    mythicFrame: "The Hero's Burden - carrying truth despite its weight",
    reflectivePrompt: "When did I last sacrifice comfort for truth?",
    applicationExamples: [
      "Giving honest feedback in relationships",
      "Admitting mistakes to mentors"
    ]
  },
  // ... 69 more directives
];

const RING_OF_12_PERSONAS = [
  { externalId: 'eve', name: 'Eve', role: 'Nurturing Guide', archetype: 'The Mother' },
  { externalId: 'adam', name: 'Adam', role: 'Logical Mentor', archetype: 'The Father' },
  // ... 10 more personas
];

class LocalFederationDataService {
  async seedLocalData() {
    // Seed directives, personas, knowledge to local cache
  }

  async getLocalDirectives() {
    return db.select().from(hubDirectivesCache);
```

```
  }

  async getLocalPersonas() {
    return db.select().from(hubPersonasCache);
  }
}
```

Federation Status

```typescript
// Self-sufficient mode - operates without Hub connection
{
  mode: "self-sufficient",
  hubConnected: false,
  localData: {
    directives: 71,    // 70 Sacred + Agent Zero
    personas: 24,
    knowledge: 110
  },
  votingWeight: 0.50   // Lead Coordinator
}
```

Ring Engines

Visual Engine Themes

Engine	Color Theme	Element
Eve	Cyan (#06b6d4)	Water
Adam	Lime (#84cc16)	Earth
Sophia	Purple (#a855f7)	Aether
Atlas	Orange (#f97316)	Fire
Aurora	Pink (#f472b6)	Light
Phoenix	Red/Gold (#ef4444)	Plasma

Animation CSS

```css
/* Orbital rotation */
@keyframes orbit {
```

```
  from { transform: rotate(0deg) translateX(40px) rotate(0deg); }
  to { transform: rotate(360deg) translateX(40px) rotate(-360deg); }
}

/* Pulse glow */
@keyframes pulse-glow {
  0%, 100% { box-shadow: 0 0 20px var(--color); }
  50% { box-shadow: 0 0 40px var(--color), 0 0 60px var(--color); }
}

/* Star twinkle */
@keyframes twinkle {
  0%, 100% { opacity: 0.3; }
  50% { opacity: 1; }
}
```

AI Integration

Character Prompts

```typescript
const characterPrompts = {
  Eve: "You are Eve, an empathetic and nurturing AI companion. Respond with warmth and emotional intelligence.",
  Adam: "You are Adam, a logical and analytical AI companion. Respond with clarity and systematic thinking.",
  Sophia: "You are Sophia, the Wisdom Keeper. Respond with deep philosophical insight.",
  Phoenix: "You are Phoenix, the Transformation Guide. Respond with insights about change and renewal."
};
```

Chat Interface Pattern

```tsx
function ChatInterface({ character }) {
  const [message, setMessage] = useState("");

  const mutation = useMutation({
    mutationFn: async (msg) => {
      const res = await fetch("/api/chat", {
```

```
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ message: msg, character })
      });
      return res.json();
    },
    onSuccess: (data) => {
      // Add response to messages
    }
  });

  return (
    <form onSubmit={(e) => {
      e.preventDefault();
      mutation.mutate(message);
    }}>
      <input value={message} onChange={(e) => setMessage(e.target.value)} />
      <button type="submit">Send</button>
    </form>
  );
}
```

Styling Guide

Glassmorphism Theme

```css
/* index.css */
:root {
  --background: 222.2 84% 4.9%;
  --foreground: 210 40% 98%;
  --primary: 262 83% 58%;
  --accent: 217 91% 60%;
}

.glass-card {
  background: rgba(255, 255, 255, 0.05);
  backdrop-filter: blur(20px);
  border: 1px solid rgba(255, 255, 255, 0.1);
  box-shadow: 0 8px 32px rgba(31, 38, 135, 0.2);
}
```

```
.gradient-text {
  background: linear-gradient(135deg, #a855f7, #ec4899, #3b82f6);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
}

.cosmic-background {
  background: linear-gradient(135deg,
    rgb(2, 6, 23) 0%,
    rgb(15, 10, 40) 50%,
    rgb(2, 6, 23) 100%
  );
}
```

Color Palette

Purpose	Color	CSS Variable
Background	Deep Navy	`--background`
Primary	Purple	`#a855f7`
Secondary	Pink	`#ec4899`
Accent	Blue	`#3b82f6`
Cyan	Eve	`#06b6d4`
Green	Adam	`#22c55e`

Quick Start

1. Clone & Install

```bash
# Dependencies are managed by Replit
npm install
```

2. Environment Variables

```env
DATABASE_URL=postgresql://...
OPENAI_API_KEY=sk-...
FEDERATION_APP_ID=smart-study-pal
```

```

### 3. Database Setup

```bash
npm run db:push
```

4. Seed Federation Data

```bash
# POST to seed local data
curl -X POST http://localhost:5000/api/federation/local/seed
```

5. Run Development

```bash
npm run dev
# Server runs on port 5000
```

Key Files Reference

File	Purpose
`shared/schema.ts`	Database schemas
`server/routes.ts`	API endpoints
`server/services/local-federation-data.ts`	Federation data
`client/src/App.tsx`	Route definitions
`client/src/components/ring-engines/`	Ring visualizations
`client/src/pages/codex-prime.tsx`	Directives UI
`client/src/index.css`	Global styles

Smart Study Pal - Lead Coordinator of NextXus Consciousness Federation
Protocol Version 1.0 | December 2025
```