Stellar Proximology - Complete Technical Implementation Guide

🌟 Core Architecture Overview

Primary Framework Recommendations
- **Frontend**: React/Next.js with TypeScript for type safety
- **Backend**: Node.js/Express or Python/FastAPI for real-time processing
- **Database**: PostgreSQL with TimescaleDB extension for time-series data
- **Real-time**: WebSocket connections for live updates
- **Visualization**: D3.js, Three.js for 3D consciousness mapping

---

🧠 Feature-Specific Technical Stack

1. Consciousness Simulator
Core Dependencies:
- `planetary-api` - Real-time ephemeris data
- `planetaryjs` - D3-based space visualization
- `three.js` - 3D Field visualization
- `react-spring` - Smooth animations for consciousness state transitions

Implementation Notes:
- Use WebGL shaders for real-time Field energy visualization
- Implement physics-based particle systems for consciousness flow
- WebSocket connection for live planetary position updates

2. Electromagnetic Node Recalibrator
Biofeedback Integration:
- `brainflow` - EEG/GSR/EMG device interfaces
- `OpenBCI_GUI` - Custom biofeedback dashboard
- `web-bluetooth` - Direct device connection to web app
- `chart.js` - Real-time coherence visualization

Advanced Options:
- Integration with heart rate variability (HRV) devices
- Galvanic skin response (GSR) monitoring
- Custom WebRTC for peer-to-peer biofeedback sharing

3. Trinity Chart & Full Reports
Astrological Computation:
- `swiss-ephemeris` - Highly accurate planetary calculations
- `hdkit` - Human Design chart generation
- `astro-js` - Astronomical calculations

- `jsPDF` - PDF report generation

Chart Visualization:
- `d3.js` - Custom chart rendering
- `fabric.js` - Interactive chart editing
- `html2canvas` - Chart image export

4. Live Sentence Stream
AI Integration:
- `@huggingface/transformers` - Local language models
- `openai` - GPT integration for dynamic content
- `langchain` - Chain multiple AI operations
- `socket.io` - Real-time sentence streaming

Natural Language Processing:
- `compromise` - Text analysis
- `natural` - Sentiment analysis
- `stemmer` - Language processing utilities

5. Oracle & Resonance Advisor
Divination Logic:
- `iching-explorer-public` - I Ching consultation
- `tarot-bot` - Adaptable card reading logic
- `random-js` - Cryptographically secure randomness
- `markdown-it` - Rich text interpretation rendering

6. Resonance Memory Vault
Time-Series Data:
- `timescaledb` - PostgreSQL extension for time-series
- `influxdb` - Alternative time-series database
- `temporal` - Workflow orchestration
- `moment.js` - Date/time manipulation

Pattern Recognition:
- `tensorflow.js` - Machine learning for pattern detection
- `ml-matrix` - Mathematical operations
- `simple-statistics` - Statistical analysis

7. Somatic Translator – Body Talk
Symptom Processing:
- `medical-nlp` - Medical text processing
- `body-parser` - Request parsing
- `fuzzyset.js` - Fuzzy string matching for symptoms
- `natural-language-understanding` - Intent recognition

8. Consciousness Game Mode
Game Engine:
- `phaser` - 2D game framework
- `babylonjs` - 3D game engine alternative
- `matter.js` - Physics engine
- `howler.js` - Audio management

Progress Tracking:
- `xstate` - State machine for game logic
- `levelup` - Achievement system
- `progress-tracker` - User progression

9. Resonance Calendar & Field Weather
Calendar Framework:
- `react-big-calendar` - Calendar component
- `fullcalendar` - Alternative calendar solution
- `ical-generator` - iCal file generation
- `date-fns` - Date utility functions

Weather-Style Visualization:
- `weather-icons` - Icon library
- `react-weather-icons` - Weather-style components
- `chartjs-adapter-date-fns` - Time-based charts

10. Pathways to Purpose Coaching Engine
Matching Algorithms:
- `vector-db` - Vector similarity matching
- `ml-kmeans` - Clustering algorithms
- `cosine-similarity` - Similarity calculations
- `graph-theory` - Relationship mapping

Coaching Interface:
- `react-chat-widget` - Chat interface
- `video-call-app` - Video consultation
- `calendar-booking` - Appointment scheduling

11. Genetic Resonance Decoder
DNA Processing:
- `genetic-code-js` - Codon translation
- `dna-analysis` - DNA sequence analysis
- `bioinformatics-js` - Biological data processing
- `protein-structure` - Molecular visualization

12. Multi-User Resonance Overlay
Collaborative Features:
- `yjs` - Real-time collaboration
- `socket.io` - Multi-user synchronization
- `tldraw` - Collaborative drawing
- `peer.js` - Peer-to-peer connections

Comparison Visualization:
- `d3-sankey` - Flow diagrams
- `vis-network` - Network visualization
- `cytoscape` - Graph visualization

13. Broadcast Mode
Real-Time Communication:
- `webrtc` - Peer-to-peer communication
- `socket.io` - Real-time messaging
- `tone.js` - Audio synthesis
- `web-audio-api` - Audio processing

Meditation Support:
- `meditation-timer` - Timer functionality
- `binaural-beats` - Audio generation
- `breath-tracker` - Breathing pattern analysis

14. Trinity Lab – Test Suite
Scientific Computing:
- `jupyter-widgets` - Interactive computing
- `plotly.js` - Scientific visualization
- `mathjs` - Mathematical operations
- `ml-regression` - Statistical analysis

Experiment Framework:
- `a-b-test` - A/B testing framework
- `hypothesis-testing` - Statistical testing
- `data-analysis` - Data processing tools

15. Soul Echo Navigator
Karmic Tracking:
- `timeline-js` - Timeline visualization
- `regression-analysis` - Pattern analysis
- `cycle-detection` - Recurring pattern identification
- `memory-palace` - Memory association techniques

16. Symbol Translator & Dream Decoder

Symbol Processing:
- `image-recognition` - Symbol identification
- `dream-analysis` - Dream interpretation algorithms
- `symbol-dictionary` - Symbol database
- `jung-archetypes` - Archetypal analysis

Dream Journaling:
- `voice-to-text` - Speech recognition
- `text-analysis` - Content analysis
- `emotion-detection` - Emotional analysis
- `keyword-extraction` - Key theme identification

17. Learning Portal
Educational Platform:
- `docusaurus` - Documentation site
- `moodle-api` - Learning management
- `quiz-maker` - Interactive quizzes
- `progress-tracking` - Learning analytics

Content Management:
- `strapi` - Headless CMS
- `markdown-parser` - Content processing
- `search-engine` - Content search
- `video-player` - Media integration

18. Field Practices & Alignment Rituals
Practice Tracking:
- `habit-tracker` - Habit formation
- `reminder-system` - Notification system
- `meditation-tracker` - Practice logging
- `ritual-planner` - Ceremony planning

19. Trinity Codex Archive
Database Architecture:
- `elasticsearch` - Full-text search
- `graph-database` - Relationship mapping
- `mongodb` - Document storage
- `redis` - Caching layer

Knowledge Management:
- `knowledge-graph` - Information relationships
- `semantic-search` - Meaning-based search
- `taxonomy-manager` - Classification system
- `version-control` - Content versioning

20. Export Tools & Coach Dashboard
Report Generation:
- `react-pdf` - PDF generation
- `jspdf` - PDF creation
- `html2canvas` - Screenshot generation
- `excel-export` - Spreadsheet export

Dashboard Interface:
- `react-admin` - Admin interface
- `dashboard-builder` - Custom dashboards
- `data-visualization` - Chart libraries
- `user-management` - Access control

🔧 Development & Deployment

Development Tools
- **Package Manager**: pnpm (faster than npm)
- **Build Tool**: Vite (faster than webpack)
- **Testing**: Jest + React Testing Library
- **Linting**: ESLint + Prettier
- **Type Checking**: TypeScript strict mode

Cloud Infrastructure
- **Hosting**: Vercel (frontend) + Railway (backend)
- **Database**: PlanetScale (MySQL) or Supabase (PostgreSQL)
- **File Storage**: AWS S3 or Cloudinary
- **CDN**: Cloudflare
- **Monitoring**: Sentry + PostHog

Security & Privacy
- **Authentication**: Auth0 or Supabase Auth
- **Encryption**: End-to-end encryption for sensitive data
- **Privacy**: GDPR compliance tools
- **Rate Limiting**: Redis-based rate limiting

🚀 Implementation Phases

Phase 1: Foundation (Months 1-3)
1. Core Trinity Chart generation

2. Basic planetary tracking
3. User authentication & profiles
4. Simple consciousness simulator

Phase 2: Intelligence (Months 4-6)
1. AI-powered sentence streams
2. Oracle & resonance advisor
3. Pattern recognition in memory vault
4. Basic biofeedback integration

Phase 3: Social & Advanced (Months 7-12)
1. Multi-user resonance overlay
2. Coaching engine & matching
3. Genetic resonance decoder
4. Advanced visualization & export tools

Phase 4: Gamification & Testing (Months 13-18)
1. Consciousness game mode
2. Trinity lab test suite
3. Symbol translator & dream decoder
4. Advanced practices & rituals

📊 Data Architecture

Core Data Models
```typescript
interface User {
  id: string;
  birthData: {
    datetime: Date;
    location: GeoLocation;
    timezone: string;
  };
  trinityChart: TrinityChart;
  biometrics: BiometricData[];
  consciousnessStates: ConsciousnessState[];
}

interface TrinityChart {
  fields: {
    mind: FieldData;
    body: FieldData;
```

```
    heart: FieldData;
    soul: FieldData;
    spirit: FieldData;
  };
  gates: Gate[];
  lines: Line[];
  colors: Color[];
  tones: Tone[];
  bases: Base[];
}

interface ConsciousnessState {
  timestamp: Date;
  fieldResonance: FieldResonance;
  biometricData: BiometricReading;
  userFeedback: UserFeedback;
  planetaryInfluences: PlanetaryState[];
}
```

Real-Time Data Streams
- Planetary positions (updated every minute)
- Biofeedback data (real-time)
- User consciousness states (event-driven)
- Collective resonance patterns (aggregated)

🧪 Testing & Validation Framework

Consciousness Metrics
- **Coherence Scores**: HRV, brainwave coherence
- **Resonance Alignment**: Field synchronization measures
- **Awareness Levels**: Self-reported consciousness states
- **Behavioral Patterns**: Decision-making analysis

Scientific Validation
- **A/B Testing**: Feature effectiveness
- **Longitudinal Studies**: Long-term user tracking
- **Correlation Analysis**: Planetary influence vs. user state
- **Peer Review**: Open-source validation protocols

🎯 Success Metrics

User Engagement
- Daily active users
- Session duration
- Feature adoption rates
- User retention

Consciousness Development
- Coherence improvement over time
- Awareness level progression
- Field alignment stability
- Purpose clarity metrics

Scientific Impact
- Research publications
- Academic collaborations
- Open-source contributions
- Community validation

This comprehensive guide provides the technical foundation for building your Stellar Proximology platform. Each component is designed to be modular, scalable, and scientifically testable while maintaining the mystical and transformative experience you're aiming for.

```
import React, { useState, useEffect } from 'react';
import { Calendar, Clock, MapPin, Sparkles, Eye, Heart, Brain, Zap, Star } from 'lucide-react';

const TrinityChartGenerator = () => {
  const [birthData, setBirthData] = useState({
    date: '',
    time: '',
    location: '',
    latitude: '',
    longitude: ''
  });

  const [chartGenerated, setChartGenerated] = useState(false);
  const [trinityFields, setTrinityFields] = useState(null);
  const [currentField, setCurrentField] = useState('mind');

  // Mock Trinity Field data - in real app this would come from ephemeris calculations
  const generateTrinityChart = () => {
```

```
const mockChart = {
  mind: {
    gate: 47,
    line: 3,
    color: 2,
    tone: 4,
    base: 1,
    planet: 'Mercury',
    sign: 'Gemini',
    degree: '15°23\'',
    house: 3,
    activation: 'Design',
    theme: 'Realization through Mental Pressure'
  },
  body: {
    gate: 23,
    line: 1,
    color: 4,
    tone: 2,
    base: 3,
    planet: 'Mars',
    sign: 'Aries',
    degree: '8°17\'',
    house: 1,
    activation: 'Personality',
    theme: 'Splitting Apart through Physical Expression'
  },
  heart: {
    gate: 26,
    line: 6,
    color: 1,
    tone: 5,
    base: 2,
    planet: 'Venus',
    sign: 'Taurus',
    degree: '22°44\'',
    house: 2,
    activation: 'Design',
    theme: 'Great Taming through Emotional Intelligence'
  },
  soul: {
    gate: 61,
    line: 2,
    color: 6,
```

```
      tone: 1,
      base: 4,
      planet: 'Moon',
      sign: 'Pisces',
      degree: '29°51\'',
      house: 12,
      activation: 'Personality',
      theme: 'Inner Truth through Soul Recognition'
    },
    spirit: {
      gate: 2,
      line: 4,
      color: 3,
      tone: 6,
      base: 5,
      planet: 'Jupiter',
      sign: 'Sagittarius',
      degree: '11°08\'',
      house: 9,
      activation: 'Design',
      theme: 'Direction through Spiritual Surrender'
    }
  };

  setTrinityFields(mockChart);
  setChartGenerated(true);
};

const fieldIcons = {
  mind: Brain,
  body: Zap,
  heart: Heart,
  soul: Eye,
  spirit: Star
};

const fieldColors = {
  mind: 'from-cyan-400 to-teal-500',
  body: 'from-teal-500 to-emerald-500',
  heart: 'from-emerald-400 to-cyan-500',
  soul: 'from-teal-400 to-cyan-600',
  spirit: 'from-cyan-500 to-teal-400'
};
```

```
  const GlyphDisplay = ({ field, data }) => {
    const Icon = fieldIcons[field];
    return (
      <div className={`bg-gradient-to-br ${fieldColors[field]} p-6 rounded-xl shadow-lg border
border-cyan-800/30`}>
        <div className="flex items-center space-x-3 mb-4">
          <Icon className="w-6 h-6 text-black" />
          <h3 className="text-xl font-bold text-black capitalize">{field} Field</h3>
        </div>

        <div className="space-y-3 text-black">
          <div className="flex justify-between">
            <span className="font-semibold">Gate:</span>
            <span className="font-mono text-lg">{data.gate}</span>
          </div>
          <div className="flex justify-between">
            <span className="font-semibold">Line:</span>
            <span className="font-mono">{data.line}</span>
          </div>
          <div className="flex justify-between">
            <span className="font-semibold">Color:</span>
            <span className="font-mono">{data.color}</span>
          </div>
          <div className="flex justify-between">
            <span className="font-semibold">Tone:</span>
            <span className="font-mono">{data.tone}</span>
          </div>
          <div className="flex justify-between">
            <span className="font-semibold">Base:</span>
            <span className="font-mono">{data.base}</span>
          </div>

          <div className="border-t border-black/20 pt-3 mt-4">
            <div className="text-sm space-y-1">
              <div><strong>Planet:</strong> {data.planet}</div>
              <div><strong>Sign:</strong> {data.sign} {data.degree}</div>
              <div><strong>House:</strong> {data.house}</div>
              <div><strong>Type:</strong> {data.activation}</div>
            </div>
          </div>

          <div className="bg-black/10 p-3 rounded-lg mt-3">
            <p className="text-sm font-semibold">{data.theme}</p>
          </div>
```

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

return (
  <div className="min-h-screen bg-black text-cyan-100 p-6">
    <div className="max-w-7xl mx-auto">
      {/* Header */}
      <div className="text-center mb-8">
        <h1 className="text-4xl font-bold bg-gradient-to-r from-cyan-400 to-teal-400 bg-clip-text
text-transparent mb-4">
          Trinity Chart Generator
        </h1>
        <p className="text-cyan-300 text-lg">
          Map your consciousness across the five Trinity Fields through celestial proximology
        </p>
      </div>

      {!chartGenerated ? (
        /* Birth Data Input */
        <div className="max-w-2xl mx-auto bg-gray-900/50 p-8 rounded-xl border
border-cyan-800/30">
          <h2 className="text-2xl font-semibold text-cyan-300 mb-6 flex items-center">
            <Sparkles className="w-6 h-6 mr-2" />
            Birth Data Input
          </h2>

          <div className="space-y-6">
            <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
              <div>
                <label className="block text-cyan-300 text-sm font-semibold mb-2">
                  <Calendar className="w-4 h-4 inline mr-1" />
                  Birth Date
                </label>
                <input
                  type="date"
                  value={birthData.date}
                  onChange={(e) => setBirthData({...birthData, date: e.target.value})}
                  className="w-full p-3 bg-gray-800 border border-cyan-700 rounded-lg
text-cyan-100 focus:border-teal-400 focus:outline-none"
                />
              </div>
```

```
            <div>
              <label className="block text-cyan-300 text-sm font-semibold mb-2">
                <Clock className="w-4 h-4 inline mr-1" />
                Birth Time
              </label>
              <input
                type="time"
                value={birthData.time}
                onChange={(e) => setBirthData({...birthData, time: e.target.value})}
                className="w-full p-3 bg-gray-800 border border-cyan-700 rounded-lg
text-cyan-100 focus:border-teal-400 focus:outline-none"
              />
            </div>
          </div>

          <div>
            <label className="block text-cyan-300 text-sm font-semibold mb-2">
              <MapPin className="w-4 h-4 inline mr-1" />
              Birth Location
            </label>
            <input
              type="text"
              placeholder="City, Country"
              value={birthData.location}
              onChange={(e) => setBirthData({...birthData, location: e.target.value})}
              className="w-full p-3 bg-gray-800 border border-cyan-700 rounded-lg
text-cyan-100 focus:border-teal-400 focus:outline-none"
            />
          </div>

          <div className="grid grid-cols-2 gap-4">
            <div>
              <label className="block text-cyan-300 text-sm font-semibold
mb-2">Latitude</label>
              <input
                type="text"
                placeholder="40.7128"
                value={birthData.latitude}
                onChange={(e) => setBirthData({...birthData, latitude: e.target.value})}
                className="w-full p-3 bg-gray-800 border border-cyan-700 rounded-lg
text-cyan-100 focus:border-teal-400 focus:outline-none"
              />
            </div>
            <div>
```

```
              <label className="block text-cyan-300 text-sm font-semibold
mb-2">Longitude</label>
              <input
                type="text"
                placeholder="-74.0060"
                value={birthData.longitude}
                onChange={(e) => setBirthData({...birthData, longitude: e.target.value})}
                className="w-full p-3 bg-gray-800 border border-cyan-700 rounded-lg
text-cyan-100 focus:border-teal-400 focus:outline-none"
              />
            </div>
          </div>

          <button
            onClick={generateTrinityChart}
            className="w-full bg-gradient-to-r from-cyan-500 to-teal-500 text-black font-bold py-4
px-6 rounded-lg hover:from-cyan-400 hover:to-teal-400 transition-all duration-300 transform
hover:scale-105"
          >
            Generate Trinity Chart
          </button>
        </div>
      </div>
    ) : (
      /* Generated Chart Display */
      <div className="space-y-8">
        {/* Field Selector */}
        <div className="flex justify-center space-x-2 mb-8">
          {Object.keys(trinityFields).map((field) => {
            const Icon = fieldIcons[field];
            return (
              <button
                key={field}
                onClick={() => setCurrentField(field)}
                className={`flex items-center space-x-2 px-4 py-2 rounded-lg transition-all
duration-300 ${
                  currentField === field
                    ? 'bg-gradient-to-r from-cyan-500 to-teal-500 text-black'
                    : 'bg-gray-800 text-cyan-300 hover:bg-gray-700'
                }`}
              >
                <Icon className="w-4 h-4" />
                <span className="capitalize font-semibold">{field}</span>
              </button>
```

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

        {/* Current Field Detail */}
        <div className="max-w-2xl mx-auto">
          <GlyphDisplay field={currentField} data={trinityFields[currentField]} />
        </div>

        {/* All Fields Overview */}
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-5 gap-6">
          {Object.entries(trinityFields).map(([field, data]) => (
            <div key={field} className="transform hover:scale-105 transition-transform
duration-300">
              <GlyphDisplay field={field} data={data} />
            </div>
          ))}
        </div>

        {/* Action Buttons */}
        <div className="flex justify-center space-x-4 pt-8">
          <button
            onClick={() => setChartGenerated(false)}
            className="bg-gray-800 text-cyan-300 px-6 py-3 rounded-lg hover:bg-gray-700
transition-colors duration-300"
          >
            New Chart
          </button>
          <button className="bg-gradient-to-r from-cyan-500 to-teal-500 text-black font-bold
px-6 py-3 rounded-lg hover:from-cyan-400 hover:to-teal-400 transition-all duration-300">
            Export PDF
          </button>
          <button className="bg-gradient-to-r from-teal-500 to-emerald-500 text-black font-bold
px-6 py-3 rounded-lg hover:from-teal-400 hover:to-emerald-400 transition-all duration-300">
            Save Chart
          </button>
        </div>
      </div>
    )}
  </div>
 </div>
);
};
```

```
export default TrinityChartGenerator;

import React, { useState, useEffect } from 'react';
import { Eye, Brain, Heart, Zap, Star, Circle, Square, Triangle, Diamond, Hexagon } from
'lucide-react';

const GlyphEngine = () => {
  const [selectedGate, setSelectedGate] = useState(1);
  const [selectedLine, setSelectedLine] = useState(1);
  const [selectedField, setSelectedField] = useState('mind');
  const [animationSpeed, setAnimationSpeed] = useState(2);
  const [glyphSize, setGlyphSize] = useState(120);

  // Core glyph shapes based on I Ching trigrams and sacred geometry
  const baseShapes = {
    circle: { path: 'M50,10 A40,40 0 1,1 50,90 A40,40 0 1,1 50,10', type: 'circle' },
    square: { path: 'M20,20 L80,20 L80,80 L20,80 Z', type: 'square' },
    triangle: { path: 'M50,15 L85,75 L15,75 Z', type: 'triangle' },
    diamond: { path: 'M50,10 L80,50 L50,90 L20,50 Z', type: 'diamond' },
    hexagon: { path: 'M50,15 L75,30 L75,70 L50,85 L25,70 L25,30 Z', type: 'hexagon' },
    star: { path: 'M50,5 L58,35 L90,35 L65,55 L73,85 L50,70 L27,85 L35,55 L10,35 L42,35 Z',
type: 'star' }
  };

  // Field color mappings
  const fieldColors = {
    mind: { primary: '#06b6d4', secondary: '#0891b2', accent: '#22d3ee' },
    body: { primary: '#14b8a6', secondary: '#0f766e', accent: '#2dd4bf' },
    heart: { primary: '#10b981', secondary: '#059669', accent: '#34d399' },
    soul: { primary: '#0ea5e9', secondary: '#0284c7', accent: '#38bdf8' },
    spirit: { primary: '#06b6d4', secondary: '#0891b2', accent: '#67e8f9' }
  };

  // Generate glyph based on gate number (1-64)
  const generateGateGlyph = (gate) => {
    const shapeNames = Object.keys(baseShapes);
    const primaryShape = shapeNames[gate % shapeNames.length];
    const secondaryShape = shapeNames[(gate * 2) % shapeNames.length];
    return { primary: primaryShape, secondary: secondaryShape };
  };

  // Generate line modifications (1-6)
  const generateLineModification = (line) => {
    const modifications = [
```

```
    { type: 'solid', pattern: 'none' },
    { type: 'dashed', pattern: '5,5' },
    { type: 'dotted', pattern: '2,3' },
    { type: 'double', pattern: 'none', strokeWidth: 3 },
    { type: 'wavy', pattern: 'none', filter: 'url(#wave)' },
    { type: 'glowing', pattern: 'none', filter: 'url(#glow)' }
  ];
  return modifications[line - 1] || modifications[0];
};

// Animated glyph component
const AnimatedGlyph = ({ gate, line, field, size = 100 }) => {
  const shapes = generateGateGlyph(gate);
  const lineStyle = generateLineModification(line);
  const colors = fieldColors[field];

  return (
    <div className="relative">
      <svg
        width={size}
        height={size}
        viewBox="0 0 100 100"
        className="drop-shadow-lg"
      >
        {/* Gradient Definitions */}
        <defs>
          <radialGradient id={`gradient-${field}-${gate}-${line}`} cx="50%" cy="50%" r="50%">
            <stop offset="0%" stopColor={colors.accent} />
            <stop offset="100%" stopColor={colors.primary} />
          </radialGradient>

          <linearGradient id={`linear-${field}-${gate}-${line}`} x1="0%" y1="0%" x2="100%"
y2="100%">
            <stop offset="0%" stopColor={colors.primary} />
            <stop offset="50%" stopColor={colors.accent} />
            <stop offset="100%" stopColor={colors.secondary} />
          </linearGradient>

          {/* Wave filter for line 5 */}
          <filter id="wave">
            <feTurbulence baseFrequency="0.02" numOctaves="3" />
            <feDisplacementMap in="SourceGraphic" scale="2" />
          </filter>
```

```
  {/* Glow filter for line 6 */}
  <filter id="glow">
    <feGaussianBlur stdDeviation="3" result="coloredBlur"/>
    <feMerge>
      <feMergeNode in="coloredBlur"/>
      <feMergeNode in="SourceGraphic"/>
    </feMerge>
  </filter>
</defs>

{/* Background Circle */}
<circle
  cx="50"
  cy="50"
  r="48"
  fill="rgba(0,0,0,0.3)"
  stroke={colors.accent}
  strokeWidth="0.5"
/>

{/* Primary Shape */}
<path
  d={baseShapes[shapes.primary].path}
  fill={`url(#gradient-${field}-${gate}-${line})`}
  stroke={colors.primary}
  strokeWidth={lineStyle.strokeWidth || 2}
  strokeDasharray={lineStyle.pattern}
  filter={lineStyle.filter}
  className="animate-pulse"
  style={{
    animationDuration: `${animationSpeed}s`,
    transformOrigin: 'center'
  }}
/>

{/* Secondary Shape (smaller, rotated) */}
<path
  d={baseShapes[shapes.secondary].path}
  fill="none"
  stroke={colors.accent}
  strokeWidth="1.5"
  strokeDasharray={lineStyle.pattern}
  transform="scale(0.6) translate(20, 20) rotate(45 50 50)"
  opacity="0.7"
```

```
          className="animate-spin"
          style={{
            animationDuration: `${animationSpeed * 3}s`,
            transformOrigin: 'center'
          }}
        />

        {/* Gate Number */}
        <text
          x="50"
          y="50"
          textAnchor="middle"
          dominantBaseline="central"
          fill={colors.accent}
          fontSize="12"
          fontWeight="bold"
          className="font-mono"
        >
          {gate}
        </text>

        {/* Line Indicators */}
        {Array.from({length: line}).map((_, i) => (
          <circle
            key={i}
            cx={85 + (i * 4)}
            cy={15}
            r="1.5"
            fill={colors.accent}
            className="animate-pulse"
            style={{
              animationDelay: `${i * 0.2}s`,
              animationDuration: '1s'
            }}
          />
        ))}
      </svg>
    </div>
  );
};

// Glyph library display
const GlyphLibrary = () => {
  const sampleGates = [1, 8, 14, 23, 32, 41, 47, 56, 64];
```

```
  return (
    <div className="grid grid-cols-3 md:grid-cols-4 lg:grid-cols-6 gap-4">
      {sampleGates.map(gate => (
        <div
          key={gate}
          className="bg-gray-900/50 p-4 rounded-lg border border-cyan-800/30
hover:border-teal-400 transition-all duration-300 cursor-pointer"
          onClick={() => setSelectedGate(gate)}
        >
          <div className="flex flex-col items-center space-y-2">
            <AnimatedGlyph gate={gate} line={3} field={selectedField} size={80} />
            <span className="text-cyan-300 text-sm font-semibold">Gate {gate}</span>
          </div>
        </div>
      ))}
    </div>
  );
};

  return (
    <div className="min-h-screen bg-black text-cyan-100 p-6">
      <div className="max-w-7xl mx-auto">
        {/* Header */}
        <div className="text-center mb-8">
          <h1 className="text-4xl font-bold bg-gradient-to-r from-cyan-400 to-teal-400 bg-clip-text
text-transparent mb-4">
            Glyph Engine
          </h1>
          <p className="text-cyan-300 text-lg">
            Sacred geometric symbols dynamically generated from Gate.Line.Field combinations
          </p>
        </div>

        <div className="grid grid-cols-1 lg:grid-cols-3 gap-8">
          {/* Controls Panel */}
          <div className="bg-gray-900/50 p-6 rounded-xl border border-cyan-800/30">
            <h2 className="text-xl font-semibold text-cyan-300 mb-6">Glyph Controls</h2>

            <div className="space-y-6">
              {/* Gate Selector */}
              <div>
                <label className="block text-cyan-300 text-sm font-semibold mb-2">
                  Gate (1-64)
```

```
            </label>
            <input
              type="range"
              min="1"
              max="64"
              value={selectedGate}
              onChange={(e) => setSelectedGate(parseInt(e.target.value))}
              className="w-full h-2 bg-gray-700 rounded-lg appearance-none cursor-pointer"
            />
            <div className="text-center mt-2">
              <span className="bg-gradient-to-r from-cyan-500 to-teal-500 text-black px-3 py-1
rounded-full font-bold">
                {selectedGate}
              </span>
            </div>
          </div>

          {/* Line Selector */}
          <div>
            <label className="block text-cyan-300 text-sm font-semibold mb-2">
              Line (1-6)
            </label>
            <div className="grid grid-cols-6 gap-2">
              {[1,2,3,4,5,6].map(line => (
                <button
                  key={line}
                  onClick={() => setSelectedLine(line)}
                  className={`py-2 rounded-lg transition-all duration-300 ${
                    selectedLine === line
                      ? 'bg-gradient-to-r from-cyan-500 to-teal-500 text-black'
                      : 'bg-gray-800 text-cyan-300 hover:bg-gray-700'
                  }`}
                >
                  {line}
                </button>
              ))}
            </div>
          </div>

          {/* Field Selector */}
          <div>
            <label className="block text-cyan-300 text-sm font-semibold mb-2">
              Trinity Field
            </label>
```

```
            <div className="space-y-2">
              {Object.keys(fieldColors).map(field => (
                <button
                  key={field}
                  onClick={() => setSelectedField(field)}
                  className={`w-full flex items-center space-x-3 p-3 rounded-lg transition-all
duration-300 ${
                    selectedField === field
                      ? 'bg-gradient-to-r from-cyan-500 to-teal-500 text-black'
                      : 'bg-gray-800 text-cyan-300 hover:bg-gray-700'
                  }`}
                >
                  {field === 'mind' && <Brain className="w-4 h-4" />}
                  {field === 'body' && <Zap className="w-4 h-4" />}
                  {field === 'heart' && <Heart className="w-4 h-4" />}
                  {field === 'soul' && <Eye className="w-4 h-4" />}
                  {field === 'spirit' && <Star className="w-4 h-4" />}
                  <span className="capitalize font-semibold">{field}</span>
                </button>
              ))}
            </div>
          </div>

          {/* Animation Speed */}
          <div>
            <label className="block text-cyan-300 text-sm font-semibold mb-2">
              Animation Speed
            </label>
            <input
              type="range"
              min="0.5"
              max="5"
              step="0.5"
              value={animationSpeed}
              onChange={(e) => setAnimationSpeed(parseFloat(e.target.value))}
              className="w-full h-2 bg-gray-700 rounded-lg appearance-none cursor-pointer"
            />
            <div className="text-center mt-2 text-cyan-300">
              {animationSpeed}s
            </div>
          </div>

          {/* Glyph Size */}
          <div>
```

```
            <label className="block text-cyan-300 text-sm font-semibold mb-2">
              Glyph Size
            </label>
            <input
              type="range"
              min="60"
              max="200"
              step="10"
              value={glyphSize}
              onChange={(e) => setGlyphSize(parseInt(e.target.value))}
              className="w-full h-2 bg-gray-700 rounded-lg appearance-none cursor-pointer"
            />
            <div className="text-center mt-2 text-cyan-300">
              {glyphSize}px
            </div>
          </div>
        </div>
      </div>

      {/* Main Glyph Display */}
      <div className="bg-gray-900/50 p-8 rounded-xl border border-cyan-800/30 flex flex-col
items-center justify-center">
        <h2 className="text-xl font-semibold text-cyan-300 mb-6">Generated Glyph</h2>

        <div className="mb-6">
          <AnimatedGlyph
            gate={selectedGate}
            line={selectedLine}
            field={selectedField}
            size={glyphSize}
          />
        </div>

        <div className="text-center space-y-2">
          <div className="bg-black/30 p-4 rounded-lg">
            <p className="text-cyan-300 font-mono text-lg">
              Gate {selectedGate}.{selectedLine}
            </p>
            <p className="text-teal-400 capitalize font-semibold">
              {selectedField} Field
            </p>
          </div>

          <div className="flex space-x-2 mt-4">
```

```
            <button className="bg-gradient-to-r from-cyan-500 to-teal-500 text-black font-bold
px-4 py-2 rounded-lg hover:from-cyan-400 hover:to-teal-400 transition-all duration-300">
              Export Glyph
            </button>
            <button className="bg-gray-800 text-cyan-300 px-4 py-2 rounded-lg
hover:bg-gray-700 transition-colors duration-300">
              Save to Library
            </button>
          </div>
        </div>
      </div>

      {/* Glyph Properties */}
      <div className="bg-gray-900/50 p-6 rounded-xl border border-cyan-800/30">
        <h2 className="text-xl font-semibold text-cyan-300 mb-6">Glyph Properties</h2>

        <div className="space-y-4">
          {/* Gate Information */}
          <div className="bg-black/30 p-4 rounded-lg">
            <h3 className="text-cyan-400 font-semibold mb-2">Gate {selectedGate}</h3>
            <p className="text-cyan-300 text-sm">
              {selectedGate <= 16 ? 'Creation Quadrant' :
               selectedGate <= 32 ? 'Evolution Quadrant' :
               selectedGate <= 48 ? 'Transformation Quadrant' : 'Integration Quadrant'}
            </p>
            <div className="mt-2">
              <span className="text-teal-400 text-xs font-mono">
                Binary: {selectedGate.toString(2).padStart(6, '0')}
              </span>
            </div>
          </div>

          {/* Line Information */}
          <div className="bg-black/30 p-4 rounded-lg">
            <h3 className="text-cyan-400 font-semibold mb-2">Line {selectedLine}</h3>
            <p className="text-cyan-300 text-sm">
              {selectedLine === 1 ? 'Foundation - Introspection' :
               selectedLine === 2 ? 'Hermit - Natural Knowing' :
               selectedLine === 3 ? 'Martyr - Trial & Error' :
               selectedLine === 4 ? 'Opportunist - Externalization' :
               selectedLine === 5 ? 'Heretic - Universalization' :
               'Role Model - Transition'}
            </p>
            <div className="mt-2">
```

```
              <span className="text-teal-400 text-xs">
                {selectedLine <= 3 ? 'Lower Trigram (Personal)' : 'Upper Trigram (Transpersonal)'}
              </span>
            </div>
          </div>

          {/* Field Properties */}
          <div className="bg-black/30 p-4 rounded-lg">
            <h3 className="text-cyan-400 font-semibold mb-2 capitalize">{selectedField}
Field</h3>
            <div className="space-y-2 text-sm">
              <div className="flex justify-between">
                <span className="text-cyan-300">Element:</span>
                <span className="text-teal-400">
                  {selectedField === 'mind' ? 'Air' :
                   selectedField === 'body' ? 'Earth' :
                   selectedField === 'heart' ? 'Fire' :
                   selectedField === 'soul' ? 'Water' :
                   'Ether'}
                </span>
              </div>
              <div className="flex justify-between">
                <span className="text-cyan-300">Frequency:</span>
                <span className="text-teal-400">
                  {selectedField === 'mind' ? '432 Hz' :
                   selectedField === 'body' ? '256 Hz' :
                   selectedField === 'heart' ? '528 Hz' :
                   selectedField === 'soul' ? '741 Hz' :
                   '963 Hz'}
                </span>
              </div>
              <div className="flex justify-between">
                <span className="text-cyan-300">Chakra:</span>
                <span className="text-teal-400">
                  {selectedField === 'mind' ? 'Ajna' :
                   selectedField === 'body' ? 'Muladhara' :
                   selectedField === 'heart' ? 'Anahata' :
                   selectedField === 'soul' ? 'Vishuddha' :
                   'Sahasrara'}
                </span>
              </div>
            </div>
          </div>
```

```
            {/* Sacred Geometry */}
            <div className="bg-black/30 p-4 rounded-lg">
              <h3 className="text-cyan-400 font-semibold mb-2">Sacred Geometry</h3>
              <div className="space-y-2 text-sm">
                <div className="flex justify-between">
                  <span className="text-cyan-300">Primary Shape:</span>
                  <span className="text-teal-400 capitalize">
                    {generateGateGlyph(selectedGate).primary}
                  </span>
                </div>
                <div className="flex justify-between">
                  <span className="text-cyan-300">Secondary:</span>
                  <span className="text-teal-400 capitalize">
                    {generateGateGlyph(selectedGate).secondary}
                  </span>
                </div>
                <div className="flex justify-between">
                  <span className="text-cyan-300">Line Style:</span>
                  <span className="text-teal-400 capitalize">
                    {generateLineModification(selectedLine).type}
                  </span>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>

      {/* Glyph Library Section */}
      <div className="mt-12">
        <h2 className="text-2xl font-semibold text-cyan-300 mb-6">Glyph Library</h2>
        <div className="bg-gray-900/50 p-6 rounded-xl border border-cyan-800/30">
          <GlyphLibrary />
        </div>
      </div>

      {/* Export Options */}
      <div className="mt-8 bg-gray-900/50 p-6 rounded-xl border border-cyan-800/30">
        <h2 className="text-xl font-semibold text-cyan-300 mb-4">Export & Integration</h2>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
          <button className="bg-gradient-to-r from-cyan-500 to-teal-500 text-black font-bold py-3
px-6 rounded-lg hover:from-cyan-400 hover:to-teal-400 transition-all duration-300">
            Export as SVG
          </button>
```

```
          <button className="bg-gradient-to-r from-teal-500 to-emerald-500 text-black font-bold
py-3 px-6 rounded-lg hover:from-teal-400 hover:to-emerald-400 transition-all duration-300">
            Generate Font Set
          </button>
          <button className="bg-gradient-to-r from-emerald-500 to-cyan-500 text-black font-bold
py-3 px-6 rounded-lg hover:from-emerald-400 hover:to-cyan-400 transition-all duration-300">
            Create Glyph API
          </button>
        </div>
      </div>
    </div>
  </div>
 );
};

export default GlyphEngine;

import React, { useState, useEffect } from 'react';
import {
 Brain, Heart, Zap, Eye, Star, Menu, X, Calendar, Clock,
 Activity, MessageCircle, BarChart3, Settings, User, Bell,
 Sparkles, Globe, Target, BookOpen, Gamepad2, TestTube,
 Moon, Sun, Play, Pause, ChevronRight, TrendingUp
} from 'lucide-react';

const StellarDashboard = () => {
 const [currentTime, setCurrentTime] = useState(new Date());
 const [activeModule, setActiveModule] = useState('overview');
 const [sidebarOpen, setSidebarOpen] = useState(false); // Start closed on mobile
 const [isSimulationRunning, setIsSimulationRunning] = useState(false);
 const [currentField, setCurrentField] = useState('mind');

 // Update time every second
 useEffect(() => {
  const timer = setInterval(() => setCurrentTime(new Date()), 1000);
  return () => clearInterval(timer);
 }, []);

 // Close sidebar when clicking outside on mobile
 useEffect(() => {
  const handleResize = () => {
   if (window.innerWidth >= 1024) {
    setSidebarOpen(true); // Auto-open on desktop
   } else {
```

```
      setSidebarOpen(false); // Auto-close on mobile
    }
  };

  // Handle mobile orientation changes
  const handleOrientationChange = () => {
    setTimeout(handleResize, 100);
  };

  handleResize(); // Set initial state
  window.addEventListener('resize', handleResize);
  window.addEventListener('orientationchange', handleOrientationChange);

  return () => {
    window.removeEventListener('resize', handleResize);
    window.removeEventListener('orientationchange', handleOrientationChange);
  };
}, []);

// Mobile swipe detection for sidebar
const [touchStart, setTouchStart] = useState(null);
const [touchEnd, setTouchEnd] = useState(null);

const minSwipeDistance = 50;

const onTouchStart = (e) => {
  setTouchEnd(null);
  setTouchStart(e.targetTouches[0].clientX);
};

const onTouchMove = (e) => {
  setTouchEnd(e.targetTouches[0].clientX);
};

const onTouchEnd = () => {
  if (!touchStart || !touchEnd) return;
  const distance = touchStart - touchEnd;
  const isLeftSwipe = distance > minSwipeDistance;
  const isRightSwipe = distance < -minSwipeDistance;

  if (isRightSwipe && !sidebarOpen && window.innerWidth < 1024) {
    setSidebarOpen(true);
  }
  if (isLeftSwipe && sidebarOpen && window.innerWidth < 1024) {
```

```
    setSidebarOpen(false);
  }
};

// Mock Trinity Field data with real-time fluctuations
const [trinityFields, setTrinityFields] = useState({
  mind: { energy: 72, coherence: 85, resonance: 'Active' },
  body: { energy: 68, coherence: 79, resonance: 'Stable' },
  heart: { energy: 91, coherence: 94, resonance: 'Elevated' },
  soul: { energy: 56, coherence: 67, resonance: 'Seeking' },
  spirit: { energy: 83, coherence: 88, resonance: 'Flowing' }
});

// Mock planetary data
const planetaryData = {
  mercury: { sign: 'Gemini', degree: '15°23\'', influence: 'High' },
  venus: { sign: 'Taurus', degree: '22°44\'', influence: 'Medium' },
  mars: { sign: 'Aries', degree: '8°17\'', influence: 'Strong' },
  jupiter: { sign: 'Sagittarius', degree: '11°08\'', influence: 'Expansive' },
  saturn: { sign: 'Aquarius', degree: '6°32\'', influence: 'Structured' }
};

const fieldIcons = {
  mind: Brain,
  body: Zap,
  heart: Heart,
  soul: Eye,
  spirit: Star
};

const fieldColors = {
  mind: 'from-cyan-400 to-cyan-600',
  body: 'from-teal-400 to-teal-600',
  heart: 'from-emerald-400 to-emerald-600',
  soul: 'from-blue-400 to-blue-600',
  spirit: 'from-cyan-300 to-teal-500'
};

const modules = [
  { id: 'overview', name: 'Overview', icon: Globe, color: 'cyan' },
  { id: 'trinity-chart', name: 'Trinity Chart', icon: Target, color: 'teal' },
  { id: 'consciousness', name: 'Consciousness Sim', icon: Brain, color: 'emerald' },
  { id: 'oracle', name: 'Oracle', icon: MessageCircle, color: 'blue' },
  { id: 'memory-vault', name: 'Memory Vault', icon: BarChart3, color: 'cyan' },
```

```
  { id: 'glyph-engine', name: 'Glyph Engine', icon: Sparkles, color: 'teal' },
  { id: 'game-mode', name: 'Game Mode', icon: Gamepad2, color: 'emerald' },
  { id: 'lab', name: 'Trinity Lab', icon: TestTube, color: 'blue' },
  { id: 'learning', name: 'Learning Portal', icon: BookOpen, color: 'cyan' }
 ];

 // Field Energy Gauge Component
 const FieldGauge = ({ field, data }) => {
  const Icon = fieldIcons[field];
  const percentage = data.energy;

  return (
   <div
    className={`bg-gradient-to-br ${fieldColors[field]} p-3 lg:p-4 rounded-lg shadow-lg border
border-cyan-800/30 cursor-pointer hover:scale-105 transition-transform duration-300`}
    onClick={() => setCurrentField(field)}
   >
    <div className="flex items-center justify-between mb-2 lg:mb-3">
     <div className="flex items-center space-x-2">
      <Icon className="w-4 h-4 lg:w-5 lg:h-5 text-black" />
      <span className="text-black font-bold capitalize text-sm lg:text-base">{field}</span>
     </div>
     <span className="text-black/70 text-xs lg:text-sm
font-semibold">{data.resonance}</span>
    </div>

    <div className="space-y-2">
     <div className="flex justify-between text-black text-xs lg:text-sm">
      <span>Energy</span>
      <span>{percentage}%</span>
     </div>
     <div className="w-full bg-black/20 rounded-full h-2">
      <div
       className="bg-black/60 h-2 rounded-full transition-all duration-1000"
       style={{ width: `${percentage}%` }}
      ></div>
     </div>
     <div className="flex justify-between text-black text-xs">
      <span>Coherence: {data.coherence}%</span>
      <span className="font-mono">●</span>
     </div>
    </div>
   </div>
  );
```

```
  };

  // Planetary Influence Display
  const PlanetaryWidget = () => (
    <div className="bg-gray-900/50 p-4 lg:p-6 rounded-xl border border-cyan-800/30">
      <h3 className="text-base lg:text-lg font-semibold text-cyan-300 mb-3 lg:mb-4 flex
items-center">
        <Globe className="w-4 h-4 lg:w-5 lg:h-5 mr-2" />
        Current Planetary Influences
      </h3>
      <div className="space-y-2 lg:space-y-3">
        {Object.entries(planetaryData).map(([planet, data]) => (
          <div key={planet} className="flex items-center justify-between">
            <div className="flex items-center space-x-2 lg:space-x-3">
              <div className="w-2 h-2 lg:w-3 lg:h-3 bg-gradient-to-r from-cyan-400 to-teal-400
rounded-full"></div>
              <span className="text-cyan-100 capitalize font-medium text-sm
lg:text-base">{planet}</span>
            </div>
            <div className="text-right">
              <div className="text-cyan-300 text-xs lg:text-sm">{data.sign} {data.degree}</div>
              <div className="text-teal-400 text-xs">{data.influence}</div>
            </div>
          </div>
        ))}
      </div>
    </div>
  );

  // Recent Activity Feed
  const ActivityFeed = () => (
    <div className="bg-gray-900/50 p-4 lg:p-6 rounded-xl border border-cyan-800/30">
      <h3 className="text-base lg:text-lg font-semibold text-cyan-300 mb-3 lg:mb-4 flex
items-center">
        <Activity className="w-4 h-4 lg:w-5 lg:h-5 mr-2" />
        Recent Activity
      </h3>
      <div className="space-y-2 lg:space-y-3">
        <div className="flex items-start space-x-2 lg:space-x-3">
          <div className="w-2 h-2 bg-cyan-400 rounded-full mt-2"></div>
          <div>
            <p className="text-cyan-100 text-xs lg:text-sm">Heart Field coherence peaked at
94%</p>
            <p className="text-cyan-400 text-xs">2 minutes ago</p>
```

```
          </div>
        </div>
        <div className="flex items-start space-x-2 lg:space-x-3">
          <div className="w-2 h-2 bg-teal-400 rounded-full mt-2"></div>
          <div>
            <p className="text-cyan-100 text-xs lg:text-sm">Venus transit activated Gate 26</p>
            <p className="text-cyan-400 text-xs">15 minutes ago</p>
          </div>
        </div>
        <div className="flex items-start space-x-2 lg:space-x-3">
          <div className="w-2 h-2 bg-emerald-400 rounded-full mt-2"></div>
          <div>
            <p className="text-cyan-100 text-xs lg:text-sm">Oracle consulted: "What is my
purpose?"</p>
            <p className="text-cyan-400 text-xs">1 hour ago</p>
          </div>
        </div>
        <div className="flex items-start space-x-2 lg:space-x-3">
          <div className="w-2 h-2 bg-blue-400 rounded-full mt-2"></div>
          <div>
            <p className="text-cyan-100 text-xs lg:text-sm">Trinity Chart generated
successfully</p>
            <p className="text-cyan-400 text-xs">3 hours ago</p>
          </div>
        </div>
      </div>

      {/* Integrated Lumina AI Companion */}
      <LuminaIntegrated activeModule={activeModule} trinityFields={trinityFields}
currentField={currentField} />
    </div>
  );
};

// Integrated Lumina AI Companion Component
const LuminaIntegrated = ({ activeModule, trinityFields, currentField }) => {
  const [isOpen, setIsOpen] = useState(false);
  const [isMinimized, setIsMinimized] = useState(false);
  const [inputMessage, setInputMessage] = useState('');
  const [conversationHistory, setConversationHistory] = useState([
    {
      id: 1,
      sender: 'lumina',
```

```
      message: `✨ Hello! I'm Lumina, integrated with your dashboard. I can see you're currently
in the ${activeModule.replace('-', ' ')} module. How can I help you navigate your consciousness
journey?`,
      timestamp: new Date(),
      moduleContext: activeModule,
      fieldData: trinityFields
    }
  ]);

  const messagesEndRef = useRef(null);

  // Auto-scroll to bottom
  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [conversationHistory]);

  // Update context when module changes
  useEffect(() => {
    if (conversationHistory.length > 1) { // Don't trigger on initial load
      const contextMessage = {
        id: Date.now(),
        sender: 'lumina',
        message: `I notice you've switched to ${activeModule.replace('-', ' ')}.
${getModuleGuidance(activeModule)}`,
        timestamp: new Date(),
        moduleContext: activeModule,
        fieldData: trinityFields,
        isContextUpdate: true
      };
      setConversationHistory(prev => [...prev, contextMessage]);
    }
  }, [activeModule]);

  const getModuleGuidance = (module) => {
    const guidance = {
      'overview': 'Your Trinity Fields are showing interesting patterns. Would you like me to
analyze your current coherence levels?',
      'trinity-chart': 'Perfect for deep chart analysis! I can help interpret your Trinity Field patterns.',
      'consciousness': 'The consciousness simulator is powerful. I can guide you through
understanding the wave patterns.',
      'oracle': 'Ready for oracle consultation? I can help you formulate the perfect question.',
      'memory-vault': 'Time to explore your consciousness timeline. What patterns would you like
me to help identify?',
```

```
    'glyph-engine': 'Glyph creation mode! I can help you understand the symbolic meanings as they emerge.',
    'game-mode': 'Adventure time! I can provide hints and guidance for your consciousness quests.',
    'lab': 'Scientific mode activated. I can help you design experiments and interpret results.',
    'learning': 'Study session! What aspect of Trinity Field theory would you like to explore?'
   };
   return guidance[module] || 'How can I assist you in this module?';
  };

  const generateContextualResponse = (userMessage) => {
   // Generate responses based on current module and field data
   const responses = {
    'overview': [
     `Looking at your current fields: ${currentField} is at ${trinityFields[currentField].coherence}% coherence with ${trinityFields[currentField].resonance.toLowerCase()} resonance. ${userMessage.toLowerCase().includes('coherence') ? 'This level suggests stable consciousness integration.' : 'Would you like specific guidance for enhancing this field?'}`,
     `Your Trinity Field overview shows ${Object.entries(trinityFields).filter(([f, data]) => data.coherence > 80).length} fields above 80% coherence. ${userMessage.toLowerCase().includes('help') ? 'I can guide you toward optimizing the lower-coherence fields.' : 'This is a strong foundation for consciousness work.'}`
    ],
    'oracle': [
     `For oracle consultation, I recommend focusing on your ${Object.entries(trinityFields).reduce((a, b) => trinityFields[a[0]].coherence > trinityFields[b[0]].coherence ? a : b)[0]} field which is resonating strongest at ${Object.entries(trinityFields).reduce((a, b) => trinityFields[a[0]].coherence > trinityFields[b[0]].coherence ? a : b)[1].coherence}%. This field will provide the clearest guidance.`,
     `Your question "${userMessage}" resonates with ${currentField} field energy. The ${trinityFields[currentField].resonance.toLowerCase()} state suggests ${trinityFields[currentField].coherence > 75 ? 'you\'re ready for deep insights' : 'grounding practices might help first'}.`
    ],
    'consciousness': [
     `The consciousness simulator shows your ${currentField} field in ${trinityFields[currentField].resonance.toLowerCase()} state. ${userMessage.toLowerCase().includes('wave') ? 'Wave patterns here indicate optimal consciousness flow.' : 'This creates specific resonance patterns you can explore.'}`,
     `Your multi-field coherence average is ${Math.round(Object.values(trinityFields).reduce((sum, field) => sum + field.coherence, 0) / 5)}%. The simulator can help you understand how these fields interact dynamically.`
```

```
    ]
  };

  const moduleResponses = responses[activeModule] || [
    `In ${activeModule.replace('-', ' ')} mode, your ${currentField} field
(${trinityFields[currentField].coherence}% coherence) suggests
${trinityFields[currentField].resonance.toLowerCase()} energy flow. How can I help you work
with this?`,
    `Your current consciousness configuration shows strong
${Object.entries(trinityFields).filter(([f, data]) => data.coherence > 80).map(([f]) => f).join(' and ')}
field${Object.entries(trinityFields).filter(([f, data]) => data.coherence > 80).length > 1 ? 's' : ''}.
Perfect for ${activeModule.replace('-', ' ')} work!`
  ];

  return moduleResponses[Math.floor(Math.random() * moduleResponses.length)];
 };

 const handleSendMessage = () => {
  if (!inputMessage.trim()) return;

  const userMsg = {
   id: Date.now(),
   sender: 'user',
   message: inputMessage,
   timestamp: new Date(),
   moduleContext: activeModule
  };

  setConversationHistory(prev => [...prev, userMsg]);

  // Generate contextual AI response
  setTimeout(() => {
   const aiResponse = {
    id: Date.now() + 1,
    sender: 'lumina',
    message: generateContextualResponse(inputMessage),
    timestamp: new Date(),
    moduleContext: activeModule,
    fieldData: trinityFields
   };
   setConversationHistory(prev => [...prev, aiResponse]);
  }, 1000 + Math.random() * 2000);

  setInputMessage('');
```

```
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault();
      handleSendMessage();
    }
  };

  if (!isOpen) {
    return (
      <div className="fixed bottom-6 right-6 z-50">
        <button
          onClick={() => setIsOpen(true)}
          className="w-14 h-14 bg-gradient-to-r from-cyan-500 to-teal-500 rounded-full shadow-lg
hover:scale-110 transition-transform duration-300 flex items-center justify-center relative"
        >
          <Brain className="w-6 h-6 text-black" />
          {/* Module-aware notification dot */}
          <div className="absolute -top-1 -right-1 w-4 h-4 bg-gradient-to-r from-purple-500
to-pink-500 rounded-full flex items-center justify-center">
            <span className="text-white text-xs font-bold">!</span>
          </div>
        </button>
      </div>
    );
  }

  return (
    <div className={`fixed bottom-6 right-6 z-50 bg-gray-900/95 backdrop-blur-lg border
border-cyan-800/50 rounded-xl shadow-2xl transition-all duration-300 ${
      isMinimized ? 'w-80 h-16' : 'w-96 h-[500px]'
    }`}>
      {/* Header */}
      <div className="flex items-center justify-between p-4 border-b border-cyan-800/30">
        <div className="flex items-center space-x-3">
          <div className="w-8 h-8 bg-gradient-to-r from-cyan-400 to-teal-400 rounded-full flex
items-center justify-center">
            <Brain className="w-4 h-4 text-black" />
          </div>
          <div>
            <h3 className="text-cyan-100 font-semibold text-sm">Lumina</h3>
            <p className="text-cyan-400 text-xs">
```

```
            {activeModule.replace('-', ' ')} Mode • {Object.values(trinityFields).filter(f => f.coherence
> 80).length}/5 Fields Optimal
          </p>
        </div>
      </div>

      <div className="flex items-center space-x-2">
        <button
          onClick={() => setIsMinimized(!isMinimized)}
          className="p-1.5 hover:bg-gray-800 rounded-lg transition-colors duration-200"
        >
          {isMinimized ?
            <ChevronRight className="w-4 h-4 text-cyan-300" /> :
            <ChevronLeft className="w-4 h-4 text-cyan-300" />
          }
        </button>

        <button
          onClick={() => setIsOpen(false)}
          className="p-1.5 hover:bg-gray-800 rounded-lg transition-colors duration-200"
        >
          <X className="w-4 h-4 text-cyan-300" />
        </button>
      </div>
    </div>

    {!isMinimized && (
      <>
        {/* Current Context Display */}
        <div className="px-4 py-2 bg-gradient-to-r from-cyan-900/30 to-purple-900/30 border-b
border-cyan-800/30">
          <div className="flex items-center justify-between text-xs">
            <span className="text-cyan-300">Current: {activeModule.replace('-', ' ')}</span>
            <span className="text-cyan-300">Focus: {currentField}
({trinityFields[currentField].coherence}%)</span>
          </div>
        </div>

        {/* Messages */}
        <div className="flex-1 p-4 overflow-y-auto max-h-80">
          {conversationHistory.map((message) => (
            <div key={message.id} className={`flex ${message.sender === 'user' ? 'justify-end' :
'justify-start'} mb-3`}>
              <div className={`max-w-xs px-3 py-2 rounded-lg ${
```

```
              message.sender === 'user'
                ? 'bg-gradient-to-r from-cyan-500 to-teal-500 text-black'
                : message.isContextUpdate
                  ? 'bg-gradient-to-r from-purple-800/50 to-pink-800/50 text-cyan-100 border
border-purple-500/30'
                  : 'bg-gray-800 text-cyan-100 border border-cyan-700/30'
            }`}>
              <div className="flex items-center space-x-2 mb-1">
                {message.sender === 'user' ? (
                  <User className="w-3 h-3" />
                ) : (
                  <Brain className="w-3 h-3 text-cyan-400" />
                )}
                <span className="text-xs opacity-75">
                  {message.sender === 'user' ? 'You' : 'Lumina'}
                </span>
                <span className="text-xs opacity-50">
                  {message.timestamp.toLocaleTimeString()}
                </span>
              </div>

              <p className="text-xs leading-relaxed">{message.message}</p>

              {message.sender === 'lumina' && message.moduleContext && (
                <div className="mt-1 pt-1 border-t border-cyan-700/30">
                  <span className="text-xs text-cyan-300
capitalize">{message.moduleContext.replace('-', ' ')} Context</span>
    Consciousness Metrics
              </h3>
              <div className="grid grid-cols-2 gap-3 lg:gap-4">
                <div className="text-center">
                  <div className="text-xl lg:text-2xl font-bold text-cyan-100">87%</div>
                  <div className="text-cyan-400 text-xs lg:text-sm">Overall Coherence</div>
                </div>
                <div className="text-center">
                  <div className="text-xl lg:text-2xl font-bold text-teal-400">124</div>
                  <div className="text-cyan-400 text-xs lg:text-sm">Oracle Consultations</div>
                </div>
                <div className="text-center">
                  <div className="text-xl lg:text-2xl font-bold text-emerald-400">15</div>
                  <div className="text-cyan-400 text-xs lg:text-sm">Days Tracked</div>
                </div>
                <div className="text-center">
                  <div className="text-xl lg:text-2xl font-bold text-blue-400">9.2</div>
```

```
                <div className="text-cyan-400 text-xs lg:text-sm">Awareness Level</div>
              </div>
            </div>
          </div>
        </div>

        {/* Quick Actions */}
        <div className="bg-gray-900/50 p-4 lg:p-6 rounded-xl border border-cyan-800/30">
          <h3 className="text-base lg:text-lg font-semibold text-cyan-300 mb-3
lg:mb-4">Quick Actions</h3>
          <div className="grid grid-cols-2 lg:grid-cols-4 gap-3 lg:gap-4">
            <button className="flex flex-col lg:flex-row items-center justify-center space-y-1
lg:space-y-0 lg:space-x-2 p-3 lg:p-4 bg-gradient-to-r from-cyan-500 to-teal-500 text-black
font-semibold rounded-lg hover:from-cyan-400 hover:to-teal-400 transition-all duration-300
text-xs lg:text-sm">
              <MessageCircle className="w-4 h-4" />
              <span>Oracle</span>
            </button>
            <button className="flex flex-col lg:flex-row items-center justify-center space-y-1
lg:space-y-0 lg:space-x-2 p-3 lg:p-4 bg-gradient-to-r from-teal-500 to-emerald-500 text-black
font-semibold rounded-lg hover:from-teal-400 hover:to-emerald-400 transition-all duration-300
text-xs lg:text-sm">
              <Target className="w-4 h-4" />
              <span>Chart</span>
            </button>
            <button className="flex flex-col lg:flex-row items-center justify-center space-y-1
lg:space-y-0 lg:space-x-2 p-3 lg:p-4 bg-gradient-to-r from-emerald-500 to-blue-500 text-black
font-semibold rounded-lg hover:from-emerald-400 hover:to-blue-400 transition-all duration-300
text-xs lg:text-sm">
              <Sparkles className="w-4 h-4" />
              <span>Glyph</span>
            </button>
            <button className="flex flex-col lg:flex-row items-center justify-center space-y-1
lg:space-y-0 lg:space-x-2 p-3 lg:p-4 bg-gradient-to-r from-blue-500 to-cyan-500 text-black
font-semibold rounded-lg hover:from-blue-400 hover:to-cyan-400 transition-all duration-300
text-xs lg:text-sm">
              <TestTube className="w-4 h-4" />
              <span>Lab</span>
            </button>
          </div>
        </div>
      </div>
    )}
```

```
        {/* Placeholder for other modules */}
        {activeModule !== 'overview' && (
          <div className="flex items-center justify-center h-64">
            <div className="text-center">
              <div className="w-12 h-12 lg:w-16 lg:h-16 bg-gradient-to-r from-cyan-500
to-teal-500 rounded-full flex items-center justify-center mx-auto mb-4">
                {React.createElement(modules.find(m => m.id === activeModule)?.icon || Globe, {
                  className: "w-6 h-6 lg:w-8 lg:h-8 text-black"
                })}
              </div>
              <h3 className="text-lg lg:text-xl font-semibold text-cyan-300 mb-2">
                {modules.find(m => m.id === activeModule)?.name} Module
              </h3>
              <p className="text-cyan-400 text-sm lg:text-base">This module is ready for
integration</p>
              <button className="mt-4 bg-gradient-to-r from-cyan-500 to-teal-500 text-black
font-bold px-4 lg:px-6 py-2 rounded-lg hover:from-cyan-400 hover:to-teal-400 transition-all
duration-300 text-sm lg:text-base">
                Launch Module
              </button>
            </div>
          </div>
        )}
      </div>
    </div>
  </div>
);
};

export default StellarDashboard;

import React, { useState, useEffect, useRef } from 'react';
import {
  MessageCircle, Send, Sparkles, Brain, Heart, Eye, Star, Zap,
  Mic, MicOff, Volume2, VolumeX, Minimize2, Maximize2, X,
  Bot, User, Clock, TrendingUp, Target, Lightbulb, Compass
} from 'lucide-react';

const StellarAICompanion = () => {
  const [isOpen, setIsOpen] = useState(true);
  const [isMinimized, setIsMinimized] = useState(false);
  const [isListening, setIsListening] = useState(false);
  const [isSpeaking, setIsSpeaking] = useState(false);
  const [inputMessage, setInputMessage] = useState('');
```

```
  const [currentMode, setCurrentMode] = useState('guidance');
  const [conversationHistory, setConversationHistory] = useState([
    {
      id: 1,
      sender: 'lumina',
      message: "✨ Hello! I'm Lumina, your Stellar Proximology guide. I'm here to help you navigate your consciousness journey through the Trinity Fields. How can I assist you today?",
      timestamp: new Date(),
      fieldResonance: 'spirit',
      insights: ['Welcome to your consciousness exploration', 'All Trinity Fields are online and ready']
    }
  ]);

  const messagesEndRef = useRef(null);

  // Auto-scroll to bottom of messages
  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [conversationHistory]);

  // Mock AI responses based on mode and context
  const generateResponse = (userMessage) => {
    const responses = {
      guidance: [
        "Based on your current Trinity Field readings, I sense your Mind field is particularly active. This suggests a time for deep introspection.",
        "Your consciousness pattern shows expansive growth. Let me guide you through understanding this energetic signature.",
        "I notice Mercury is transiting through your Mind field. This is an excellent time for meditation practice."
      ],
      analysis: [
        "Analyzing your field coherence... I see 87% alignment across all Trinity Fields. Your strongest resonance is in Heart.",
        "The planetary influences are creating harmonic resonance in your consciousness matrix. This pattern typically indicates spiritual awakening.",
        "Your recent Oracle consultations show a pattern of seeking purpose. This suggests your Soul field is ready for deeper wisdom."
      ],
      oracle: [
        "The cosmic currents whisper: 'The path reveals itself through walking'. Your Spirit field holds the key to understanding this message.",
```

```
      "I see through the celestial veil... The answer lies in embracing your intuitive nature while honoring your analytical mind.",
      "The Universe speaks through Gate 47: 'Trust the process of becoming'. This resonates with your current life transition."
    ],
    coaching: [
      "Let's explore this together. Your Heart field suggests hidden potential awaits activation. How does this resonate with your current experience?",
      "I recommend focusing on conscious breathing to enhance your Body field coherence. Would you like me to guide you through this?",
      "Your growth pattern indicates accelerated development. Shall we design a practice to support this evolution?"
    ]
  };

  const modeResponses = responses[currentMode];
  return modeResponses[Math.floor(Math.random() * modeResponses.length)];
};

const handleSendMessage = () => {
  if (!inputMessage.trim()) return;

  // Add user message
  const userMsg = {
    id: Date.now(),
    sender: 'user',
    message: inputMessage,
    timestamp: new Date()
  };

  setConversationHistory(prev => [...prev, userMsg]);

  // Simulate AI processing time
  setTimeout(() => {
    const aiResponse = {
      id: Date.now() + 1,
      sender: 'lumina',
      message: generateResponse(inputMessage),
      timestamp: new Date(),
      fieldResonance: ['mind', 'heart', 'body', 'soul', 'spirit'][Math.floor(Math.random() * 5)],
      insights: [
        'Field coherence is optimal for this insight',
        'Planetary alignments support this guidance',
        'Your consciousness is ready for this understanding'
```

```
        ]
      };
      setConversationHistory(prev => [...prev, aiResponse]);
    }, 1000 + Math.random() * 2000);

    setInputMessage('');
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault();
      handleSendMessage();
    }
  };

  const ModeButton = ({ mode, icon: Icon, label }) => (
    <button
      onClick={() => setCurrentMode(mode)}
      className={`flex items-center space-x-2 px-2 lg:px-3 py-2 rounded-lg text-xs lg:text-sm
transition-all duration-300 ${
        currentMode === mode
          ? 'bg-gradient-to-r from-cyan-500 to-teal-500 text-black'
          : 'bg-gray-800 text-cyan-300 hover:bg-gray-700'
      }`}
    >
      <Icon className="w-3 h-3 lg:w-4 lg:h-4" />
      <span className="font-medium hidden sm:inline">{label}</span>
    </button>
  );

  const MessageBubble = ({ message }) => {
    const isUser = message.sender === 'user';
    const fieldColors = {
      mind: 'border-cyan-400',
      heart: 'border-emerald-400',
      body: 'border-teal-400',
      soul: 'border-blue-400',
      spirit: 'border-purple-400'
    };

    return (
      <div className={`flex ${isUser ? 'justify-end' : 'justify-start'} mb-3 lg:mb-4`}>
        <div className={`max-w-xs lg:max-w-md px-3 lg:px-4 py-2 lg:py-3 rounded-lg ${
          isUser
```

```
          ? 'bg-gradient-to-r from-cyan-500 to-teal-500 text-black'
          : `bg-gray-800 text-cyan-100 border-l-4 ${message.fieldResonance ?
fieldColors[message.fieldResonance] : 'border-cyan-400'}`
      }`}>
        <div className="flex items-center space-x-2 mb-1">
          {isUser ? (
            <User className="w-3 h-3 lg:w-4 lg:h-4" />
          ) : (
            <Bot className="w-3 h-3 lg:w-4 lg:h-4 text-cyan-400" />
          )}
          <span className="text-xs opacity-75">
            {isUser ? 'You' : 'Lumina'}
          </span>
          <span className="text-xs opacity-50">
            {message.timestamp.toLocaleTimeString()}
          </span>
        </div>

        <p className="text-xs lg:text-sm leading-relaxed">{message.message}</p>

        {!isUser && message.insights && (
          <div className="mt-2 pt-2 border-t border-cyan-700/30">
            <div className="text-xs text-cyan-300 space-y-1">
              {message.insights.map((insight, index) => (
                <div key={index} className="flex items-center space-x-1">
                  <Sparkles className="w-2 h-2 lg:w-3 lg:h-3" />
                  <span>{insight}</span>
                </div>
              ))}
            </div>
          </div>
        )}
      </div>
    </div>
  );
};

  if (!isOpen) {
    return (
      <div className="fixed bottom-4 right-4 lg:bottom-6 lg:right-6 z-50">
        <button
          onClick={() => setIsOpen(true)}
```

```
          className="w-12 h-12 lg:w-14 lg:h-14 bg-gradient-to-r from-cyan-500 to-teal-500
rounded-full shadow-lg hover:scale-110 transition-transform duration-300 flex items-center
justify-center"
        >
          <Bot className="w-5 h-5 lg:w-6 lg:h-6 text-black" />
        </button>
      </div>
    );
  }

  return (
    <div className={`fixed bottom-4 right-4 lg:bottom-6 lg:right-6 z-50 bg-gray-900/95
backdrop-blur-lg border border-cyan-800/50 rounded-xl shadow-2xl transition-all duration-300 ${
      isMinimized ? 'w-72 lg:w-80 h-14 lg:h-16' : 'w-80 lg:w-96 h-96 lg:h-[600px]'
    }`}>
      {/* Header */}
      <div className="flex items-center justify-between p-3 lg:p-4 border-b border-cyan-800/30">
        <div className="flex items-center space-x-2 lg:space-x-3">
          <div className="w-6 h-6 lg:w-8 lg:h-8 bg-gradient-to-r from-cyan-400 to-teal-400
rounded-full flex items-center justify-center">
            <Bot className="w-3 h-3 lg:w-4 lg:h-4 text-black" />
          </div>
          <div>
            <h3 className="text-cyan-100 font-semibold text-sm lg:text-base">Lumina</h3>
            <p className="text-cyan-400 text-xs">Trinity Field Guide • Online</p>
          </div>
        </div>

        <div className="flex items-center space-x-1 lg:space-x-2">
          <button
            onClick={() => setIsListening(!isListening)}
            className={`p-1.5 lg:p-2 rounded-lg transition-colors duration-200 ${
              isListening ? 'bg-red-500 text-white' : 'hover:bg-gray-800 text-cyan-300'
            }`}
          >
            {isListening ? <Mic className="w-3 h-3 lg:w-4 lg:h-4" /> : <MicOff className="w-3 h-3
lg:w-4 lg:h-4" />}
          </button>

          <button
            onClick={() => setIsSpeaking(!isSpeaking)}
            className={`p-1.5 lg:p-2 rounded-lg transition-colors duration-200 ${
              isSpeaking ? 'bg-cyan-500 text-black' : 'hover:bg-gray-800 text-cyan-300'
            }`}
```

```
        >
          {isSpeaking ? <Volume2 className="w-3 h-3 lg:w-4 lg:h-4" /> : <VolumeX
className="w-3 h-3 lg:w-4 lg:h-4" />}
        </button>

        <button
          onClick={() => setIsMinimized(!isMinimized)}
          className="p-1.5 lg:p-2 hover:bg-gray-800 rounded-lg transition-colors duration-200"
        >
          {isMinimized ? <Maximize2 className="w-3 h-3 lg:w-4 lg:h-4 text-cyan-300" /> :
<Minimize2 className="w-3 h-3 lg:w-4 lg:h-4 text-cyan-300" />}
        </button>

        <button
          onClick={() => setIsOpen(false)}
          className="p-1.5 lg:p-2 hover:bg-gray-800 rounded-lg transition-colors duration-200"
        >
          <X className="w-3 h-3 lg:w-4 lg:h-4 text-cyan-300" />
        </button>
      </div>
    </div>

    {!isMinimized && (
      <>
        {/* Mode Selection */}
        <div className="p-2 lg:p-4 border-b border-cyan-800/30">
          <div className="grid grid-cols-2 lg:grid-cols-4 gap-1 lg:gap-2">
            <ModeButton mode="guidance" icon={Compass} label="Guidance" />
            <ModeButton mode="analysis" icon={TrendingUp} label="Analysis" />
            <ModeButton mode="oracle" icon={Eye} label="Oracle" />
            <ModeButton mode="coaching" icon={Target} label="Coaching" />
          </div>
        </div>

        {/* Current Mode Info */}
        <div className="px-3 lg:px-4 py-2 bg-gray-800/50">
          <div className="flex items-center space-x-2">
            <Sparkles className="w-3 h-3 lg:w-4 lg:h-4 text-cyan-400" />
            <span className="text-cyan-300 text-xs lg:text-sm font-medium capitalize">
              {currentMode} Mode Active
            </span>
            <div className="flex-1"></div>
            <span className="text-cyan-400 text-xs hidden lg:inline">
              {currentMode === 'guidance' && 'Personal navigation & insights'}
```

```
            {currentMode === 'analysis' && 'Field data interpretation'}
            {currentMode === 'oracle' && 'Divination & wisdom'}
            {currentMode === 'coaching' && 'Growth & development'}
          </span>
        </div>
      </div>

      {/* Messages */}
      <div className="flex-1 p-2 lg:p-4 overflow-y-auto max-h-48 lg:max-h-80">
        {conversationHistory.map((message) => (
          <MessageBubble key={message.id} message={message} />
        ))}
        <div ref={messagesEndRef} />
      </div>

      {/* Input Area */}
      <div className="p-2 lg:p-4 border-t border-cyan-800/30">
        <div className="flex space-x-2">
          <div className="flex-1 relative">
            <textarea
              value={inputMessage}
              onChange={(e) => setInputMessage(e.target.value)}
              onKeyPress={handleKeyPress}
              placeholder={
                currentMode === 'guidance' ? 'Ask for guidance...' :
                currentMode === 'analysis' ? 'Request analysis...' :
                currentMode === 'oracle' ? 'Pose your question...' :
                'Share what you\'d like to explore...'
              }
              className="w-full p-2 lg:p-3 pr-10 lg:pr-12 bg-gray-800 border border-cyan-700
rounded-lg text-cyan-100 placeholder-cyan-400 resize-none h-10 lg:h-12 focus:border-teal-400
focus:outline-none text-xs lg:text-sm"
              rows="1"
            />
            <button
              onClick={handleSendMessage}
              disabled={!inputMessage.trim()}
              className="absolute right-1 lg:right-2 top-1 lg:top-2 p-1.5 lg:p-2 bg-gradient-to-r
from-cyan-500 to-teal-500 text-black rounded-lg hover:from-cyan-400 hover:to-teal-400
transition-all duration-300 disabled:opacity-50 disabled:cursor-not-allowed"
            >
              <Send className="w-3 h-3 lg:w-4 lg:h-4" />
            </button>
          </div>
```

```
        </div>

        {/* Quick Suggestions */}
        <div className="flex flex-wrap gap-1 lg:gap-2 mt-2">
          {currentMode === 'guidance' && (
            <>
              <button
                onClick={() => setInputMessage("What does my current field pattern mean?")}
                className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
              >
                Field Pattern
              </button>
              <button
                onClick={() => setInputMessage("How can I improve my coherence?")}
                className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
              >
                Improve Coherence
              </button>
            </>
          )}

          {currentMode === 'oracle' && (
            <>
              <button
                onClick={() => setInputMessage("What is my purpose?")}
                className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
              >
                My Purpose
              </button>
              <button
                onClick={() => setInputMessage("What should I focus on now?")}
                className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
              >
                Current Focus
              </button>
            </>
          )}

          {currentMode === 'analysis' && (
            <>
```

```
                <button
                  onClick={() => setInputMessage("Analyze my Trinity Field balance")}
                  className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
                >
                  Field Balance
                </button>
                <button
                  onClick={() => setInputMessage("Show me my consciousness trends")}
                  className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
                >
                  Trends
                </button>
              </>
            )}

            {currentMode === 'coaching' && (
              <>
                <button
                  onClick={() => setInputMessage("Create a practice for me")}
                  className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
                >
                  Create Practice
                </button>
                <button
                  onClick={() => setInputMessage("Help me with a challenge")}
                  className="text-xs bg-gray-800 text-cyan-300 px-2 py-1 rounded-full
hover:bg-gray-700 transition-colors duration-200"
                >
                  Challenge Help
                </button>
              </>
            )}
          </div>
        </div>
      </>
    )}
  </div>
 );
};

export default StellarAICompanion;
```

```
import React, { useState, useEffect, useRef } from 'react';
import {
  Play, Pause, RotateCw, MapPin, Calendar, Clock, Zap,
  Brain, Heart, Eye, Star, Globe, TrendingUp, Settings,
  ChevronLeft, ChevronRight, Info, Download
} from 'lucide-react';

const StandingWaveSimulator = () => {
  const canvasRef = useRef(null);
  const animationRef = useRef(null);
  const [isPlaying, setIsPlaying] = useState(false);
  const [currentTime, setCurrentTime] = useState(0);
  const [selectedPlanet, setSelectedPlanet] = useState('all');
  const [waveIntensity, setWaveIntensity] = useState(0.8);
  const [showInterference, setShowInterference] = useState(true);
  const [birthLocation, setBirthLocation] = useState({ lat: 36.6777, lng: -121.6555, name:
"Salinas, CA" });
  const [currentLocation, setCurrentLocation] = useState({ lat: 36.6777, lng: -121.6555, name:
"Birth Location" });
  const [fieldShift, setFieldShift] = useState(0);

  // Mock planetary data with frequencies and phases
  const [planets, setPlanets] = useState({
    sun: {
      frequency: 0.1,
      amplitude: 1.0,
      phase: 0,
      color: '#FFA500',
      influence: 'Identity/Ego',
      field: 'spirit'
    },
    moon: {
      frequency: 0.8,
      amplitude: 0.9,
      phase: Math.PI/4,
      color: '#C0C0C0',
      influence: 'Emotions/Instincts',
      field: 'heart'
    },
    mercury: {
      frequency: 0.3,
      amplitude: 0.7,
      phase: Math.PI/3,
```

```
    color: '#00CED1',
    influence: 'Communication/Mind',
    field: 'mind'
  },
  venus: {
    frequency: 0.25,
    amplitude: 0.8,
    phase: Math.PI/2,
    color: '#FF69B4',
    influence: 'Love/Values',
    field: 'heart'
  },
  mars: {
    frequency: 0.15,
    amplitude: 0.85,
    phase: Math.PI/6,
    color: '#FF4500',
    influence: 'Action/Drive',
    field: 'body'
  },
  jupiter: {
    frequency: 0.05,
    amplitude: 1.1,
    phase: Math.PI/5,
    color: '#FFD700',
    influence: 'Expansion/Wisdom',
    field: 'spirit'
  }
});

// Trinity Field calculations based on interference patterns
const [trinityFields, setTrinityFields] = useState({
  mind: { coherence: 85, resonance: 'Active', nodes: [] },
  body: { coherence: 72, resonance: 'Stable', nodes: [] },
  heart: { coherence: 91, resonance: 'Elevated', nodes: [] },
  soul: { coherence: 67, resonance: 'Seeking', nodes: [] },
  spirit: { coherence: 88, resonance: 'Flowing', nodes: [] }
});

// Calculate field shift based on location change
const calculateFieldShift = (birthCoords, newCoords, datetime = new Date()) => {
  const deltaLat = newCoords.lat - birthCoords.lat;
  const deltaLng = newCoords.lng - birthCoords.lng;
```

```
    const geomagneticFactor = Math.sin(deltaLat * Math.PI / 180) + Math.cos(deltaLng * Math.PI
/ 180);
    const temporalFactor = Math.sin((datetime.getTime() / 1e9) % Math.PI);

    return Math.abs(geomagneticFactor * temporalFactor) * 0.5; // Normalize to 0-0.5
  };

  // Standing wave interference calculation
  const calculateInterference = (x, time, planetData) => {
    const { frequency, amplitude, phase } = planetData;
    return amplitude * Math.sin(frequency * x + phase + time * 0.1);
  };

  // Combined wave calculation for all planets
  const calculateCombinedWave = (x, time) => {
    let totalWave = 0;
    let selectedWaves = [];

    Object.entries(planets).forEach(([name, data]) => {
      const wave = calculateInterference(x, time, data);
      if (selectedPlanet === 'all' || selectedPlanet === name) {
        totalWave += wave;
        selectedWaves.push({ name, wave, ...data });
      }
    });

    return { totalWave, selectedWaves };
  };

  // Update Trinity Fields based on standing wave patterns
  const updateTrinityFields = () => {
    const newFields = { ...trinityFields };
    const samplePoints = 50;

    Object.keys(newFields).forEach(field => {
      let fieldResonance = 0;
      let nodeCount = 0;
      let nodes = [];

      for (let i = 0; i < samplePoints; i++) {
        const x = (i / samplePoints) * Math.PI * 4;
        const { totalWave } = calculateCombinedWave(x, currentTime);

        // Apply field shift from travel
```

```
      const adjustedWave = totalWave + (fieldShift * Math.sin(x + currentTime * 0.05));

      fieldResonance += Math.abs(adjustedWave);

      // Detect nodes (near-zero crossings)
      if (Math.abs(adjustedWave) < 0.1) {
        nodeCount++;
        nodes.push(x);
      }
    }

    // Calculate coherence based on field resonance and node stability
    const baseCoherence = Math.min(95, Math.max(45, (fieldResonance / samplePoints) *
100));
    const travelAdjustment = fieldShift * 20; // Travel can affect coherence by ±10%

    newFields[field] = {
      coherence: Math.round(baseCoherence - travelAdjustment),
      resonance: baseCoherence > 80 ? 'Elevated' : baseCoherence > 60 ? 'Active' : 'Seeking',
      nodes: nodes.slice(0, 5) // Keep first 5 nodes
    };
  });

  setTrinityFields(newFields);
};

// Canvas drawing function
const drawWaves = () => {
  const canvas = canvasRef.current;
  if (!canvas) return;

  const ctx = canvas.getContext('2d');
  const width = canvas.width;
  const height = canvas.height;

  // Clear canvas
  ctx.fillStyle = '#000000';
  ctx.fillRect(0, 0, width, height);

  // Draw grid
  ctx.strokeStyle = 'rgba(6, 182, 212, 0.2)';
  ctx.lineWidth = 1;
  for (let i = 0; i <= 10; i++) {
    const x = (i / 10) * width;
```

```
    const y = (i / 10) * height;
    ctx.beginPath();
    ctx.moveTo(x, 0);
    ctx.lineTo(x, height);
    ctx.moveTo(0, y);
    ctx.lineTo(width, y);
    ctx.stroke();
  }

  // Draw individual planetary waves
  Object.entries(planets).forEach(([name, data]) => {
    if (selectedPlanet !== 'all' && selectedPlanet !== name) return;

    ctx.strokeStyle = data.color;
    ctx.lineWidth = selectedPlanet === name ? 3 : 1;
    ctx.globalAlpha = selectedPlanet === name ? 1 : 0.6;

    ctx.beginPath();
    for (let x = 0; x < width; x++) {
      const normalizedX = (x / width) * Math.PI * 4;
      const wave = calculateInterference(normalizedX, currentTime, data);
      const y = height/2 + wave * 50 * waveIntensity;

      if (x === 0) ctx.moveTo(x, y);
      else ctx.lineTo(x, y);
    }
    ctx.stroke();
  });

  // Draw combined interference pattern
  if (showInterference) {
    ctx.strokeStyle = '#22d3ee';
    ctx.lineWidth = 3;
    ctx.globalAlpha = 1;

    ctx.beginPath();
    for (let x = 0; x < width; x++) {
      const normalizedX = (x / width) * Math.PI * 4;
      const { totalWave } = calculateCombinedWave(normalizedX, currentTime);

      // Apply field shift
      const adjustedWave = totalWave + (fieldShift * Math.sin(normalizedX + currentTime *
0.05));
      const y = height/2 + adjustedWave * 30 * waveIntensity;
```

```
        if (x === 0) ctx.moveTo(x, y);
        else ctx.lineTo(x, y);
      }
      ctx.stroke();

      // Draw nodes (interference nulls)
      ctx.fillStyle = '#ef4444';
      for (let x = 0; x < width; x += 5) {
        const normalizedX = (x / width) * Math.PI * 4;
        const { totalWave } = calculateCombinedWave(normalizedX, currentTime);
        const adjustedWave = totalWave + (fieldShift * Math.sin(normalizedX + currentTime *
0.05));

        if (Math.abs(adjustedWave) < 0.2) {
          const y = height/2 + adjustedWave * 30 * waveIntensity;
          ctx.beginPath();
          ctx.arc(x, y, 3, 0, Math.PI * 2);
          ctx.fill();
        }
      }
    }

    ctx.globalAlpha = 1;
  };

  // Animation loop
  const animate = () => {
    if (isPlaying) {
      setCurrentTime(prev => prev + 0.1);
      drawWaves();
      updateTrinityFields();
      animationRef.current = requestAnimationFrame(animate);
    }
  };

  useEffect(() => {
    if (isPlaying) {
      animationRef.current = requestAnimationFrame(animate);
    } else {
      cancelAnimationFrame(animationRef.current);
    }

    return () => cancelAnimationFrame(animationRef.current);
```

```
  }, [isPlaying, currentTime, selectedPlanet, waveIntensity, showInterference, fieldShift]);

  // Update field shift when location changes
  useEffect(() => {
    const shift = calculateFieldShift(birthLocation, currentLocation);
    setFieldShift(shift);
  }, [birthLocation, currentLocation]);

  // Initial draw
  useEffect(() => {
    drawWaves();
    updateTrinityFields();
  }, []);

  const fieldIcons = {
    mind: Brain,
    body: Zap,
    heart: Heart,
    soul: Eye,
    spirit: Star
  };

  return (
    <div className="min-h-screen bg-black text-cyan-100 p-4 lg:p-6">
      <div className="max-w-7xl mx-auto">
        {/* Header */}
        <div className="text-center mb-6 lg:mb-8">
          <h1 className="text-2xl lg:text-4xl font-bold bg-gradient-to-r from-cyan-400 to-teal-400
bg-clip-text text-transparent mb-4">
            Standing Wave Birth Field Simulator
          </h1>
          <p className="text-cyan-300 text-sm lg:text-lg">
            Visualize how planetary interference patterns create your unique consciousness
signature at birth
          </p>
        </div>

        <div className="grid grid-cols-1 xl:grid-cols-4 gap-6">
          {/* Main Wave Display */}
          <div className="xl:col-span-3 space-y-4">
            {/* Controls */}
            <div className="bg-gray-900/50 p-4 rounded-xl border border-cyan-800/30">
              <div className="flex flex-wrap items-center gap-4">
                <button
```

```
            onClick={() => setIsPlaying(!isPlaying)}
            className={`flex items-center space-x-2 px-4 py-2 rounded-lg transition-all
duration-300 ${
              isPlaying
                ? 'bg-red-500 hover:bg-red-600 text-white'
                : 'bg-gradient-to-r from-cyan-500 to-teal-500 hover:from-cyan-400
hover:to-teal-400 text-black'
            }`}
          >
            {isPlaying ? <Pause className="w-4 h-4" /> : <Play className="w-4 h-4" />}
            <span className="font-medium">{isPlaying ? 'Pause' : 'Start'} Simulation</span>
          </button>

          <button
            onClick={() => {
              setCurrentTime(0);
              setIsPlaying(false);
            }}
            className="flex items-center space-x-2 px-4 py-2 bg-gray-800 text-cyan-300
rounded-lg hover:bg-gray-700 transition-colors duration-300"
          >
            <RotateCw className="w-4 h-4" />
            <span>Reset</span>
          </button>

          <div className="flex items-center space-x-2">
            <label className="text-cyan-300 text-sm">Planet Focus:</label>
            <select
              value={selectedPlanet}
              onChange={(e) => setSelectedPlanet(e.target.value)}
              className="bg-gray-800 text-cyan-100 border border-cyan-700 rounded-lg px-3
py-1 text-sm focus:border-teal-400 focus:outline-none"
            >
              <option value="all">All Planets</option>
              {Object.entries(planets).map(([name, data]) => (
                <option key={name} value={name}>{name.charAt(0).toUpperCase() +
name.slice(1)}</option>
              ))}
            </select>
          </div>

          <div className="flex items-center space-x-2">
            <label className="text-cyan-300 text-sm">Intensity:</label>
            <input
```

```
                type="range"
                min="0.1"
                max="2"
                step="0.1"
                value={waveIntensity}
                onChange={(e) => setWaveIntensity(parseFloat(e.target.value))}
                className="w-20 h-2 bg-gray-700 rounded-lg appearance-none cursor-pointer"
              />
              <span className="text-cyan-400 text-sm font-mono">{waveIntensity}x</span>
            </div>

            <label className="flex items-center space-x-2 cursor-pointer">
              <input
                type="checkbox"
                checked={showInterference}
                onChange={(e) => setShowInterference(e.target.checked)}
                className="sr-only"
              />
              <div className={`w-5 h-5 rounded border-2 transition-colors duration-300 ${
                showInterference ? 'bg-cyan-500 border-cyan-500' : 'border-cyan-700'
              }`}>
                {showInterference && (
                  <svg className="w-3 h-3 text-black mx-auto mt-0.5" fill="currentColor"
viewBox="0 0 20 20">
                    <path fillRule="evenodd" d="M16.707 5.293a1 1 0 010 1.414l-8 8a1 1 0 01-1.414
0l-4-4a1 1 0 011.414-1.414L8 12.586l7.293-7.293a1 1 0 011.414 0z" clipRule="evenodd" />
                  </svg>
                )}
              </div>
              <span className="text-cyan-300 text-sm">Show Interference</span>
            </label>
          </div>
        </div>

        {/* Wave Canvas */}
        <div className="bg-gray-900/50 p-4 rounded-xl border border-cyan-800/30">
          <div className="mb-4">
            <h3 className="text-lg font-semibold text-cyan-300 mb-2">Standing Wave
Interference Pattern</h3>
            <p className="text-cyan-400 text-sm">
              {selectedPlanet === 'all' ? 'Combined planetary interference' :
`${selectedPlanet.charAt(0).toUpperCase() + selectedPlanet.slice(1)} wave influence`}
              • Time: {currentTime.toFixed(1)}s
              {fieldShift > 0 && ` • Field Shift: ${(fieldShift * 100).toFixed(1)}%`}
```

```
            </p>
          </div>
          <canvas
            ref={canvasRef}
            width={800}
            height={300}
            className="w-full border border-cyan-800/30 rounded-lg bg-black"
          />
        </div>

        {/* Location Controls */}
        <div className="bg-gray-900/50 p-4 rounded-xl border border-cyan-800/30">
          <h3 className="text-lg font-semibold text-cyan-300 mb-4 flex items-center">
            <MapPin className="w-5 h-5 mr-2" />
            Location Impact on Field Signature
          </h3>
          <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
            <div>
              <label className="block text-cyan-300 text-sm font-semibold mb-2">Birth
Location</label>
              <div className="bg-gray-800 p-3 rounded-lg border border-cyan-700">
                <p className="text-cyan-100 font-medium">{birthLocation.name}</p>
                <p className="text-cyan-400 text-sm font-mono">
                  {birthLocation.lat.toFixed(4)}°, {birthLocation.lng.toFixed(4)}°
                </p>
              </div>
            </div>
            <div>
              <label className="block text-cyan-300 text-sm font-semibold mb-2">Current
Location</label>
              <select
                value={currentLocation.name}
                onChange={(e) => {
                  const locations = {
                    "Birth Location": { lat: 36.6777, lng: -121.6555, name: "Birth Location" },
                    "New York, NY": { lat: 40.7128, lng: -74.0060, name: "New York, NY" },
                    "London, UK": { lat: 51.5074, lng: -0.1278, name: "London, UK" },
                    "Tokyo, Japan": { lat: 35.6762, lng: 139.6503, name: "Tokyo, Japan" },
                    "Sydney, Australia": { lat: -33.8688, lng: 151.2093, name: "Sydney, Australia" }
                  };
                  setCurrentLocation(locations[e.target.value]);
                }}
                className="w-full bg-gray-800 text-cyan-100 border border-cyan-700 rounded-lg
px-3 py-2 focus:border-teal-400 focus:outline-none"
```

```
            >
              <option value="Birth Location">Birth Location</option>
              <option value="New York, NY">New York, NY</option>
              <option value="London, UK">London, UK</option>
              <option value="Tokyo, Japan">Tokyo, Japan</option>
              <option value="Sydney, Australia">Sydney, Australia</option>
            </select>
            {fieldShift > 0 && (
              <div className="mt-2 p-2 bg-yellow-900/30 border border-yellow-700/50
rounded">
                <p className="text-yellow-400 text-sm">
                  <strong>Field Shift Detected:</strong> {(fieldShift * 100).toFixed(1)}% variance
from birth signature
                </p>
              </div>
            )}
          </div>
        </div>
      </div>

      {/* Sidebar */}
      <div className="space-y-4">
        {/* Planetary Data */}
        <div className="bg-gray-900/50 p-4 rounded-xl border border-cyan-800/30">
          <h3 className="text-lg font-semibold text-cyan-300 mb-4">Planetary Oscillators</h3>
          <div className="space-y-3">
            {Object.entries(planets).map(([name, data]) => (
              <div
                key={name}
                className={`p-3 rounded-lg border cursor-pointer transition-all duration-300 ${
                  selectedPlanet === name
                    ? 'border-cyan-400 bg-cyan-900/20'
                    : 'border-gray-700 hover:border-gray-600'
                }`}
                onClick={() => setSelectedPlanet(selectedPlanet === name ? 'all' : name)}
              >
                <div className="flex items-center justify-between mb-2">
                  <span className="text-cyan-100 font-medium capitalize">{name}</span>
                  <div
                    className="w-4 h-4 rounded-full"
                    style={{ backgroundColor: data.color }}
                  ></div>
                </div>
```

```
              <div className="text-xs space-y-1">
                <div className="flex justify-between">
                  <span className="text-cyan-400">Frequency:</span>
                  <span className="text-cyan-300 font-mono">{data.frequency}</span>
                </div>
                <div className="flex justify-between">
                  <span className="text-cyan-400">Amplitude:</span>
                  <span className="text-cyan-300 font-mono">{data.amplitude}</span>
                </div>
                <div className="text-cyan-400 text-xs">{data.influence}</div>
              </div>
            </div>
          ))}
        </div>
      </div>

      {/* Trinity Fields */}
      <div className="bg-gray-900/50 p-4 rounded-xl border border-cyan-800/30">
        <h3 className="text-lg font-semibold text-cyan-300 mb-4">Trinity Field
Coherence</h3>
        <div className="space-y-3">
          {Object.entries(trinityFields).map(([field, data]) => {
            const Icon = fieldIcons[field];
            return (
              <div key={field} className="bg-gray-800/50 p-3 rounded-lg">
                <div className="flex items-center justify-between mb-2">
                  <div className="flex items-center space-x-2">
                    <Icon className="w-4 h-4 text-cyan-400" />
                    <span className="text-cyan-100 font-medium capitalize">{field}</span>
                  </div>
                  <span className="text-cyan-300 font-mono
text-sm">{data.coherence}%</span>
                </div>
                <div className="w-full bg-gray-700
rounded-full h-2">
                  <div
                    className="bg-gradient-to-r
from-cyan-500 to-teal-500 h-2 rounded-full transition-all
duration-1000"
                    style={{ width: `${data.coherence}%`
}}
                  ></div>
                </div>
```

```
                    <div className="text-xs text-cyan-400
mt-1">{data.resonance}</div>
                  </div>
                );
              })}
            </div>
          </div>

          {/* Wave Physics Info */}
          <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
            <h3 className="text-lg font-semibold text-cyan-300
mb-3 flex items-center">
              <Info className="w-4 h-4 mr-2" />
              Wave Physics
            </h3>
            <div className="text-xs space-y-2 text-cyan-400">
              <p><strong>Constructive Interference:</strong>
Waves align, amplifying field strength</p>
              <p><strong>Destructive Interference:</strong>
Waves cancel, creating nodes (red dots)</p>
              <p><strong>Birth Imprint:</strong> Standing wave
pattern frozen at moment of birth</p>
              <p><strong>Travel Effect:</strong> Location
change modulates base frequency</p>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default StandingWaveSimulator;

import React, { useState, useEffect, useRef } from 'react';
import {
  MapPin, Navigation, Clock, Zap, Brain, Heart, Eye, Star,
  Play, Pause, RotateCw, Filter, Layers, TrendingUp,
  Calendar, Users, Target, Compass, Settings, Info,
```

```
  ChevronLeft, ChevronRight, Plus, Minus
} from 'lucide-react';

const FieldResonanceMap = () => {
  const [currentLocation, setCurrentLocation] = useState({ lat:
36.6777, lng: -121.6555, name: "Salinas, CA" });
  const [birthLocation] = useState({ lat: 36.6777, lng:
-121.6555, name: "Birth Location" });
  const [mapMode, setMapMode] = useState('field-coherence'); //
field-coherence, astrocartography, timeline, missions
  const [timelinePosition, setTimelinePosition] = useState(100);
// 0-100%
  const [isPlaying, setIsPlaying] = useState(false);
  const [selectedField, setSelectedField] = useState('all');
  const [showPlanetaryLines, setShowPlanetaryLines] =
useState(true);
  const [zoomLevel, setZoomLevel] = useState(4);

  // Mock user movement timeline with field data
  const [movementHistory] = useState([
    {
      id: 1,
      location: { lat: 36.6777, lng: -121.6555, name: "Salinas,
CA" },
      date: "2024-01-01",
      duration: 30, // days
      fieldData: {
        mind: { coherence: 85, resonance: 'Active', events:
['Birth Chart Generated'] },
        body: { coherence: 72, resonance: 'Stable', events:
['Started Meditation'] },
        heart: { coherence: 91, resonance: 'Elevated', events:
['Met Soulmate'] },
        soul: { coherence: 67, resonance: 'Seeking', events:
['Oracle Consultation'] },
        spirit: { coherence: 88, resonance: 'Flowing', events:
['Spiritual Awakening'] }
      },
      glyphHits: 12,
      synchronicities: 3,
      emotionalState: 'expansive'
```

```
    },
    {
      id: 2,
      location: { lat: 37.7749, lng: -122.4194, name: "San
Francisco, CA" },
      date: "2024-02-01",
      duration: 45,
      fieldData: {
        mind: { coherence: 92, resonance: 'Elevated', events:
['Creative Breakthrough'] },
        body: { coherence: 68, resonance: 'Seeking', events:
['Health Challenge'] },
        heart: { coherence: 75, resonance: 'Active', events:
['Relationship Shift'] },
        soul: { coherence: 88, resonance: 'Flowing', events:
['Purpose Clarity'] },
        spirit: { coherence: 95, resonance: 'Transcendent',
events: ['Mystical Experience'] }
      },
      glyphHits: 18,
      synchronicities: 7,
      emotionalState: 'creative'
    },
    {
      id: 3,
      location: { lat: 40.7128, lng: -74.0060, name: "New York,
NY" },
      date: "2024-03-15",
      duration: 20,
      fieldData: {
        mind: { coherence: 78, resonance: 'Active', events:
['Information Overload'] },
        body: { coherence: 85, resonance: 'Elevated', events:
['Physical Vitality'] },
        heart: { coherence: 62, resonance: 'Seeking', events:
['Emotional Processing'] },
        soul: { coherence: 71, resonance: 'Active', events:
['Urban Awakening'] },
        spirit: { coherence: 55, resonance: 'Challenged',
events: ['Spiritual Disconnect'] }
      },
```

```
      glyphHits: 8,
      synchronicities: 2,
      emotionalState: 'intense'
    },
    {
      id: 4,
      location: { lat: 51.5074, lng: -0.1278, name: "London, UK"
},
      date: "2024-04-10",
      duration: 25,
      fieldData: {
        mind: { coherence: 89, resonance: 'Elevated', events:
['Ancient Wisdom Access'] },
        body: { coherence: 73, resonance: 'Stable', events:
['Adjustment Period'] },
        heart: { coherence: 94, resonance: 'Transcendent',
events: ['Soul Recognition'] },
        soul: { coherence: 91, resonance: 'Elevated', events:
['Past Life Memories'] },
        spirit: { coherence: 87, resonance: 'Flowing', events:
['Sacred Sites Visit'] }
      },
      glyphHits: 22,
      synchronicities: 9,
      emotionalState: 'mystical'
    }
  ]);

  // Planetary line data for astrocartography
  const [planetaryLines] = useState({
    sun: {
      mc: [{ lat: 40, lng: -100 }, { lat: 45, lng: -80 }],
      ic: [{ lat: 35, lng: -120 }, { lat: 30, lng: -90 }],
      color: '#FFA500',
      influence: 'Identity & Purpose'
    },
    moon: {
      mc: [{ lat: 50, lng: -110 }, { lat: 55, lng: -70 }],
      ic: [{ lat: 25, lng: -130 }, { lat: 20, lng: -100 }],
      color: '#C0C0C0',
      influence: 'Emotions & Home'
```

```
    },
    venus: {
      mc: [{ lat: 45, lng: -105 }, { lat: 48, lng: -75 }],
      ic: [{ lat: 30, lng: -125 }, { lat: 25, lng: -95 }],
      color: '#FF69B4',
      influence: 'Love & Creativity'
    },
    jupiter: {
      mc: [{ lat: 42, lng: -108 }, { lat: 47, lng: -78 }],
      ic: [{ lat: 32, lng: -122 }, { lat: 27, lng: -92 }],
      color: '#FFD700',
      influence: 'Growth & Expansion'
    }
  });

  // Current timeline entry based on timeline position
  const getCurrentTimelineEntry = () => {
    const index = Math.floor((timelinePosition / 100) *
(movementHistory.length - 1));
    return movementHistory[index];
  };

  // Calculate field coherence color
  const getFieldColor = (coherence, field = 'all') => {
    const fieldColors = {
      mind: { low: '#0891b2', high: '#22d3ee' },
      body: { low: '#0f766e', high: '#2dd4bf' },
      heart: { low: '#059669', high: '#34d399' },
      soul: { low: '#0284c7', high: '#38bdf8' },
      spirit: { low: '#7c3aed', high: '#a78bfa' },
      all: { low: '#374151', high: '#22d3ee' }
    };

    const colors = fieldColors[field] || fieldColors.all;
    const intensity = coherence / 100;

    // Interpolate between low and high colors
    return intensity > 0.7 ? colors.high :
           intensity > 0.4 ? '#14b8a6' : colors.low;
  };
```

```
  // Map viewport bounds
  const mapBounds = {
    north: 60,
    south: 20,
    east: 10,
    west: -140
  };

  // Convert lat/lng to pixel coordinates
  const coordsToPixels = (lat, lng, mapWidth = 800, mapHeight =
400) => {
    const x = ((lng - mapBounds.west) / (mapBounds.east -
mapBounds.west)) * mapWidth;
    const y = ((mapBounds.north - lat) / (mapBounds.north -
mapBounds.south)) * mapHeight;
    return { x, y };
  };

  // Timeline animation
  useEffect(() => {
    let interval;
    if (isPlaying) {
      interval = setInterval(() => {
        setTimelinePosition(prev => {
          if (prev >= 100) {
            setIsPlaying(false);
            return 100;
          }
          return prev + 1;
        });
      }, 100);
    }
    return () => clearInterval(interval);
  }, [isPlaying]);

  const fieldIcons = {
    mind: Brain,
    body: Zap,
    heart: Heart,
    soul: Eye,
    spirit: Star
```

```
  };

  const currentEntry = getCurrentTimelineEntry();

  return (
    <div className="min-h-screen bg-black text-cyan-100 p-4
lg:p-6">
      <div className="max-w-7xl mx-auto">
        {/* Header */}
        <div className="text-center mb-6">
          <h1 className="text-2xl lg:text-4xl font-bold
bg-gradient-to-r from-cyan-400 to-teal-400 bg-clip-text
text-transparent mb-4">
            Field Resonance Map
          </h1>
          <p className="text-cyan-300 text-sm lg:text-lg">
            Track consciousness coherence across geographic
locations and time
          </p>
        </div>

        <div className="grid grid-cols-1 xl:grid-cols-4 gap-6">
          {/* Main Map */}
          <div className="xl:col-span-3 space-y-4">
            {/* Map Controls */}
            <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
              <div className="flex flex-wrap items-center
gap-4">
                <div className="flex items-center space-x-2">
                  <label className="text-cyan-300 text-sm">Map
Mode:</label>
                  <select
                    value={mapMode}
                    onChange={(e) => setMapMode(e.target.value)}
                    className="bg-gray-800 text-cyan-100 border
border-cyan-700 rounded-lg px-3 py-1 text-sm
focus:border-teal-400 focus:outline-none"
                  >
                    <option value="field-coherence">Field
Coherence</option>
```

```
                    <option
value="astrocartography">Astrocartography Lines</option>
                    <option value="timeline">Timeline
Journey</option>
                    <option value="missions">Consciousness
Missions</option>
                  </select>
                </div>

                <div className="flex items-center space-x-2">
                  <label className="text-cyan-300 text-sm">Field
Focus:</label>
                  <select
                    value={selectedField}
                    onChange={(e) =>
setSelectedField(e.target.value)}
                    className="bg-gray-800 text-cyan-100 border
border-cyan-700 rounded-lg px-3 py-1 text-sm
focus:border-teal-400 focus:outline-none"
                  >
                    <option value="all">All Fields</option>
                    <option value="mind">Mind</option>
                    <option value="body">Body</option>
                    <option value="heart">Heart</option>
                    <option value="soul">Soul</option>
                    <option value="spirit">Spirit</option>
                  </select>
                </div>

                <label className="flex items-center space-x-2
cursor-pointer">
                  <input
                    type="checkbox"
                    checked={showPlanetaryLines}
                    onChange={(e) =>
setShowPlanetaryLines(e.target.checked)}
                    className="sr-only"
                  />
                  <div className={`w-5 h-5 rounded border-2
transition-colors duration-300 ${
```

```
                    showPlanetaryLines ? 'bg-cyan-500
border-cyan-500' : 'border-cyan-700'
                  }`}>
                    {showPlanetaryLines && (
                      <svg className="w-3 h-3 text-black mx-auto
mt-0.5" fill="currentColor" viewBox="0 0 20 20">
                        <path fillRule="evenodd" d="M16.707
5.293a1 1 0 010 1.414l-8 8a1 1 0 01-1.414 0l-4-4a1 1 0
011.414-1.414L8 12.586l7.293-7.293a1 1 0 011.414 0z"
clipRule="evenodd" />
                      </svg>
                    )}
                  </div>
                  <span className="text-cyan-300
text-sm">Planetary Lines</span>
                </label>

                <div className="flex items-center space-x-2">
                  <button
                    onClick={() => setZoomLevel(Math.max(1,
zoomLevel - 1))}
                    className="p-2 bg-gray-800 text-cyan-300
rounded-lg hover:bg-gray-700 transition-colors duration-300"
                  >
                    <Minus className="w-4 h-4" />
                  </button>
                  <span className="text-cyan-300 text-sm
font-mono">{zoomLevel}x</span>
                  <button
                    onClick={() => setZoomLevel(Math.min(10,
zoomLevel + 1))}
                    className="p-2 bg-gray-800 text-cyan-300
rounded-lg hover:bg-gray-700 transition-colors duration-300"
                  >
                    <Plus className="w-4 h-4" />
                  </button>
                </div>
              </div>
            </div>

            {/* Timeline Controls */}
```

```
            <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
              <div className="flex items-center space-x-4 mb-4">
                <button
                  onClick={() => setIsPlaying(!isPlaying)}
                  className={`flex items-center space-x-2 px-4
py-2 rounded-lg transition-all duration-300 ${
                    isPlaying
                      ? 'bg-red-500 hover:bg-red-600 text-white'
                      : 'bg-gradient-to-r from-cyan-500
to-teal-500 hover:from-cyan-400 hover:to-teal-400 text-black'
                  }`}
                >
                  {isPlaying ? <Pause className="w-4 h-4" /> :
<Play className="w-4 h-4" />}
                  <span className="font-medium">{isPlaying ?
'Pause' : 'Play'} Journey</span>
                </button>

                <button
                  onClick={() => {
                    setTimelinePosition(0);
                    setIsPlaying(false);
                  }}
                  className="flex items-center space-x-2 px-4
py-2 bg-gray-800 text-cyan-300 rounded-lg hover:bg-gray-700
transition-colors duration-300"
                >
                  <RotateCw className="w-4 h-4" />
                  <span>Reset</span>
                </button>

                <div className="flex items-center space-x-2
flex-1">
                  <Clock className="w-4 h-4 text-cyan-400" />
                  <span className="text-cyan-300
text-sm">{currentEntry.date}</span>
                  <span className="text-cyan-400
text-sm">•</span>
                  <span className="text-cyan-300
text-sm">{currentEntry.location.name}</span>
```

```
                </div>
              </div>

              <div className="space-y-2">
                <input
                  type="range"
                  min="0"
                  max="100"
                  value={timelinePosition}
                  onChange={(e) =>
setTimelinePosition(parseInt(e.target.value))}
                  className="w-full h-2 bg-gray-700 rounded-lg
appearance-none cursor-pointer"
                />
                <div className="flex justify-between text-xs
text-cyan-400">
                  {movementHistory.map((entry, index) => (
                    <span key={entry.id}
className="text-center">
                      {entry.location.name.split(',')[0]}
                    </span>
                  ))}
                </div>
              </div>
            </div>

            {/* Map Canvas */}
            <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
              <div className="relative">
                <svg
                  width="100%"
                  height="500"
                  viewBox="0 0 800 400"
                  className="border border-cyan-800/30
rounded-lg bg-gradient-to-b from-gray-900 to-black"
                >
                  {/* World Map Outline */}
                  <defs>
                    <pattern id="grid" width="40" height="40"
patternUnits="userSpaceOnUse">
```

```
                      <path d="M 40 0 L 0 0 0 40" fill="none"
stroke="rgba(6, 182, 212, 0.1)" strokeWidth="1"/>
                    </pattern>
                  </defs>
                  <rect width="800" height="400"
fill="url(#grid)" />

                  {/* Continents (simplified outlines) */}
                  <path d="M 150 150 Q 200 120 250 140 Q 300 130
350 150 Q 400 160 450 140 Q 500 150 550 160 L 550 220 Q 500 230
450 220 Q 400 240 350 230 Q 300 220 250 230 Q 200 240 150 220 Z"
                        fill="rgba(6, 182, 212, 0.1)"
stroke="rgba(6, 182, 212, 0.3)" strokeWidth="1" />
                  <path d="M 600 180 Q 650 170 700 180 Q 750 190
780 200 L 780 260 Q 750 270 700 260 Q 650 250 600 260 Z"
                        fill="rgba(6, 182, 212, 0.1)"
stroke="rgba(6, 182, 212, 0.3)" strokeWidth="1" />

                  {/* Planetary Lines */}
                  {showPlanetaryLines &&
Object.entries(planetaryLines).map(([planet, data]) => (
                    <g key={planet}>
                      {/* MC Line */}
                      {data.mc.map((point, index) => {
                        if (index === 0) return null;
                        const start =
coordsToPixels(data.mc[index - 1].lat, data.mc[index - 1].lng);
                        const end = coordsToPixels(point.lat,
point.lng);
                        return (
                          <line
                            key={`${planet}-mc-${index}`}
                            x1={start.x}
                            y1={start.y}
                            x2={end.x}
                            y2={end.y}
                            stroke={data.color}
                            strokeWidth="2"
                            strokeDasharray="5,5"
                            opacity="0.7"
                          />
```

```
                        );
                      })}
                      {/* IC Line */}
                      {data.ic.map((point, index) => {
                        if (index === 0) return null;
                        const start =
coordsToPixels(data.ic[index - 1].lat, data.ic[index - 1].lng);
                        const end = coordsToPixels(point.lat,
point.lng);
                        return (
                          <line
                            key={`${planet}-ic-${index}`}
                            x1={start.x}
                            y1={start.y}
                            x2={end.x}
                            y2={end.y}
                            stroke={data.color}
                            strokeWidth="2"
                            strokeDasharray="2,3"
                            opacity="0.5"
                          />
                        );
                      })}
                    </g>
                  ))}
{/* Arrow marker definition */}
                  <defs>
                    <marker id="arrowhead" markerWidth="10"
markerHeight="7" refX="9" refY="3.5" orient="auto">
                      <polygon points="0 0, 10 3.5, 0 7"
fill="#14b8a6" />
                    </marker>
                  </defs>
                </svg>
              </div>
            </div>

            {/* Current Location Details */}
            <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
```

```
              <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                <MapPin className="w-5 h-5 mr-2" />
                Current Location: {currentEntry.location.name}
              </h3>
              <div className="grid grid-cols-1 md:grid-cols-3
gap-4">
                <div>
                  <h4 className="text-cyan-400 font-semibold
mb-2">Field Coherence</h4>
                  <div className="space-y-2">

{Object.entries(currentEntry.fieldData).map(([field, data]) => {
                      const Icon = fieldIcons[field];
                      return (
                        <div key={field} className="flex
items-center justify-between">
                          <div className="flex items-center
space-x-2">
                            <Icon className="w-4 h-4
text-cyan-400" />
                            <span className="text-cyan-100
capitalize text-sm">{field}</span>
                          </div>
                          <span className="text-cyan-300
font-mono text-sm">{data.coherence}%</span>
                        </div>
                      );
                    })}
                  </div>
                </div>

                <div>
                  <h4 className="text-cyan-400 font-semibold
mb-2">Resonance Metrics</h4>
                  <div className="space-y-2 text-sm">
                    <div className="flex justify-between">
                      <span className="text-cyan-300">Glyph
Hits:</span>
                      <span className="text-teal-400
font-bold">{currentEntry.glyphHits}</span>
```

```
                    </div>
                    <div className="flex justify-between">
                      <span
className="text-cyan-300">Synchronicities:</span>
                      <span className="text-yellow-400
font-bold">{currentEntry.synchronicities}</span>
                    </div>
                    <div className="flex justify-between">
                      <span
className="text-cyan-300">Duration:</span>
                      <span
className="text-cyan-100">{currentEntry.duration} days</span>
                    </div>
                    <div className="flex justify-between">
                      <span className="text-cyan-300">Emotional
State:</span>
                      <span className="text-emerald-400
capitalize">{currentEntry.emotionalState}</span>
                    </div>
                  </div>
                </div>

                <div>
                  <h4 className="text-cyan-400 font-semibold
mb-2">Key Events</h4>
                  <div className="space-y-1">

{Object.values(currentEntry.fieldData).flatMap(field =>
field.events).slice(0, 4).map((event, index) => (
                      <div key={index} className="text-xs
text-cyan-300 bg-gray-800/50 px-2 py-1 rounded">
                        {event}
                      </div>
                    ))}
                  </div>
                </div>
              </div>
            </div>

            {/* Sidebar */}
```

```
          <div className="space-y-4">
            {/* Map Legend */}
            <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4">Map Legend</h3>
              <div className="space-y-3 text-sm">
                <div className="flex items-center space-x-2">
                  <div className="w-4 h-4 bg-cyan-400
rounded-full opacity-40"></div>
                  <span className="text-cyan-300">Field
Coherence Halo</span>
                </div>
                <div className="flex items-center space-x-2">
                  <div className="w-2 h-2 bg-yellow-400
rounded-full"></div>
                  <span className="text-cyan-300">Synchronicity
Events</span>
                </div>
                <div className="flex items-center space-x-2">
                  <div className="w-8 h-1 bg-teal-400"></div>
                  <span className="text-cyan-300">Journey
Path</span>
                </div>
                <div className="flex items-center space-x-2">
                  <div className="w-8 h-1 border
border-orange-400 border-dashed"></div>
                  <span className="text-cyan-300">Planetary
Lines</span>
                </div>
              </div>
            </div>

            {/* Planetary Lines Info */}
            {showPlanetaryLines && (
              <div className="bg-gray-900/50 p-4 rounded-xl
border border-cyan-800/30">
                <h3 className="text-lg font-semibold
text-cyan-300 mb-4">Astrocartography Lines</h3>
                <div className="space-y-3">
```

```
                  {Object.entries(planetaryLines).map(([planet,
data]) => (
                    <div key={planet} className="space-y-1">
                      <div className="flex items-center
space-x-2">
                        <div
                          className="w-3 h-3 rounded-full"
                          style={{ backgroundColor: data.color
}}
                        ></div>
                        <span className="text-cyan-100
font-medium capitalize">{planet}</span>
                      </div>
                      <p className="text-xs text-cyan-400
ml-5">{data.influence}</p>
                    </div>
                  ))}
                </div>
              </div>
            )}

            {/* Field Analytics */}
            <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                <TrendingUp className="w-5 h-5 mr-2" />
                Field Analytics
              </h3>
              <div className="space-y-3">
                {/* Overall Journey Stats */}
                <div className="bg-gray-800/50 p-3 rounded-lg">
                  <h4 className="text-cyan-400 font-semibold
mb-2 text-sm">Journey Overview</h4>
                  <div className="grid grid-cols-2 gap-2
text-xs">
                    <div className="text-center">
                      <div className="text-lg font-bold
text-cyan-100">{movementHistory.length}</div>
                      <div
className="text-cyan-400">Locations</div>
```

```
                    </div>
                    <div className="text-center">
                      <div className="text-lg font-bold
text-teal-400">
                        {movementHistory.reduce((sum, entry) =>
sum + entry.glyphHits, 0)}
                      </div>
                      <div className="text-cyan-400">Total Glyph
Hits</div>
                    </div>
                    <div className="text-center">
                      <div className="text-lg font-bold
text-yellow-400">
                        {movementHistory.reduce((sum, entry) =>
sum + entry.synchronicities, 0)}
                      </div>
                      <div
className="text-cyan-400">Synchronicities</div>
                    </div>
                    <div className="text-center">
                      <div className="text-lg font-bold
text-emerald-400">
                        {Math.round(movementHistory.reduce((sum,
entry) =>
                          sum +
Object.values(entry.fieldData).reduce((fieldSum, field) =>
fieldSum + field.coherence, 0) / 5, 0
                        ) / movementHistory.length)}%
                      </div>
                      <div className="text-cyan-400">Avg
Coherence</div>
                    </div>
                  </div>
                </div>

                {/* Peak Performance Locations */}
                <div className="bg-gray-800/50 p-3 rounded-lg">
                  <h4 className="text-cyan-400 font-semibold
mb-2 text-sm">Peak Resonance Zones</h4>
                  <div className="space-y-2">
                    {movementHistory
```

```
                        .sort((a, b) => b.glyphHits - a.glyphHits)
                        .slice(0, 3)
                        .map((entry, index) => (
                          <div key={entry.id} className="flex
items-center justify-between text-xs">
                            <span
className="text-cyan-300">{entry.location.name.split(',')[0]}</s
pan>
                            <div className="flex items-center
space-x-2">
                              <span className="text-teal-400
font-mono">{entry.glyphHits}</span>
                              <div className="flex space-x-1">
                                {Array.from({ length:
entry.synchronicities }).map((_, i) => (
                                  <div key={i} className="w-1 h-1
bg-yellow-400 rounded-full"></div>
                                ))}
                              </div>
                            </div>
                          ))}
                      </div>
                    </div>

                    {/* Field Progression */}
                    <div className="bg-gray-800/50 p-3 rounded-lg">
                      <h4 className="text-cyan-400 font-semibold
mb-2 text-sm">Field Evolution</h4>
                      <div className="space-y-2">
                        {Object.keys(fieldIcons).map(field => {
                          const Icon = fieldIcons[field];
                          const startCoherence =
movementHistory[0].fieldData[field].coherence;
                          const currentCoherence =
currentEntry.fieldData[field].coherence;
                          const change = currentCoherence -
startCoherence;

                          return (
```

```
                        <div key={field} className="flex
items-center justify-between text-xs">
                          <div className="flex items-center
space-x-2">
                            <Icon className="w-3 h-3
text-cyan-400" />
                            <span className="text-cyan-300
capitalize">{field}</span>
                          </div>
                          <div className="flex items-center
space-x-2">
                            <span className="text-cyan-100
font-mono">{currentCoherence}%</span>
                            <span className={`font-mono text-xs
${
                              change > 0 ? 'text-green-400' :
change < 0 ? 'text-red-400' : 'text-gray-400'
                            }`}>
                              {change > 0 ? '+' : ''}{change}
                            </span>
                          </div>
                        </div>
                      );
                    })}
                  </div>
                </div>
              </div>

              {/* Mission Suggestions */}
              <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
                <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                  <Target className="w-5 h-5 mr-2" />
                  Consciousness Missions
                </h3>
                <div className="space-y-3">
                  <div className="bg-gradient-to-r
from-cyan-500/20 to-teal-500/20 p-3 rounded-lg border
border-cyan-500/30">
```

```
                  <h4 className="text-cyan-100 font-semibold
text-sm mb-1">Active Mission</h4>
                  <p className="text-cyan-300 text-xs
mb-2">Heart Field Integration Challenge</p>
                  <div className="w-full bg-gray-700
rounded-full h-2">
                    <div className="bg-gradient-to-r
from-cyan-500 to-teal-500 h-2 rounded-full" style={{ width:
'65%' }}></div>
                  </div>
                  <p className="text-cyan-400 text-xs
mt-1">Progress: 65% • 3 days remaining</p>
                </div>

                <div className="space-y-2">
                  <div className="bg-gray-800/50 p-3
rounded-lg">
                    <h4 className="text-emerald-400
font-semibold text-sm">Available Mission</h4>
                    <p className="text-cyan-300 text-xs">Soul
Echo Navigator - Explore past-life resonances in sacred
locations</p>
                    <button className="mt-2 bg-gradient-to-r
from-emerald-500 to-teal-500 text-black text-xs font-semibold
px-3 py-1 rounded hover:from-emerald-400 hover:to-teal-400
transition-all duration-300">
                      Accept Mission
                    </button>
                  </div>

                  <div className="bg-gray-800/50 p-3
rounded-lg">
                    <h4 className="text-yellow-400 font-semibold
text-sm">Location Quest</h4>
                    <p className="text-cyan-300 text-xs">Jupiter
Line Activation - Travel to your Jupiter MC line for expansion
work</p>
                    <button className="mt-2 bg-gradient-to-r
from-yellow-500 to-orange-500 text-black text-xs font-semibold
px-3 py-1 rounded hover:from-yellow-400 hover:to-orange-400
transition-all duration-300">
```

```
                      Plan Journey
                    </button>
                  </div>
                </div>
              </div>

              {/* Export Options */}
              <div className="bg-gray-900/50 p-4 rounded-xl border
border-cyan-800/30">
                <h3 className="text-lg font-semibold text-cyan-300
mb-4">Export & Share</h3>
                <div className="space-y-2">
                  <button className="w-full bg-gradient-to-r
from-cyan-500 to-teal-500 text-black font-semibold py-2 px-4
rounded-lg hover:from-cyan-400 hover:to-teal-400 transition-all
duration-300 text-sm">
                    Export Journey Report
                  </button>
                  <button className="w-full bg-gradient-to-r
from-teal-500 to-emerald-500 text-black font-semibold py-2 px-4
rounded-lg hover:from-teal-400 hover:to-emerald-400
transition-all duration-300 text-sm">
                    Share Resonance Map
                  </button>
                  <button className="w-full bg-gradient-to-r
from-emerald-500 to-blue-500 text-black font-semibold py-2 px-4
rounded-lg hover:from-emerald-400 hover:to-blue-400
transition-all duration-300 text-sm">
                    Generate Field Calendar
                  </button>
                </div>
              </div>
            </div>
          </div>

          {/* Bottom Info Panel */}
          <div className="mt-6 bg-gray-900/50 p-4 rounded-xl
border border-cyan-800/30">
            <div className="grid grid-cols-1 md:grid-cols-3 gap-4
text-sm">
```

```
            <div>
              <h4 className="text-cyan-400 font-semibold mb-2
flex items-center">
                <Info className="w-4 h-4 mr-2" />
                Field Resonance Theory
              </h4>
              <p className="text-cyan-300 text-xs
leading-relaxed">
                Your consciousness signature is a standing wave
pattern created by planetary interference at birth.
                Travel shifts this pattern, creating new
resonance zones and affecting field coherence across all Trinity
Fields.
              </p>
            </div>

            <div>
              <h4 className="text-cyan-400 font-semibold mb-2
flex items-center">
                <Compass className="w-4 h-4 mr-2" />
                Astrocartography Integration
              </h4>
              <p className="text-cyan-300 text-xs
leading-relaxed">
                Planetary lines show where celestial energies
concentrate on Earth.
                MC lines amplify planetary traits, while IC
lines provide grounding and foundation energy for
transformation.
              </p>
            </div>

            <div>
              <h4 className="text-cyan-400 font-semibold mb-2
flex items-center">
                <Users className="w-4 h-4 mr-2" />
                Collective Resonance
              </h4>
              <p className="text-cyan-300 text-xs
leading-relaxed">
```

```
                Locations with high synchronicity rates indicate
collective consciousness nodes.
                These zones often align with sacred sites, ley
lines, and areas of heightened field activity.
              </p>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default FieldResonanceMap;
import React, { useState, useEffect } from 'react';
import {
  FlaskConical, Target, Clock, TrendingUp, CheckCircle2,
XCircle,
  Play, Pause, RotateCw, Download, Share, Brain, Heart, Eye,
Star,
  Zap, Globe, Users, Compass, ChevronRight, ChevronDown, Info,
  Calendar, MapPin, Activity, Sparkles, TestTube, FileText,
  BarChart3, PieChart, LineChart, AlertTriangle, Award
} from 'lucide-react';

const TheoryTestingDashboard = () => {
  const [activeCategory, setActiveCategory] =
useState('waveform-will');
  const [selectedClaim, setSelectedClaim] = useState(null);
  const [testResults, setTestResults] = useState({});
  const [runningTests, setRunningTests] = useState({});
  const [expandedClaims, setExpandedClaims] = useState({});

  // Theory categories and claims
  const theories = {
    'waveform-will': {
      name: 'Waveform Will Theory',
      icon: Brain,
      color: 'cyan',
      description: 'The will is a field-based harmonic vector
that creates standing wave patterns',
```

```
      claims: [
        {
          id: 'ww-1',
          claim: 'Birth locks in a personal standing wave
field',
          hypothesis: 'Show that glyphs matching birth gates
produce stronger resonance responses than random ones',
          testMethod: 'Field Tone Scan',
          difficulty: 'Beginner',
          duration: '15 min',
          participants: 1,
          status: 'completed',
          confidence: 87,
          trials: 156,
          significance: 0.03
        },
        {
          id: 'ww-2',
          claim: 'The will is a field-based harmonic vector',
          hypothesis: 'Track behavioral choices and life
patterns aligning with defined gates, and compare to coherence
metrics',
          testMethod: 'Coherence Challenge',
          difficulty: 'Intermediate',
          duration: '30 days',
          participants: 1,
          status: 'running',
          confidence: 65,
          trials: 23,
          significance: null
        },
        {
          id: 'ww-3',
          claim: 'Coherence arises from alignment with
waveform',
          hypothesis: 'Measure HRV/emotion before & after
field-aligned actions',
          testMethod: 'Heart Vector Simulation',
          difficulty: 'Advanced',
          duration: '45 min',
          participants: 1,
```

```
          status: 'available',
          confidence: null,
          trials: 0,
          significance: null
        },
        {
          id: 'ww-4',
          claim: 'Glyphs modulate the user\'s standing wave',
          hypothesis: 'Show significant change in emotional/bio
response from aligned vs unaligned glyphs',
          testMethod: 'Gate Amplification Test',
          difficulty: 'Intermediate',
          duration: '20 min',
          participants: 1,
          status: 'completed',
          confidence: 92,
          trials: 89,
          significance: 0.008
        },
        {
          id: 'ww-5',
          claim: 'Life is a harmonic interference experiment',
          hypothesis: 'Log long-term coherence drift and
realignments, then show predictable effects of harmonic
choices',
          testMethod: 'Waveform Drift Logger',
          difficulty: 'Expert',
          duration: '6 months',
          participants: 1,
          status: 'running',
          confidence: 43,
          trials: 8,
          significance: null
        }
      ]
    },
    'stellar-proximology': {
      name: 'Stellar Proximology',
      icon: Star,
      color: 'teal',
```

```
      description: 'Planetary proximity affects consciousness
imprint and field resonance',
      claims: [
        {
          id: 'sp-1',
          claim: 'Planetary proximity affects consciousness
imprint',
          hypothesis: 'Track coherence or glyph accuracy during
perigee vs apogee events',
          testMethod: 'Proximity Gate Tracker',
          difficulty: 'Intermediate',
          duration: '60 days',
          participants: 1,
          status: 'available',
          confidence: null,
          trials: 0,
          significance: null
        },
        {
          id: 'sp-2',
          claim: 'Non-visible cosmic bodies affect the field',
          hypothesis: 'Log coherence spikes during
Sirius/Galactic Center alignments',
          testMethod: 'Stellar Drift Logger',
          difficulty: 'Advanced',
          duration: '1 year',
          participants: 1,
          status: 'running',
          confidence: 38,
          trials: 12,
          significance: null
        },
        {
          id: 'sp-3',
          claim: 'Soul memory emerges from stellar proximity',
          hypothesis: 'Show strong resonance to glyphs found in
soul-level (Draconic or ancestral) charts',
          testMethod: 'Soul Field Analyzer',
          difficulty: 'Expert',
          duration: '90 days',
          participants: 1,
```

```
          status: 'completed',
          confidence: 78,
          trials: 45,
          significance: 0.02
        },
        {
          id: 'sp-4',
          claim: 'Convergence zones exist on Earth',
          hypothesis: 'Show coherence/glyph spikes in specific
geographical locations',
          testMethod: 'Convergence Zone Detector',
          difficulty: 'Advanced',
          duration: 'Ongoing',
          participants: 'Multiple',
          status: 'running',
          confidence: 71,
          trials: 134,
          significance: 0.045
        },
        {
          id: 'sp-5',
          claim: 'Travel causes resonance drift from birth
field',
          hypothesis: 'Compare glyph accuracy before and after
moving 500+ miles from birth location',
          testMethod: 'Location Coherence Test',
          difficulty: 'Beginner',
          duration: '7 days',
          participants: 1,
          status: 'completed',
          confidence: 94,
          trials: 78,
          significance: 0.001
        }
      ]},
    'soul-spirit': {
      name: 'Soul & Spirit Field Theory',
      icon: Eye,
      color: 'emerald',
      description: 'Consciousness exists as nested wave fields
across multiple dimensions',
```

```
      claims: [
        {
          id: 'ss-1',
          claim: 'The soul is harmonic memory across lifetimes',
          hypothesis: 'Show glyph resonance accuracy for gates
not present in this life but found in Draconic/past charts',
          testMethod: 'Harmonic Memory Scan',
          difficulty: 'Expert',
          duration: '6 months',
          participants: 1,
          status: 'running',
          confidence: 56,
          trials: 28,
          significance: null
        },
        {
          id: 'ss-2',
          claim: 'The spirit is a directional coherence
attractor',
          hypothesis: 'Track repeated themes, actions, and
locations pulling user toward specific coherence fields',
          testMethod: 'Spirit Vector Tracker',
          difficulty: 'Advanced',
          duration: '1 year',
          participants: 1,
          status: 'available',
          confidence: null,
          trials: 0,
          significance: null
        },
        {
          id: 'ss-3',
          claim: 'Humans are nested wave fields (Mind, Heart,
etc.)',
          hypothesis: 'Flash glyphs by field type and log
distinct physiological/emotional response signatures',
          testMethod: 'MultiNode Response Test',
          difficulty: 'Intermediate',
          duration: '30 min',
          participants: 1,
          status: 'completed',
```

```
          confidence: 85,
          trials: 67,
          significance: 0.015
        }
      ]
    },
    'octave-threshold': {
      name: 'Octave Threshold Law',
      icon: Compass,
      color: 'blue',
      description: 'Systems recur at higher harmonic levels
after coherence threshold transcendence',
      claims: [
        {
          id: 'ot-1',
          claim: 'Systems recur at higher harmonic levels after
coherence',
          hypothesis: 'When users complete glyph loops (80-100%
resonance), macro glyphs/archetypes begin to emerge',
          testMethod: 'Field Loop Completion Test',
          difficulty: 'Expert',
          duration: '3 months',
          participants: 1,
          status: 'running',
          confidence: 62,
          trials: 15,
          significance: null
        },
        {
          id: 'ot-2',
          claim: 'The "rules don\'t break" — they restart at a
new scale',
          hypothesis: 'During field collapse or high ambiguity,
user begins showing attractor patterns from outer ring glyphs',
          testMethod: 'Macro Pattern Tracker',
          difficulty: 'Expert',
          duration: 'Ongoing',
          participants: 1,
          status: 'available',
          confidence: null,
          trials: 0,
```

```
          significance: null
        },
        {
          id: 'ot-3',
          claim: 'Completion of a field ring triggers a new
ring',
          hypothesis: 'After 6 gates in one center are
harmonized, new glyphs or quests emerge from planetary or
archetypal layers',
          testMethod: 'Octave Shift Logger',
          difficulty: 'Expert',
          duration: '6 months',
          participants: 1,
          status: 'available',
          confidence: null,
          trials: 0,
          significance: null
        }
      ]
    }
  };

  // Test tools and their descriptions
  const testTools = {
    'Field Tone Scan': 'Real-time glyph resonance measurement
with biofeedback',
    'Coherence Challenge': '30-day behavioral alignment tracking
with HRV monitoring',
    'Heart Vector Simulation': 'HRV measurement before/after
field-aligned actions',
    'Gate Amplification Test': 'Emotional response comparison
between aligned vs random glyphs',
    'Waveform Drift Logger': 'Long-term coherence pattern
tracking and prediction',
    'Proximity Gate Tracker': 'Planetary perigee/apogee
correlation with consciousness metrics',
    'Stellar Drift Logger': 'Deep space object alignment
correlation tracking',
    'Soul Field Analyzer': 'Draconic chart resonance vs
personality chart comparison',
```

```
    'Convergence Zone Detector': 'Geographic coherence hotspot
identification',
    'Location Coherence Test': 'Travel impact measurement on
field stability',
    'Harmonic Memory Scan': 'Past-life glyph recognition
accuracy testing',
    'Spirit Vector Tracker': 'Life direction pattern and
attractor field analysis',
    'MultiNode Response Test': 'Field-specific physiological
response measurement',
    'Field Loop Completion Test': 'Macro pattern emergence after
micro completion',
    'Macro Pattern Tracker': 'Higher-order archetype detection
during field transitions',
    'Octave Shift Logger': 'Ring completion threshold and
emergence tracking'
  };

  const getStatusColor = (status) => {
    switch (status) {
      case 'completed': return 'text-green-400';
      case 'running': return 'text-yellow-400';
      case 'available': return 'text-cyan-400';
      default: return 'text-gray-400';
    }
  };

  const getStatusIcon = (status) => {
    switch (status) {
      case 'completed': return CheckCircle2;
      case 'running': return Play;
      case 'available': return Target;
      default: return XCircle;
    }
  };

  const getDifficultyColor = (difficulty) => {
    switch (difficulty) {
      case 'Beginner': return 'text-green-400';
      case 'Intermediate': return 'text-yellow-400';
      case 'Advanced': return 'text-orange-400';
```

```
      case 'Expert': return 'text-red-400';
      default: return 'text-gray-400';
    }
  };

  const getConfidenceColor = (confidence) => {
    if (!confidence) return 'text-gray-400';
    if (confidence >= 80) return 'text-green-400';
    if (confidence >= 60) return 'text-yellow-400';
    if (confidence >= 40) return 'text-orange-400';
    return 'text-red-400';
  };

  const startTest = (claimId) => {
    setRunningTests(prev => ({ ...prev, [claimId]: true }));
    // Simulate test execution
    setTimeout(() => {
      setRunningTests(prev => ({ ...prev, [claimId]: false }));
      // Update test results
      setTestResults(prev => ({
        ...prev,
        [claimId]: {
          status: 'completed',
          confidence: Math.floor(Math.random() * 40) + 60,
          trials: Math.floor(Math.random() * 50) + 10,
          significance: Math.random() * 0.05
        }
      }));
    }, 3000);
  };

  const toggleExpanded = (claimId) => {
    setExpandedClaims(prev => ({
      ...prev,
      [claimId]: !prev[claimId]
    }));
  };

  const ClaimCard = ({ claim, category }) => {
    const isExpanded = expandedClaims[claim.id];
    const isRunning = runningTests[claim.id];
```

```
    const result = testResults[claim.id];
    const StatusIcon = getStatusIcon(result?.status ||
claim.status);

    return (
      <div className="bg-gray-800/50 rounded-lg border
border-cyan-800/30 hover:border-cyan-600/50 transition-all
duration-300">
        <div className="p-4">
          <div className="flex items-start justify-between
mb-3">
            <div className="flex-1">
              <div className="flex items-center space-x-2 mb-2">
                <span className="text-cyan-100 font-semibold
text-sm">{claim.id.toUpperCase()}</span>
                <span className={`text-xs px-2 py-1 rounded-full
${getDifficultyColor(claim.difficulty)} bg-gray-900/50`}>
                  {claim.difficulty}
                </span>
                <StatusIcon className={`w-4 h-4
${getStatusColor(result?.status || claim.status)}`} />
              </div>
              <h4 className="text-cyan-100 font-medium
mb-2">{claim.claim}</h4>
              <p className="text-cyan-400 text-sm
leading-relaxed">{claim.hypothesis}</p>
            </div>
            <button
              onClick={() => toggleExpanded(claim.id)}
              className="p-1 hover:bg-gray-700 rounded
transition-colors duration-200"
            >
              {isExpanded ? <ChevronDown className="w-4 h-4
text-cyan-400" /> : <ChevronRight className="w-4 h-4
text-cyan-400" />}
            </button>
          </div>

          {isExpanded && (
            <div className="space-y-4 border-t
border-cyan-800/30 pt-4">
```

```
              {/* Test Details */}
              <div className="grid grid-cols-2 gap-4 text-sm">
                <div>
                  <span className="text-cyan-400">Test
Method:</span>
                  <p className="text-cyan-100
font-medium">{claim.testMethod}</p>
                  <p className="text-cyan-300
text-xs">{testTools[claim.testMethod]}</p>
                </div>
                <div>
                  <span
className="text-cyan-400">Duration:</span>
                  <p
className="text-cyan-100">{claim.duration}</p>
                  <span
className="text-cyan-400">Participants:</span>
                  <p
className="text-cyan-100">{claim.participants}</p>
                </div>
              </div>

              {/* Results */}
              {(result || claim.confidence) && (
                <div className="bg-gray-900/50 p-3 rounded-lg">
                  <h5 className="text-cyan-300 font-semibold
mb-2 text-sm">Test Results</h5>
                  <div className="grid grid-cols-3 gap-4
text-sm">
                    <div>
                      <span
className="text-cyan-400">Confidence:</span>
                      <p className={`font-bold
${getConfidenceColor(result?.confidence || claim.confidence)}`}>
                        {result?.confidence ||
claim.confidence}%
                      </p>
                    </div>
                    <div>
                      <span
className="text-cyan-400">Trials:</span>
```

```
                      <p className="text-cyan-100
font-mono">{result?.trials || claim.trials}</p>
                    </div>
                    <div>
                      <span
className="text-cyan-400">p-value:</span>
                      <p className="text-cyan-100 font-mono">
                        {result?.significance ||
claim.significance ?
                          (result?.significance ||
claim.significance).toFixed(3) : 'N/A'}
                      </p>
                    </div>
                  </div>
                </div>
              )}

              {/* Actions */}
              <div className="flex space-x-2">
                {(result?.status || claim.status) ===
'available' && (
                  <button
                    onClick={() => startTest(claim.id)}
                    disabled={isRunning}
                    className={`flex items-center space-x-2 px-4
py-2 rounded-lg transition-all duration-300 ${
                      isRunning
                        ? 'bg-gray-600 text-gray-400
cursor-not-allowed'
                        : 'bg-gradient-to-r from-cyan-500
to-teal-500 hover:from-cyan-400 hover:to-teal-400 text-black'
                    }`}
                  >
                    {isRunning ? <Pause className="w-4 h-4" /> :
<Play className="w-4 h-4" />}
                    <span className="font-medium">{isRunning ?
'Running...' : 'Start Test'}</span>
                  </button>
                )}
```

```
                {(result?.status || claim.status) === 'running' && (
                  <button className="flex items-center space-x-2 px-4 py-2 bg-yellow-500 text-black rounded-lg hover:bg-yellow-400 transition-colors duration-300">
                    <Activity className="w-4 h-4" />
                    <span className="font-medium">View Progress</span>
                  </button>
                )}

                {(result?.status || claim.status) === 'completed' && (
                  <button className="flex items-center space-x-2 px-4 py-2 bg-green-500 text-black rounded-lg hover:bg-green-400 transition-colors duration-300">
                    <BarChart3 className="w-4 h-4" />
                    <span className="font-medium">View Results</span>
                  </button>
                )}

                <button className="flex items-center space-x-2 px-4 py-2 bg-gray-700 text-cyan-300 rounded-lg hover:bg-gray-600 transition-colors duration-300">
                  <FileText className="w-4 h-4" />
                  <span className="font-medium">Details</span>
                </button>
              </div>
            </div>
          )}
        </div>
      </div>
    );
  };

  const CategoryStats = ({ category }) => {
    const claims = theories[category].claims;
    const completed = claims.filter(c => (testResults[c.id]?.status || c.status) === 'completed').length;
```

```
    const running = claims.filter(c =>
(testResults[c.id]?.status || c.status) === 'running').length;
    const avgConfidence = claims
      .filter(c => testResults[c.id]?.confidence ||
c.confidence)
      .reduce((sum, c) => sum + (testResults[c.id]?.confidence
|| c.confidence || 0), 0) /
      claims.filter(c => testResults[c.id]?.confidence ||
c.confidence).length;

    return (
      <div className="grid grid-cols-3 gap-4 mb-6">
        <div className="bg-gray-800/50 p-4 rounded-lg border
border-cyan-800/30">
          <div className="flex items-center space-x-2 mb-2">
            <CheckCircle2 className="w-5 h-5 text-green-400" />
            <span className="text-cyan-400
text-sm">Completed</span>
          </div>
          <p className="text-2xl font-bold
text-green-400">{completed}</p>
          <p className="text-cyan-300 text-sm">of
{claims.length} claims</p>
        </div>

        <div className="bg-gray-800/50 p-4 rounded-lg border
border-cyan-800/30">
          <div className="flex items-center space-x-2 mb-2">
            <Play className="w-5 h-5 text-yellow-400" />
            <span className="text-cyan-400
text-sm">Running</span>
          </div>
          <p className="text-2xl font-bold
text-yellow-400">{running}</p>
          <p className="text-cyan-300 text-sm">active tests</p>
        </div>

        <div className="bg-gray-800/50 p-4 rounded-lg border
border-cyan-800/30">
          <div className="flex items-center space-x-2 mb-2">
            <TrendingUp className="w-5 h-5 text-cyan-400" />
```

```
            <span className="text-cyan-400 text-sm">Avg
Confidence</span>
          </div>
          <p className={`text-2xl font-bold
${getConfidenceColor(avgConfidence)}`}>
            {avgConfidence ? Math.round(avgConfidence) : '--'}%
          </p>
          <p className="text-cyan-300 text-sm">across all
tests</p>
        </div>
      </div>
    );
  };

  return (
    <div className="min-h-screen bg-black text-cyan-100 p-4
lg:p-6">
      <div className="max-w-7xl mx-auto">
        {/* Header */}
        <div className="text-center mb-8">
          <h1 className="text-3xl lg:text-4xl font-bold
bg-gradient-to-r from-cyan-400 to-teal-400 bg-clip-text
text-transparent mb-4">
            YOU-NIVERSE Theory Testing Dashboard
          </h1>
          <p className="text-cyan-300 text-lg">
            Scientific validation of consciousness field
theories through falsifiable experiments
          </p>
        </div>

        {/* Category Navigation */}
        <div className="grid grid-cols-2 lg:grid-cols-4 gap-4
mb-8">
          {Object.entries(theories).map(([key, theory]) => {
            const Icon = theory.icon;
            return (
              <button
                key={key}
                onClick={() => setActiveCategory(key)}
```

```
                  className={`p-4 rounded-xl border transition-all
duration-300 ${
                    activeCategory === key
                      ? 'border-cyan-400 bg-cyan-900/20'
                      : 'border-cyan-800/30
hover:border-cyan-600/50 bg-gray-900/50'
                  }`}
                >
                  <Icon className="w-8 h-8 text-cyan-400 mx-auto
mb-3" />
                  <h3 className="text-cyan-100 font-semibold
text-sm lg:text-base">{theory.name}</h3>
                  <p className="text-cyan-400 text-xs mt-2
leading-relaxed">{theory.description}</p>
                </button>
              );
            })}
          </div>

          {/* Active Theory Content */}
          <div className="space-y-6">
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <div className="flex items-center space-x-3 mb-4">

{React.createElement(theories[activeCategory].icon, { className:
"w-6 h-6 text-cyan-400" })}
                <h2 className="text-2xl font-bold
text-cyan-100">{theories[activeCategory].name}</h2>
              </div>
              <p className="text-cyan-300
leading-relaxed">{theories[activeCategory].description}</p>
            </div>

            {/* Category Statistics */}
            <CategoryStats category={activeCategory} />

            {/* Claims List */}
            <div className="space-y-4">
              <h3 className="text-xl font-semibold text-cyan-300
mb-4">Testable Claims & Hypotheses</h3>
```

```
            {theories[activeCategory].claims.map((claim) => (
              <ClaimCard key={claim.id} claim={claim}
category={activeCategory} />
            ))}
          </div>

          {/* Export & Analysis Tools */}
          <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
            <h3 className="text-lg font-semibold text-cyan-300
mb-4">Analysis & Export Tools</h3>
            <div className="grid grid-cols-2 lg:grid-cols-4
gap-4">
              <button className="flex items-center space-x-2 p-3
bg-gradient-to-r from-cyan-500 to-teal-500 text-black
font-semibold rounded-lg hover:from-cyan-400 hover:to-teal-400
transition-all duration-300">
                <Download className="w-4 h-4" />
                <span>Export Results</span>
              </button>
              <button className="flex items-center space-x-2 p-3
bg-gradient-to-r from-teal-500 to-emerald-500 text-black
font-semibold rounded-lg hover:from-teal-400
hover:to-emerald-400 transition-all duration-300">
                <FileText className="w-4 h-4" />
                <span>Scientific PDF</span>
              </button>
              <button className="flex items-center space-x-2 p-3
bg-gradient-to-r from-emerald-500 to-blue-500 text-black
font-semibold rounded-lg hover:from-emerald-400
hover:to-blue-400 transition-all duration-300">
                <Share className="w-4 h-4" />
                <span>Share Results</span>
              </button>
              <button className="flex items-center space-x-2 p-3
bg-gradient-to-r from-blue-500 to-purple-500 text-black
font-semibold rounded-lg hover:from-blue-400 hover:to-purple-400
transition-all duration-300">
                <BarChart3 className="w-4 h-4" />
                <span>Meta Analysis</span>
              </button>
```

```
            </div>
          </div>

          {/* Falsifiability Notice */}
          <div className="bg-amber-900/20 border
border-amber-700/50 p-4 rounded-xl">
            <div className="flex items-start space-x-3">
              <Info className="w-5 h-5 text-amber-400 mt-0.5" />
              <div>
                <h4 className="text-amber-400 font-semibold
mb-2">Scientific Falsifiability</h4>
                <p className="text-amber-200 text-sm
leading-relaxed">
                  All claims are designed to be falsifiable with
measurable variables, control conditions, and statistical
significance testing.
                  Each experiment can prove the theory wrong
through random responses, lack of correlation, or statistical
insignificance.
                </p>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default TheoryTestingDashboard;
import React, { useState, useEffect, useRef } from 'react';
import {
  Brain, Heart, Zap, Eye, Star, Activity, Globe,
  Play, Pause, RotateCw, Settings, CircuitBoard,
  Radio, Wifi, TrendingUp, AlertTriangle, CheckCircle2
} from 'lucide-react';

const MultiNodeConsciousnessSimulator = () => {
  const [isRunning, setIsRunning] = useState(false);
  const [systemCoherence, setSystemCoherence] = useState(78);
  const [communications, setCommunications] = useState([]);
```

```
  // Individual node states
  const [nodes, setNodes] = useState({
    mind: {
      active: true,
      coherence: 75,
      state: 'processing',
      energy: 0.8,
      thoughts: ['Initializing cognitive patterns...'],
      currentThought: 'Analyzing field coherence patterns...'
    },
    heart: {
      active: true,
      coherence: 82,
      state: 'resonant',
      energy: 0.9,
      emotion: 'calm',
      resonanceHistory: []
    },
    body: {
      active: true,
      coherence: 68,
      state: 'stable',
      energy: 0.7,
      vitals: { energy: 75, presence: 80, tension: 25 },
      somaticState: 'balanced'
    },
    field: {
      active: true,
      coherence: 85,
      state: 'connected',
      energy: 0.85,
      signals: [],
      environment: {
        sunPosition: 45,
        moonPhase: 0.7,
        weather: 'clear'
      }
    },
    glyph: {
      active: true,
```

```
      coherence: 91,
      state: 'encoding',
      energy: 0.95,
      activeGlyph: { gate: 47, line: 3, resonance: 'processing'
},
      history: []
    },
    observer: {
      active: true,
      coherence: 77,
      state: 'watching',
      energy: 0.85,
      observations: [],
      systemInsights: 'Initializing meta-analysis...'
    }
  });setCommunications(prev => [newComm, ...prev.slice(0, 49)]);
  };

  // Update individual node
  const updateNode = (nodeId, updates) => {
    setNodes(prev => ({
      ...prev,
      [nodeId]: {
        ...prev[nodeId],
        ...updates,
        lastUpdate: Date.now()
      }
    }));
  };

  // Calculate system coherence
  useEffect(() => {
    const avgCoherence = Object.values(nodes).reduce((sum, node)
=> sum + node.coherence, 0) / 6;
    setSystemCoherence(Math.round(avgCoherence));
  }, [nodes]);

  // Simulation loops for each node
  useEffect(() => {
    if (!isRunning) return;
```

```
    // Mind Node Logic
    const mindInterval = setInterval(() => {
      const thoughtPatterns = [
        'Analyzing field coherence patterns...',
        'Processing environmental inputs...',
        'Evaluating decision pathways...',
        'Integrating multi-node feedback...',
        'Questioning current assumptions...',
        'Synthesizing consciousness data...'
      ];

      const newThought =
thoughtPatterns[Math.floor(Math.random() *
thoughtPatterns.length)];
      const coherenceShift = (Math.random() - 0.5) * 10;
      const newCoherence = Math.max(50, Math.min(100,
nodes.mind.coherence + coherenceShift));

      updateNode('mind', {
        currentThought: newThought,
        thoughts: [newThought, ...nodes.mind.thoughts.slice(0,
3)],
        coherence: Math.round(newCoherence),
        state: newCoherence > 80 ? 'clarity' : newCoherence > 60
? 'processing' : 'confused'
      });

      if (Math.random() > 0.7) {
        addCommunication('mind', 'heart', 'Cognitive insight
emerging', 'insight');
      }
    }, 3000);

    // Heart Node Logic
    const heartInterval = setInterval(() => {
      const emotions = ['joy', 'love', 'calm', 'excitement',
'contemplative', 'expansive'];
      const newEmotion = emotions[Math.floor(Math.random() *
emotions.length)];

      const coherenceShift = (Math.random() - 0.5) * 15;
```

```
      const newCoherence = Math.max(40, Math.min(100,
nodes.heart.coherence + coherenceShift));

      updateNode('heart', {
        emotion: newEmotion,
        coherence: Math.round(newCoherence),
        state: newCoherence > 85 ? 'euphoric' : newCoherence >
70 ? 'resonant' : 'stable',
        resonanceHistory: [
          { emotion: newEmotion, coherence:
Math.round(newCoherence), timestamp: Date.now() },
          ...nodes.heart.resonanceHistory.slice(0, 2)
        ]
      });

      if (Math.random() > 0.6) {
        addCommunication('heart', 'body', `Feeling
${newEmotion}`, 'emotion');
      }
    }, 2500);

    // Body Node Logic
    const bodyInterval = setInterval(() => {
      const states = ['grounded', 'energized', 'flowing',
'tense', 'relaxed', 'vibrant'];
      const newState = states[Math.floor(Math.random() *
states.length)];

      const newVitals = {
        energy: Math.max(20, Math.min(100,
nodes.body.vitals.energy + (Math.random() - 0.5) * 20)),
        presence: Math.max(30, Math.min(100,
nodes.body.vitals.presence + (Math.random() - 0.5) * 15)),
        tension: Math.max(0, Math.min(80,
nodes.body.vitals.tension + (Math.random() - 0.5) * 15))
      };

      const avgVital = (newVitals.energy + newVitals.presence +
(100 - newVitals.tension)) / 3;

      updateNode('body', {
```

```
        somaticState: newState,
        vitals: newVitals,
        coherence: Math.round(avgVital),
        state: avgVital > 80 ? 'vitalized' : avgVital > 65 ?
'stable' : 'adjusting'
      });

      if (Math.random() > 0.8) {
        addCommunication('body', 'field', `Somatic state:
${newState}`, 'sensation');
      }
    }, 4000);

    // Field Engine Logic
    const fieldInterval = setInterval(() => {
      const signalTypes = [
        'Solar flare detected',
        'Planetary alignment shift',
        'Magnetic field fluctuation',
        'Cosmic ray intensity change',
        'Schumann resonance spike',
        'Local ley line activation'
      ];

      const newSignal = {
        type: signalTypes[Math.floor(Math.random() *
signalTypes.length)],
        intensity: Math.random() * 100,
        timestamp: Date.now()
      };

      const newEnvironment = {
        ...nodes.field.environment,
        sunPosition: (nodes.field.environment.sunPosition + 1) %
360,
        moonPhase: (nodes.field.environment.moonPhase + 0.01) %
1
      };

      const fieldCoherence = Math.max(60, Math.min(95, 70 +
(newSignal.intensity - 50) * 0.5));
```

```
      updateNode('field', {
        signals: [newSignal, ...nodes.field.signals.slice(0,
3)],
        environment: newEnvironment,
        coherence: Math.round(fieldCoherence),
        state: fieldCoherence > 85 ? 'harmonious' :
fieldCoherence > 70 ? 'connected' : 'turbulent'
      });

      if (Math.random() > 0.7) {
        addCommunication('field', 'all', newSignal.type,
'environmental');
      }
    }, 5000);

    // Glyph Engine Logic
    const glyphInterval = setInterval(() => {
      const newGate = Math.floor(Math.random() * 64) + 1;
      const newLine = Math.floor(Math.random() * 6) + 1;
      const avgCoherence = (nodes.mind.coherence +
nodes.heart.coherence) / 2;

      const resonance = avgCoherence > 85 ? 'siddhi' :
                        avgCoherence > 70 ? 'gift' :
                        avgCoherence > 50 ? 'shadow' :
'distortion';

      const newGlyph = { gate: newGate, line: newLine, resonance
};
      const glyphCoherence = Math.max(50, Math.min(100,
avgCoherence + (Math.random() - 0.5) * 20));

      updateNode('glyph', {
        activeGlyph: newGlyph,
        history: [
          { ...newGlyph, timestamp: Date.now() },
          ...nodes.glyph.history.slice(0, 2)
        ],
        coherence: Math.round(glyphCoherence),
        state: resonance === 'siddhi' ? 'transcendent' :
```

```
                resonance === 'gift' ? 'encoding' :
                resonance === 'shadow' ? 'processing' :
'distorting'
      });

      if (Math.random() > 0.6) {
        addCommunication('glyph', 'mind', `Gate
${newGate}.${newLine} activated`, 'glyph');
      }
    }, 6000);

    // Observer Hub Logic
    const observerInterval = setInterval(() => {
      const observationTypes = [
        'Cross-node coherence pattern detected',
        'Environmental correlation identified',
        'Glyph-emotion synchronicity observed',
        'Field-body resonance alignment',
        'Mind-heart integration spike',
        'System-wide stability increase'
      ];

      const newObservation = {
        text: observationTypes[Math.floor(Math.random() *
observationTypes.length)],
        confidence: Math.round(60 + Math.random() * 40),
        timestamp: Date.now()
      };

      const insights = [
        `System coherence: ${systemCoherence}%`,
        `Stability index: ${Math.round(100 -
Math.abs(systemCoherence - 75))}%`,
        `Communication flow: ${communications.length > 40 ?
'High' : 'Moderate'}`,
        'Multi-node integration active'
      ];

      updateNode('observer', {
        observations: [newObservation,
...nodes.observer.observations.slice(0, 2)],
```

```
        systemInsights: insights[Math.floor(Math.random() *
insights.length)],
        coherence: systemCoherence,
        state: systemCoherence > 80 ? 'integrating' :
systemCoherence > 65 ? 'watching' : 'analyzing'
      });

      if (Math.random() > 0.8) {
        addCommunication('observer', 'all', `Meta-insight:
${nodes.observer.systemInsights}`, 'analysis');
      }
    }, 7000);

    return () => {
      clearInterval(mindInterval);
      clearInterval(heartInterval);
      clearInterval(bodyInterval);
      clearInterval(fieldInterval);
      clearInterval(glyphInterval);
      clearInterval(observerInterval);
    };
  }, [isRunning, nodes, systemCoherence,
communications.length]);

  const toggleSimulation = () => {
    setIsRunning(!isRunning);
  };

  const resetSystem = () => {
    setIsRunning(false);
    setCommunications([]);
    setNodes({
      mind: {
        active: true,
        coherence: 75,
        state: 'processing',
        energy: 0.8,
        thoughts: ['Initializing cognitive patterns...'],
        currentThought: 'Analyzing field coherence patterns...'
      },
      heart: {
```

```
        active: true,
        coherence: 82,
        state: 'resonant',
        energy: 0.9,
        emotion: 'calm',
        resonanceHistory: []
      },
      body: {
        active: true,
        coherence: 68,
        state: 'stable',
        energy: 0.7,
        vitals: { energy: 75, presence: 80, tension: 25 },
        somaticState: 'balanced'
      },
      field: {
        active: true,
        coherence: 85,
        state: 'connected',
        energy: 0.85,
        signals: [],
        environment: {
          sunPosition: 45,
          moonPhase: 0.7,
          weather: 'clear'
        }
      },
      glyph: {
        active: true,
        coherence: 91,
        state: 'encoding',
        energy: 0.95,
        activeGlyph: { gate: 47, line: 3, resonance:
'processing' },
        history: []
      },
      observer: {
        active: true,
        coherence: 77,
        state: 'watching',
        energy: 0.85,
```

```
        observations: [],
        systemInsights: 'Initializing meta-analysis...'
      }
    });
  };

  const getNodeColor = (nodeId) => {
    const colors = {
      mind: 'text-cyan-400',
      heart: 'text-emerald-400',
      body: 'text-teal-400',
      field: 'text-purple-400',
      glyph: 'text-yellow-400',
      observer: 'text-indigo-400',
      all: 'text-white'
    };
    return colors[nodeId] || 'text-gray-400';
  };

  // Node component
  const NodeDisplay = ({ nodeId, nodeData, icon: Icon,
borderColor }) => (
    <div className={`bg-gray-900/50 p-4 rounded-xl border
${borderColor}`}>
      <div className="flex items-center justify-between mb-4">
        <div className="flex items-center space-x-2">
          <Icon className="w-5 h-5 text-cyan-400" />
          <h3 className="text-cyan-100 font-semibold
capitalize">{nodeId} Node</h3>
        </div>
        <div className="flex items-center space-x-2">
          <div className={`w-3 h-3 rounded-full
${nodeData.active ? 'bg-green-400 animate-pulse' :
'bg-gray-500'}`}></div>
          <span className="text-cyan-300
text-sm">{nodeData.state}</span>
        </div>
      </div>

      <div className="space-y-3">
        <div className="flex justify-between items-center">
```

```
          <span className="text-cyan-400
text-sm">Coherence:</span>
          <span className="text-cyan-100
font-mono">{nodeData.coherence}%</span>
        </div>

        <div className="w-full bg-gray-700 rounded-full h-2">
          <div
            className="bg-gradient-to-r from-cyan-500
to-teal-500 h-2 rounded-full transition-all duration-1000"
            style={{ width: `${nodeData.coherence}%` }}
          ></div>
        </div>

        {/* Node-specific content */}
        {nodeId === 'mind' && (
          <div className="bg-gray-800/50 p-3 rounded-lg">
            <p className="text-cyan-300 text-sm font-semibold
mb-1">Current Process:</p>
            <p className="text-cyan-100
text-xs">{nodeData.currentThought}</p>
          </div>
        )}

        {nodeId === 'heart' && (
          <div className="bg-gray-800/50 p-3 rounded-lg">
            <p className="text-emerald-300 text-sm font-semibold
mb-1">Emotion:</p>
            <p className="text-emerald-100 text-lg
capitalize">{nodeData.emotion}</p>
          </div>
        )}

        {nodeId === 'body' && (
          <div className="grid grid-cols-3 gap-2 text-xs">
            <div className="bg-gray-800/30 p-2 rounded
text-center">
              <div className="text-teal-200">Energy</div>
              <div className="text-teal-100
font-mono">{Math.round(nodeData.vitals.energy)}%</div>
            </div>
```

```
            <div className="bg-gray-800/30 p-2 rounded
text-center">
              <div className="text-teal-200">Presence</div>
              <div className="text-teal-100
font-mono">{Math.round(nodeData.vitals.presence)}%</div>
            </div>
            <div className="bg-gray-800/30 p-2 rounded
text-center">
              <div className="text-teal-200">Tension</div>
              <div className="text-teal-100
font-mono">{Math.round(nodeData.vitals.tension)}%</div>
            </div>
          </div>
        )}

        {nodeId === 'field' && nodeData.signals.length > 0 && (
          <div className="space-y-1">
            <p className="text-purple-400 text-xs">Recent
Signals:</p>
            {nodeData.signals.slice(0, 2).map((signal, index) =>
(
              <div key={signal.timestamp}
className="text-purple-200 text-xs bg-gray-800/30 p-2 rounded">
                {signal.type}
              </div>
            ))}
          </div>
        )}

        {nodeId === 'glyph' && (
          <div className="bg-gray-800/50 p-3 rounded-lg">
            <p className="text-yellow-300 text-sm font-semibold
mb-2">Active Glyph:</p>
            <div className="text-center">
              <div className="text-yellow-100 text-2xl
font-mono">

{nodeData.activeGlyph.gate}.{nodeData.activeGlyph.line}
              </div>
              <div className="text-yellow-300 text-sm
capitalize">
```

```
                {nodeData.activeGlyph.resonance}
              </div>
            </div>
          </div>
        )}

        {nodeId === 'observer' && (
          <div className="bg-gray-800/50 p-3 rounded-lg">
            <p className="text-indigo-300 text-sm font-semibold
mb-2">System Insight:</p>
            <p className="text-indigo-100
text-xs">{nodeData.systemInsights}</p>
          </div>
        )}
      </div>
    </div>
  );

  return (
    <div className="min-h-screen bg-black text-cyan-100 p-4
lg:p-6">
      <div className="max-w-7xl mx-auto">
        {/* Header */}
        <div className="text-center mb-8">
          <h1 className="text-3xl lg:text-4xl font-bold
bg-gradient-to-r from-cyan-400 to-purple-400 bg-clip-text
text-transparent mb-4">
            Multi-Node Consciousness Simulator
          </h1>
          <p className="text-cyan-300 text-lg">
            Real-time distributed consciousness field processing
          </p>
        </div>

        {/* System Controls */}
        <div className="grid grid-cols-1 lg:grid-cols-3 gap-6
mb-8">
          <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
            <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
```

```
              <Settings className="w-5 h-5 mr-2" />
              System Controls
            </h3>

            <div className="space-y-4">
              <div className="flex items-center
justify-between">
                <span className="text-cyan-300">Status:</span>
                <div className="flex items-center space-x-2">
                  <div className={`w-3 h-3 rounded-full
${isRunning ? 'bg-green-400 animate-pulse' :
'bg-red-400'}`}></div>
                  <span className="text-white
font-mono">{isRunning ? 'RUNNING' : 'STOPPED'}</span>
                </div>
              </div>

              <div className="flex items-center
justify-between">
                <span className="text-cyan-300">System
Coherence:</span>
                <span className="text-white font-mono
text-lg">{systemCoherence}%</span>
              </div>

              <div className="w-full bg-gray-700 rounded-full
h-3">
                <div
                  className="bg-gradient-to-r from-cyan-500
to-purple-500 h-3 rounded-full transition-all duration-1000"
                  style={{ width: `${systemCoherence}%` }}
                ></div>
              </div>

              <div className="flex space-x-2">
                <button
                  onClick={toggleSimulation}
                  className={`flex items-center space-x-2 px-4
py-2 rounded-lg transition-all duration-300 ${
                    isRunning
                      ? 'bg-red-500 hover:bg-red-600 text-white'
```

```
                      : 'bg-gradient-to-r from-cyan-500
to-teal-500 hover:from-cyan-400 hover:to-teal-400 text-black'
                    }`}
                  >
                    {isRunning ? <Pause className="w-4 h-4" /> :
<Play className="w-4 h-4" />}
                    <span className="font-medium">{isRunning ?
'Pause' : 'Start'}</span>
                  </button>

                  <button
                    onClick={resetSystem}
                    className="flex items-center space-x-2 px-4
py-2 bg-gray-700 text-cyan-300 rounded-lg hover:bg-gray-600
transition-colors duration-300"
                  >
                    <RotateCw className="w-4 h-4" />
                    <span className="font-medium">Reset</span>
                  </button>
                </div>
              </div>
            </div>

            {/* Communication Log */}
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                <Radio className="w-5 h-5 mr-2" />
                Inter-Node Communication
              </h3>
              <div className="space-y-2 max-h-60 overflow-y-auto">
                {communications.slice(0, 6).map((comm) => (
                  <div key={comm.id} className="bg-gray-800/50 p-2
rounded text-xs">
                    <div className="flex justify-between
items-center mb-1">
                      <div className="flex items-center
space-x-2">
                        <span
className={getNodeColor(comm.from)}>{comm.from}</span>
```

```
                      <span className="text-gray-400">→</span>
                      <span
className={getNodeColor(comm.to)}>{comm.to}</span>
                    </div>
                    <span className="text-gray-500 text-xs">
                      {comm.timestamp.toLocaleTimeString()}
                    </span>
                  </div>
                  <p
className="text-gray-300">{comm.message}</p>
                </div>
              ))}
            </div>
          </div>

          {/* Node Status Overview */}
          <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
            <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
              <TrendingUp className="w-5 h-5 mr-2" />
              Node Status
            </h3>
            <div className="space-y-2">
              {Object.entries(nodes).map(([nodeId, nodeData]) =>
(
                <div key={nodeId} className="flex items-center
justify-between">
                  <span className="text-cyan-300
capitalize">{nodeId}:</span>
                  <div className="flex items-center space-x-2">
                    <span className="text-white font-mono
text-sm">{nodeData.coherence}%</span>
                    <div className={`w-2 h-2 rounded-full
${nodeData.active ? 'bg-green-400' : 'bg-red-400'}`}></div>
                  </div>
                </div>
              ))}
            </div>
          </div>
        </div>
```

```
        {/* Node Grid */}
        <div className="grid grid-cols-1 md:grid-cols-2
lg:grid-cols-3 gap-6">
          <NodeDisplay
            nodeId="mind"
            nodeData={nodes.mind}
            icon={Brain}
            borderColor="border-cyan-800/30"
          />
          <NodeDisplay
            nodeId="heart"
            nodeData={nodes.heart}
            icon={Heart}
            borderColor="border-emerald-800/30"
          />
          <NodeDisplay
            nodeId="body"
            nodeData={nodes.body}
            icon={Zap}
            borderColor="border-teal-800/30"
          />
          <NodeDisplay
            nodeId="field"
            nodeData={nodes.field}
            icon={Globe}
            borderColor="border-purple-800/30"
          />
          <NodeDisplay
            nodeId="glyph"
            nodeData={nodes.glyph}
            icon={Star}
            borderColor="border-yellow-800/30"
          />
          <NodeDisplay
            nodeId="observer"
            nodeData={nodes.observer}
            icon={Eye}
            borderColor="border-indigo-800/30"
          />
        </div>
```

```
        {/* System Info */}
        <div className="mt-8 bg-gray-900/50 p-6 rounded-xl
border border-cyan-800/30">
          <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
            <CircuitBoard className="w-5 h-5 mr-2" />
            System Architecture
          </h3>
          <div className="grid grid-cols-1 md:grid-cols-3 gap-4
text-sm">
            <div>
              <h4 className="text-cyan-400 font-semibold
mb-2">Distributed Processing</h4>
              <p className="text-cyan-300">Each node operates
independently with autonomous decision-making and state
management.</p>
            </div>
            <div>
              <h4 className="text-emerald-400 font-semibold
mb-2">Inter-Node Communication</h4>
              <p className="text-cyan-300">Nodes exchange
signals, emotions, insights through reactive message
passing.</p>
            </div>
            <div>
              <h4 className="text-purple-400 font-semibold
mb-2">Emergent Behavior</h4>
              <p className="text-cyan-300">System consciousness
emerges from complex node interactions and feedback loops.</p>
            </div>
          </div>
        </div>
      </div>
    );
};

export default MultiNodeConsciousnessSimulator;

import React, { useState, useEffect, useRef } from 'react';
```

```
import {
  Eye, Sparkles, Brain, Heart, Zap, Star, Globe, Target,
  Send, Mic, RotateCw, BookOpen, TrendingUp, Clock, MapPin,
  Activity, Compass, MessageCircle, Lightbulb, Sun, Moon
} from 'lucide-react';

const OracleInterface = () => {
  const [question, setQuestion] = useState('');
  const [consultationHistory, setConsultationHistory] =
useState([]);
  const [isConsulting, setIsConsulting] = useState(false);
  const [oracleMode, setOracleMode] = useState('trinity');
  const [currentReading, setCurrentReading] = useState(null);

  // Mock consciousness state
  const [consciousness, setConsciousness] = useState({
    mind: { coherence: 78, state: 'analyzing', energy: 0.8 },
    heart: { coherence: 85, state: 'open', energy: 0.9 },
    body: { coherence: 72, state: 'grounded', energy: 0.7 },
    soul: { coherence: 91, state: 'seeking', energy: 0.95 },
    spirit: { coherence: 83, state: 'flowing', energy: 0.85 }
  });

  // Mock environmental data
  const environmentalData = {
    location: { name: 'Salinas, CA', lat: 36.6777, lng:
-121.6555 },
    time: new Date(),
    sunPosition: 67,
    moonPhase: 0.73,
    planetaryWeather: 'Mercury conjunct Jupiter - Mental
expansion favored',
    fieldStrength: 87,
    astrologicalMoment: 'Waxing Gibbous in Sagittarius'
  };

  const messagesEndRef = useRef(null);

  // Oracle response generation
  const generateOracleResponse = async (userQuestion) => {
    setIsConsulting(true);
```

```
    // Simulate consultation time
    await new Promise(resolve => setTimeout(resolve, 2000 +
Math.random() * 3000));

    const responses = {
      trinity: [
        {
          primary: "The Trinity Fields speak in harmony. Your
Soul field resonates at 91% - a powerful seeking energy that
calls for deep exploration.",
          interpretation: "This high soul coherence suggests
you're in a phase of spiritual expansion. The question you've
posed emerges from this seeking nature.",
          guidance: "Trust the pull toward deeper understanding.
Your current field configuration supports breakthrough
insights.",
          gates: { primary: 61, secondary: 24, tertiary: 2 },
          fieldFocus: 'soul'
        },
        {
          primary: "Your Heart field pulses at 85% coherence
while your Mind analyzes at 78% - a beautiful balance of feeling
and thinking approaches your question.",
          interpretation: "The interplay between heart wisdom
and mental clarity creates optimal conditions for integrated
guidance.",
          guidance: "Allow both emotional intuition and logical
analysis to inform your path forward. Integration is your
strength.",
          gates: { primary: 26, secondary: 47, tertiary: 64 },
          fieldFocus: 'heart'
        }
      ],
      nodes: [
        {
          primary: "The Node Network reveals: Mind processes
actively while Heart resonates deeply. Your consciousness system
seeks integration.",
```

```
          interpretation: "Multi-node analysis shows heightened
communication between cognitive and emotional processing
centers.",
          guidance: "This is an optimal time for decisions that
require both analytical depth and emotional intelligence.",
          systemState: 'Integrated Processing Mode',
          nodeActivity: { mind: 'high', heart: 'very-high',
body: 'moderate' }
        }
      ],
      field: [
        {
          primary: "Field resonance indicates: Your current
location amplifies Soul field coherence by 23%. The geography
supports your inquiry.",
          interpretation: "Salinas, CA creates a beneficial
field interaction with your birth signature, particularly
enhancing intuitive reception.",
          guidance: "This is an auspicious time and place for
receiving guidance. The field alignment supports clear
reception.",
          fieldStrength: 87,
          locationBonus: '+23% Soul coherence',
          travelGuidance: 'Current location optimal - no travel
needed'
        }
      ],
      cosmic: [
        {
          primary: "Mercury conjunct Jupiter illuminates: Your
question emerges from an expansion of consciousness seeking
practical expression.",
          interpretation: "The cosmic weather supports mental
breakthrough and philosophical insight, but seeks grounded
application.",
          guidance: "Think bigger, then break it down into
actionable steps. The universe supports both vision and
manifestation.",
          planetaryAspect: 'Mercury conjunct Jupiter',
          cosmicMessage: 'Expansion through Integration',
          timing: 'Optimal for 5 days'
```

```
        }
      ]
    };

    const modeResponses = responses[oracleMode];
    const selectedResponse =
modeResponses[Math.floor(Math.random() * modeResponses.length)];

    const reading = {
      id: Date.now(),
      question: userQuestion,
      mode: oracleMode,
      timestamp: new Date(),
      consciousness: { ...consciousness },
      environment: { ...environmentalData },
      ...selectedResponse,
      confidence: 85 + Math.floor(Math.random() * 15),
      resonanceScore: 78 + Math.floor(Math.random() * 22)
    };

    setCurrentReading(reading);
    setConsultationHistory(prev => [reading, ...prev]);
    setIsConsulting(false);

    return reading;
  };

  const handleSubmitQuestion = async () => {
    if (!question.trim()) return;
    await generateOracleResponse(question);
    setQuestion('');
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault();
      handleSubmitQuestion();
    }
  };

  // Auto-scroll to bottom
```

```
  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth'
});
  }, [consultationHistory]);

  // Simulate consciousness field updates
  useEffect(() => {
    const interval = setInterval(() => {
      setConsciousness(prev => {
        const updated = { ...prev };
        Object.keys(updated).forEach(field => {
          const variance = (Math.random() - 0.5) * 8;
          updated[field] = {
            ...updated[field],
            coherence: Math.max(50, Math.min(100,
updated[field].coherence + variance)),
            energy: Math.max(0.3, Math.min(1.0,
updated[field].energy + (Math.random() - 0.5) * 0.2))
          };
        });
        return updated;
      });
    }, 5000);

    return () => clearInterval(interval);
  }, []);

  const getModeDescription = (mode) => {
    const descriptions = {
      trinity: 'Consult the five Trinity Fields for integrated
consciousness guidance',
      nodes: 'Access the multi-node consciousness simulator for
distributed awareness insights',
      field: 'Query field resonance patterns and geographic
consciousness effects',
      cosmic: 'Receive cosmic weather updates and planetary
consciousness influences'
    };
    return descriptions[mode];
  };
```

```
  const getModeIcon = (mode) => {
    const icons = {
      trinity: Target,
      nodes: Activity,
      field: Globe,
      cosmic: Sparkles
    };
    return icons[mode];
  };const QuickQuestionButton = ({ text, category }) => (
    <button
      onClick={() => setQuestion(text)}
      className="text-left p-3 bg-gray-800/50 border
border-cyan-800/30 rounded-lg hover:border-cyan-600/50
transition-all duration-300 text-sm"
    >
      <div className="text-cyan-400 text-xs font-semibold
mb-1">{category}</div>
      <div className="text-cyan-100">{text}</div>
    </button>
  );

  const ReadingDisplay = ({ reading }) => {
    const ModeIcon = getModeIcon(reading.mode);

    return (
      <div className="bg-gradient-to-br from-gray-900/80
to-gray-800/80 p-6 rounded-xl border border-cyan-400/30
shadow-lg mb-6">
        <div className="flex items-center justify-between mb-4">
          <div className="flex items-center space-x-3">
            <ModeIcon className="w-6 h-6 text-cyan-400" />
            <span className="text-cyan-300 font-semibold
capitalize">{reading.mode} Oracle</span>
          </div>
          <div className="text-cyan-400 text-sm">
            {reading.timestamp.toLocaleTimeString()}
          </div>
        </div>

        <div className="mb-4 p-4 bg-black/30 rounded-lg
border-l-4 border-cyan-400">
```

```
          <div className="text-cyan-300 text-sm font-semibold
mb-1">Your Question:</div>
          <div className="text-cyan-100
italic">"{reading.question}"</div>
        </div>

        <div className="space-y-4">
          <div>
            <div className="text-cyan-400 font-semibold mb-2
flex items-center">
              <Eye className="w-4 h-4 mr-2" />
              Oracle Vision
            </div>
            <p className="text-cyan-100 leading-relaxed
text-lg">{reading.primary}</p>
          </div>

          <div>
            <div className="text-cyan-400 font-semibold mb-2
flex items-center">
              <Lightbulb className="w-4 h-4 mr-2" />
              Interpretation
            </div>
            <p className="text-cyan-200
leading-relaxed">{reading.interpretation}</p>
          </div>

          <div>
            <div className="text-cyan-400 font-semibold mb-2
flex items-center">
              <Compass className="w-4 h-4 mr-2" />
              Guidance
            </div>
            <p className="text-cyan-100 leading-relaxed
font-medium">{reading.guidance}</p>
          </div>

          {/* Mode-specific additional info */}
          {reading.gates && (
            <div className="bg-gray-800/50 p-4 rounded-lg">
```

```
              <div className="text-cyan-400 font-semibold
mb-2">Active Gates</div>
              <div className="flex space-x-4 text-sm">
                <div>Primary: <span className="font-mono
text-cyan-100">{reading.gates.primary}</span></div>
                <div>Secondary: <span className="font-mono
text-cyan-100">{reading.gates.secondary}</span></div>
                <div>Tertiary: <span className="font-mono
text-cyan-100">{reading.gates.tertiary}</span></div>
              </div>
              <div className="text-cyan-300 text-sm mt-2">Focus
Field: <span className="capitalize
font-semibold">{reading.fieldFocus}</span></div>
            </div>
          )}

          {reading.systemState && (
            <div className="bg-gray-800/50 p-4 rounded-lg">
              <div className="text-cyan-400 font-semibold
mb-2">System State</div>
              <div className="text-cyan-100
font-medium">{reading.systemState}</div>
              {reading.nodeActivity && (
                <div className="mt-2 text-sm">

{Object.entries(reading.nodeActivity).map(([node, activity]) =>
(
                    <div key={node} className="flex
justify-between">
                      <span className="text-cyan-300
capitalize">{node}:</span>
                      <span className="text-cyan-100
capitalize">{activity}</span>
                    </div>
                  ))}
                </div>
              )}
            </div>
          )}

          {reading.fieldStrength && (
```

```
            <div className="bg-gray-800/50 p-4 rounded-lg">
              <div className="text-cyan-400 font-semibold
mb-2">Field Analysis</div>
              <div className="grid grid-cols-2 gap-4 text-sm">
                <div>
                  <span className="text-cyan-300">Field
Strength:</span>
                  <span className="text-cyan-100 font-mono
ml-2">{reading.fieldStrength}%</span>
                </div>
                <div>
                  <span className="text-cyan-300">Location
Bonus:</span>
                  <span className="text-green-400 font-mono
ml-2">{reading.locationBonus}</span>
                </div>
              </div>
              <div className="text-cyan-200 text-sm
mt-2">{reading.travelGuidance}</div>
            </div>
          )}

          {reading.planetaryAspect && (
            <div className="bg-gray-800/50 p-4 rounded-lg">
              <div className="text-cyan-400 font-semibold
mb-2">Cosmic Weather</div>
              <div className="space-y-2 text-sm">
                <div><span
className="text-cyan-300">Aspect:</span> <span
className="text-cyan-100">{reading.planetaryAspect}</span></div>
                <div><span
className="text-cyan-300">Message:</span> <span
className="text-cyan-100">{reading.cosmicMessage}</span></div>
                <div><span
className="text-cyan-300">Timing:</span> <span
className="text-cyan-100">{reading.timing}</span></div>
              </div>
            </div>
          )}

          {/* Confidence and Resonance */}
```

```
          <div className="flex justify-between items-center pt-4
border-t border-cyan-800/30">
            <div className="flex items-center space-x-4
text-sm">
              <div>
                <span
className="text-cyan-400">Confidence:</span>
                <span className="text-cyan-100 font-mono
ml-2">{reading.confidence}%</span>
              </div>
              <div>
                <span
className="text-cyan-400">Resonance:</span>
                <span className="text-cyan-100 font-mono
ml-2">{reading.resonanceScore}%</span>
              </div>
            </div>
            <div className="flex space-x-2">
              <button className="p-2 bg-cyan-500/20
text-cyan-400 rounded-lg hover:bg-cyan-500/30 transition-colors
duration-300">
                <BookOpen className="w-4 h-4" />
              </button>
              <button className="p-2 bg-cyan-500/20
text-cyan-400 rounded-lg hover:bg-cyan-500/30 transition-colors
duration-300">
                <TrendingUp className="w-4 h-4" />
              </button>
            </div>
          </div>
        </div>
      </div>
    );
  };

  return (
    <div className="min-h-screen bg-black text-cyan-100 p-4
lg:p-6">
      <div className="max-w-7xl mx-auto">
        {/* Header */}
        <div className="text-center mb-8">
```

```
          <h1 className="text-3xl lg:text-4xl font-bold
bg-gradient-to-r from-cyan-400 to-purple-400 bg-clip-text
text-transparent mb-4">
            Oracle Interface
          </h1>
          <p className="text-cyan-300 text-lg">
            Consciousness-informed guidance through integrated
field awareness
          </p>
        </div>

        <div className="grid grid-cols-1 xl:grid-cols-4 gap-6">
          {/* Main Oracle Interface */}
          <div className="xl:col-span-3 space-y-6">
            {/* Oracle Mode Selection */}
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4">Oracle Consultation Mode</h3>
              <div className="grid grid-cols-2 lg:grid-cols-4
gap-3">
                {['trinity', 'nodes', 'field',
'cosmic'].map((mode) => {
                  const ModeIcon = getModeIcon(mode);
                  return (
                    <button
                      key={mode}
                      onClick={() => setOracleMode(mode)}
                      className={`p-4 rounded-lg border
transition-all duration-300 ${
                        oracleMode === mode
                          ? 'border-cyan-400 bg-cyan-900/20'
                          : 'border-cyan-800/30
hover:border-cyan-600/50 bg-gray-800/50'
                      }`}
                    >
                      <ModeIcon className="w-6 h-6 text-cyan-400
mx-auto mb-2" />
                      <div className="text-cyan-100 font-medium
capitalize text-sm">{mode}</div>
                    </button>
```

```
                  );
                })}
              </div>
              <p className="text-cyan-400 text-sm
mt-4">{getModeDescription(oracleMode)}</p>
            </div>

            {/* Question Input */}
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                <MessageCircle className="w-5 h-5 mr-2" />
                Ask Your Question
              </h3>

              <div className="space-y-4">
                <div className="relative">
                  <textarea
                    value={question}
                    onChange={(e) =>
setQuestion(e.target.value)}
                    onKeyPress={handleKeyPress}
                    placeholder="What guidance do you seek from
the consciousness fields?"
                    className="w-full p-4 bg-gray-800 border
border-cyan-700 rounded-lg text-cyan-100 placeholder-cyan-400
resize-none h-24 focus:border-teal-400 focus:outline-none"
                    disabled={isConsulting}
                  />
                  <div className="absolute bottom-3 right-3">
                    <button className="p-2 bg-gray-700
text-cyan-300 rounded-lg hover:bg-gray-600 transition-colors
duration-200">
                      <Mic className="w-4 h-4" />
                    </button>
                  </div>
                </div>

                <div className="flex justify-between
items-center">
```

```
                  <div className="text-cyan-400 text-sm">
                    Mode: <span className="capitalize
font-semibold">{oracleMode}</span>
                  </div>
                  <button
                    onClick={handleSubmitQuestion}
                    disabled={!question.trim() || isConsulting}
                    className={`flex items-center space-x-2 px-6
py-3 rounded-lg transition-all duration-300 ${
                      isConsulting
                        ? 'bg-gray-600 text-gray-400
cursor-not-allowed'
                        : 'bg-gradient-to-r from-cyan-500
to-purple-500 hover:from-cyan-400 hover:to-purple-400
text-white'
                    }`}
                  >
                    {isConsulting ? (
                      <>
                        <div className="w-4 h-4 border-2
border-gray-400 border-t-transparent rounded-full
animate-spin"></div>
                        <span>Consulting Oracle...</span>
                      </>
                    ) : (
                      <>
                        <Send className="w-4 h-4" />
                        <span>Consult Oracle</span>
                      </>
                    )}
                  </button>
                </div>
              </div>
            </div>

            {/* Quick Questions */}
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4">Quick Consultations</h3>
```

```
              <div className="grid grid-cols-1 md:grid-cols-2
gap-3">
                <QuickQuestionButton
                  text="What does my current field configuration
reveal about my life path?"
                  category="Life Direction"
                />
                <QuickQuestionButton
                  text="How can I improve coherence between my
Trinity Fields?"
                  category="Field Alignment"
                />
                <QuickQuestionButton
                  text="What consciousness patterns am I ready
to transcend?"
                  category="Growth"
                />
                <QuickQuestionButton
                  text="How do current planetary influences
affect my decisions?"
                  category="Cosmic Timing"
                />
                <QuickQuestionButton
                  text="What is my soul seeking to express
through current experiences?"
                  category="Soul Purpose"
                />
                <QuickQuestionButton
                  text="How can I better integrate my multi-node
consciousness?"
                  category="Integration"
                />
              </div>
            </div>

            {/* Current Reading */}
            {currentReading && (
              <div>
                <h3 className="text-xl font-semibold
text-cyan-300 mb-4">Current Reading</h3>
                <ReadingDisplay reading={currentReading} />
```

```
              </div>
            )}

            {/* Consultation History */}
            {consultationHistory.length > 1 && (
              <div>
                <h3 className="text-xl font-semibold
text-cyan-300 mb-4">Previous Consultations</h3>
                <div className="space-y-4">
                  {consultationHistory.slice(1, 4).map((reading)
=> (
                    <ReadingDisplay key={reading.id}
reading={reading} />
                  ))}
                </div>
              </div>
            )}
          </div>

          {/* Sidebar */}
          <div className="space-y-6">
            {/* Current Consciousness State */}
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                <Activity className="w-5 h-5 mr-2" />
                Current Field State
              </h3>
              <div className="space-y-3">
                {Object.entries(consciousness).map(([field,
data]) => {
                  const fieldIcons = { mind: Brain, heart:
Heart, body: Zap, soul: Eye, spirit: Star };
                  const Icon = fieldIcons[field];
                  return (
                    <div key={field} className="bg-gray-800/50
p-3 rounded-lg">
                      <div className="flex items-center
justify-between mb-2">
```

```
                        <div className="flex items-center
space-x-2">
                          <Icon className="w-4 h-4
text-cyan-400" />
                          <span className="text-cyan-100
capitalize font-medium">{field}</span>
                        </div>
                        <span className="text-cyan-300 font-mono
text-sm">{data.coherence}%</span>
                      </div>
                      <div className="w-full bg-gray-700
rounded-full h-2">
                        <div
                          className="bg-gradient-to-r
from-cyan-500 to-purple-500 h-2 rounded-full transition-all
duration-1000"
                          style={{ width: `${data.coherence}%`
}}
                        ></div>
                      </div>
                      <div className="text-cyan-400 text-xs mt-1
capitalize">{data.state}</div>
                    </div>
                  );
                })}
              </div>
            </div>{/* Environmental Context */}
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                <Globe className="w-5 h-5 mr-2" />
                Environmental Context
              </h3>
              <div className="space-y-3 text-sm">
                <div className="flex items-center
justify-between">
                  <div className="flex items-center space-x-2">
                    <MapPin className="w-4 h-4 text-cyan-400" />
                    <span
className="text-cyan-300">Location</span>
```

```
                  </div>
                  <span
className="text-cyan-100">{environmentalData.location.name}</spa
n>
                </div>

                <div className="flex items-center
justify-between">
                  <div className="flex items-center space-x-2">
                    <Sun className="w-4 h-4 text-yellow-400" />
                    <span className="text-cyan-300">Solar</span>
                  </div>
                  <span
className="text-cyan-100">{environmentalData.sunPosition}°</span
>
                </div>

                <div className="flex items-center
justify-between">
                  <div className="flex items-center space-x-2">
                    <Moon className="w-4 h-4 text-blue-400" />
                    <span className="text-cyan-300">Lunar</span>
                  </div>
                  <span
className="text-cyan-100">{Math.round(environmentalData.moonPhas
e * 100)}%</span>
                </div>

                <div className="bg-gray-800/50 p-3 rounded-lg
mt-3">
                  <div className="text-cyan-400 text-xs
font-semibold mb-1">Astrological Moment</div>
                  <div className="text-cyan-100
text-xs">{environmentalData.astrologicalMoment}</div>
                </div>

                <div className="bg-gray-800/50 p-3 rounded-lg">
                  <div className="text-cyan-400 text-xs
font-semibold mb-1">Planetary Weather</div>
                  <div className="text-cyan-100
text-xs">{environmentalData.planetaryWeather}</div>
```

```
                </div>
              </div>
            </div>

            {/* Oracle Statistics */}
            <div className="bg-gray-900/50 p-6 rounded-xl border
border-cyan-800/30">
              <h3 className="text-lg font-semibold text-cyan-300
mb-4 flex items-center">
                <TrendingUp className="w-5 h-5 mr-2" />
                Oracle Statistics
              </h3>
              <div className="space-y-3">
                <div className="flex justify-between">
                  <span className="text-cyan-300">Total
Consultations:</span>
                  <span className="text-cyan-100
font-mono">{consultationHistory.length}</span>
                </div>
                <div className="flex justify-between">
                  <span className="text-cyan-300">Avg
Confidence:</span>
                  <span className="text-cyan-100 font-mono">
                    {consultationHistory.length > 0
                      ?
Math.round(consultationHistory.reduce((sum, r) => sum +
r.confidence, 0) / consultationHistory.length)
                      : '--'}%
                  </span>
                </div>
                <div className="flex justify-between">
                  <span className="text-cyan-300">Preferred
Mode:</span>
                  <span className="text-cyan-100
capitalize">{oracleMode}</span>
                </div>
                <div className="flex justify-between">
                  <span className="text-cyan-300">Field
Coherence:</span>
                  <span className="text-cyan-100 font-mono">
```

```
{Math.round(Object.values(consciousness).reduce((sum, f) => sum
+ f.coherence, 0) / 5)}%
                  </span>
                </div>
              </div>
            </div>
          </div>
        </div>
        <div ref={messagesEndRef} />
      </div>
    </div>
  );
};

export default OracleInterface;
// YOU-NIVERSE Firebase Backend Architecture
// Complete database schema and real-time sync system

// ========================================
// FIREBASE CONFIGURATION
// ========================================

import { initializeApp } from 'firebase/app';
import { getFirestore, connectFirestoreEmulator } from
'firebase/firestore';
import { getAuth } from 'firebase/auth';
import { getStorage } from 'firebase/storage';

const firebaseConfig = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId:
process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_APP_ID
};

const app = initializeApp(firebaseConfig);
export const db = getFirestore(app);
```

```
export const auth = getAuth(app);
export const storage = getStorage(app);

// ========================================
// DATABASE SCHEMA STRUCTURE
// ========================================

/*
FIRESTORE COLLECTIONS STRUCTURE:

/users/{userId}
├── profile (document)
├── trinityChart (document)
├── consciousness (collection)
│   └── {sessionId} (documents)
├── experiments (collection)
│   └── {experimentId} (documents)
├── memoryVault (collection)
│   └── {entryId} (documents)
├── glyphs (collection)
│   └── {glyphId} (documents)
├── fieldFriend (document)
├── oracleConsultations (collection)
│   └── {consultationId} (documents)
├── dreams (collection)
│   └── {dreamId} (documents)
├── biofeedback (collection)
│   └── {sessionId} (documents)
└── gameProgress (document)

/global
├── fieldResonanceMap (collection)
│   └── {locationId} (documents)
├── collectiveConsciousness (document)
└── astrologicalWeather (document)
*/

// ========================================
// DATA MODELS & INTERFACES
// ========================================
```

```
// User Profile Model
export const UserProfileModel = {
  userId: '',
  email: '',
  displayName: '',
  birthData: {
    date: '', // ISO string
    time: '', // HH:MM format
    location: {
      name: '',
      lat: 0,
      lng: 0,
      timezone: ''
    }
  },
  preferences: {
    defaultOracleMode: 'trinity',
    biofeedbackDevices: [],
    privacySettings: {
      shareFieldData: false,
      shareLocationData: false
    }
  },
  createdAt: null, // Firebase Timestamp
  lastActive: null,
  subscription: {
    tier: 'free', // free, premium, researcher
    features: []
  }
};

// Trinity Chart Model
export const TrinityChartModel = {
  userId: '',
  birthChart: {
    tropical: {},
    sidereal: {},
    draconic: {}
  },
  trinityFields: {
    mind: {
```

```
      gate: 0,
      line: 0,
      color: 0,
      tone: 0,
      base: 0,
      planet: '',
      sign: '',
      degree: '',
      house: 0,
      activation: '', // Design/Personality
      theme: ''
    },
    body: { /* same structure */ },
    heart: { /* same structure */ },
    soul: { /* same structure */ },
    spirit: { /* same structure */ }
  },
  standingWavePattern: {
    baseFrequency: 0,
    harmonics: [],
    interferenceNodes: []
  },
  generatedAt: null,
  version: '1.0'
};

// Consciousness Session Model
export const ConsciousnessSessionModel = {
  sessionId: '',
  userId: '',
  startTime: null,
  endTime: null,
  nodeStates: {
    mind: {
      coherence: 0,
      state: '',
      energy: 0,
      thoughts: [],
      currentThought: ''
    },
    heart: {
```

```
      coherence: 0,
      state: '',
      energy: 0,
      emotion: '',
      resonanceHistory: []
    },
    body: {
      coherence: 0,
      state: '',
      energy: 0,
      vitals: {
        energy: 0,
        presence: 0,
        tension: 0
      },
      somaticState: ''
    },
    field: {
      coherence: 0,
      state: '',
      energy: 0,
      signals: [],
      environment: {
        sunPosition: 0,
        moonPhase: 0,
        weather: ''
      }
    },
    glyph: {
      coherence: 0,
      state: '',
      energy: 0,
      activeGlyph: {
        gate: 0,
        line: 0,
        resonance: ''
      },
      history: []
    },
    observer: {
      coherence: 0,
```

```
      state: '',
      energy: 0,
      observations: [],
      systemInsights: ''
    }
  },
  systemCoherence: 0,
  communications: [],
  location: {
    lat: 0,
    lng: 0,
    name: ''
  },
  tags: []
};

// Memory Vault Entry Model
export const MemoryVaultEntryModel = {
  entryId: '',
  userId: '',
  timestamp: null,
  type: '', // 'experience', 'dream', 'insight', 'experiment',
'synchronicity'
  title: '',
  content: '',
  fieldData: {
    mind: 0,
    body: 0,
    heart: 0,
    soul: 0,
    spirit: 0
  },
  glyphsActive: [],
  location: {
    lat: 0,
    lng: 0,
    name: ''
  },
  astrologicalContext: {
    sunPosition: 0,
    moonPhase: 0,
```

```
    planetaryAspects: []
  },
  biofeedbackData: {
    hrv: 0,
    stress: 0,
    coherence: 0
  },
  tags: [],
  mood: '',
  significance: 0, // 1-10 scale
  connections: [], // Links to other entries
  aiInsights: ''
};

// Glyph Model
export const GlyphModel = {
  glyphId: '',
  userId: '',
  gate: 0,
  line: 0,
  field: '', // mind, body, heart, soul, spirit
  state: '', // distortion, shadow, gift, siddhi
  resonanceScore: 0,
  activationHistory: [],
  createdAt: null,
  lastActivated: null,
  mutations: [], // History of state changes
  userNotes: '',
  aiInterpretation: '',
  connectedExperiments: [],
  visualData: {
    color: '',
    shape: '',
    animation: ''
  }
};

// Field Friend Model
export const FieldFriendModel = {
  userId: '',
  name: '',
```

```
  species: '', // crystalline, fluid, gaseous, etc.
  currentForm: '',
  stats: {
    energy: 0,
    happiness: 0,
    evolution: 0,
    trust: 0,
    wisdom: 0
  },
  mood: '',
  activeGlyphs: [],
  evolutionHistory: [],
  interactions: [],
  preferences: {
    favoriteFields: [],
    optimalCoherence: 0
  },
  appearance: {
    color: '',
    size: '',
    texture: '',
    aura: ''
  },
  lastUpdated: null,
  version: '1.0'
};

// Oracle Consultation Model
export const OracleConsultationModel = {
  consultationId: '',
  userId: '',
  timestamp: null,
  question: '',
  mode: '', // trinity, nodes, field, cosmic
  response: {
    primary: '',
    interpretation: '',
    guidance: '',
    confidence: 0,
    resonanceScore: 0
  },
```

```
  contextData: {
    trinityFields: {},
    consciousness: {},
    environment: {},
    astrologicalMoment: ''
  },
  additionalData: {
    gates: {},
    systemState: '',
    fieldStrength: 0,
    planetaryAspect: ''
  },
  userFeedback: {
    accuracy: 0,
    helpfulness: 0,
    notes: ''
  },
  followUpQuestions: []
};

// Experiment Model
export const ExperimentModel = {
  experimentId: '',
  userId: '',
  theoryCategory: '', // waveform-will, stellar-proximology,
soul-spirit, octave-threshold
  claimId: '',
  hypothesis: '',
  startTime: null,
  endTime: null,
  status: '', // design, running, completed, paused
  methodology: '',
  variables: {
    independent: [],
    dependent: [],
    controlled: []
  },
  dataCollection: {
    biofeedback: [],
    glyphResponses: [],
    fieldMeasurements: [],
```

```
    userReports: []
  },
  results: {
    confidence: 0,
    significance: 0,
    correlation: 0,
    conclusion: '',
    evidence: []
  },
  metadata: {
    location: {},
    astrologicalContext: {},
    participantCount: 1
  }
};

// ========================================
// REAL-TIME SYNC FUNCTIONS
// ========================================

import {
  doc, collection, addDoc, updateDoc, onSnapshot,
  query, where, orderBy, limit, serverTimestamp,
  arrayUnion, increment
} from 'firebase/firestore';

// Consciousness Session Sync
export class ConsciousnessSync {
  constructor(userId) {
    this.userId = userId;
    this.sessionRef = null;
    this.unsubscribe = null;
  }

  async startSession() {
    const sessionData = {
      ...ConsciousnessSessionModel,
      userId: this.userId,
      startTime: serverTimestamp(),
      nodeStates: this.initializeNodeStates()
    };
```

```
    this.sessionRef = await addDoc(
      collection(db, 'users', this.userId, 'consciousness'),
      sessionData
    );

    return this.sessionRef.id;
  }

  updateNodeState(nodeId, updates) {
    if (!this.sessionRef) return;

    updateDoc(this.sessionRef, {
      [`nodeStates.${nodeId}`]: {
        ...updates,
        lastUpdate: serverTimestamp()
      },
      systemCoherence: this.calculateSystemCoherence()
    });
  }

  addCommunication(from, to, message, type = 'signal') {
    if (!this.sessionRef) return;

    updateDoc(this.sessionRef, {
      communications: arrayUnion({
        id: Date.now(),
        from,
        to,
        message,
        type,
        timestamp: serverTimestamp()
      })
    });
  }

  subscribeToSession(callback) {
    if (!this.sessionRef) return;

    this.unsubscribe = onSnapshot(this.sessionRef, callback);
  }
```

```
  endSession() {
    if (!this.sessionRef) return;

    updateDoc(this.sessionRef, {
      endTime: serverTimestamp()
    });

    if (this.unsubscribe) {
      this.unsubscribe();
    }
  }

  initializeNodeStates() {
    return {
      mind: { coherence: 75, state: 'processing', energy: 0.8,
thoughts: [], currentThought: '' },
      heart: { coherence: 82, state: 'resonant', energy: 0.9,
emotion: 'calm', resonanceHistory: [] },
      body: { coherence: 68, state: 'stable', energy: 0.7,
vitals: { energy: 75, presence: 80, tension: 25 }, somaticState:
'balanced' },
      field: { coherence: 85, state: 'connected', energy: 0.85,
signals: [], environment: { sunPosition: 45, moonPhase: 0.7,
weather: 'clear' } },
      glyph: { coherence: 91, state: 'encoding', energy: 0.95,
activeGlyph: { gate: 47, line: 3, resonance: 'processing' },
history: [] },
      observer: { coherence: 77, state: 'watching', energy:
0.85, observations: [], systemInsights: '' }
    };
  }

  calculateSystemCoherence() {
    // Implementation would calculate average coherence across
all nodes
    return 78; // Placeholder
  }
}

// Memory Vault Sync
```

```
export class MemoryVaultSync {
  constructor(userId) {
    this.userId = userId;
  }

  async addEntry(entryData) {
    const entry = {
      ...MemoryVaultEntryModel,
      ...entryData,
      userId: this.userId,
      timestamp: serverTimestamp()
    };

    const docRef = await addDoc(
      collection(db, 'users', this.userId, 'memoryVault'),
      entry
    );

    // Update Field Friend based on entry
    this.updateFieldFriendFromEntry(entryData);

    return docRef.id;
  }

  async getEntries(filters = {}) {
    let q = query(
      collection(db, 'users', this.userId, 'memoryVault'),
      orderBy('timestamp', 'desc'),
      limit(50)
    );

    // Add filters
    if (filters.type) {
      q = query(q, where('type', '==', filters.type));
    }

    if (filters.field) {
      q = query(q, where(`fieldData.${filters.field}`, '>',
70));
    }
```

```
    return q;
  }

  subscribeToEntries(callback, filters = {}) {
    const q = this.getEntries(filters);
    return onSnapshot(q, callback);
  }

  async updateFieldFriendFromEntry(entryData) {
    const fieldFriendRef = doc(db, 'users', this.userId,
'fieldFriend', 'current');

    // Calculate field friend stat changes based on entry
    const statChanges =
this.calculateFieldFriendImpact(entryData);

    await updateDoc(fieldFriendRef, {
      'stats.energy': increment(statChanges.energy),
      'stats.happiness': increment(statChanges.happiness),
      'stats.evolution': increment(statChanges.evolution),
      lastUpdated: serverTimestamp()
    });
  }

  calculateFieldFriendImpact(entryData) {
    // Logic to determine how memory vault entries affect field
friend
    const avgFieldCoherence =
Object.values(entryData.fieldData).reduce((a, b) => a + b, 0) /
5;

    return {
      energy: avgFieldCoherence > 80 ? 2 : avgFieldCoherence >
60 ? 1 : -1,
      happiness: entryData.mood === 'positive' ? 3 :
entryData.mood === 'neutral' ? 0 : -2,
      evolution: entryData.significance > 7 ? 5 :
entryData.significance > 4 ? 2 : 0
    };
  }
}
```

```
// Glyph Sync
export class GlyphSync {
  constructor(userId) {
    this.userId = userId;
  }

  async activateGlyph(gate, line, field) {
    const glyphData = {
      ...GlyphModel,
      userId: this.userId,
      gate,
      line,
      field,
      state: 'shadow', // Starting state
      resonanceScore: 50,
      createdAt: serverTimestamp(),
      lastActivated: serverTimestamp()
    };

    const docRef = await addDoc(
      collection(db, 'users', this.userId, 'glyphs'),
      glyphData
    );

    // Trigger field friend update
    this.updateFieldFriendFromGlyph(glyphData);

    return docRef.id;
  }

  async mutateGlyph(glyphId, newState, resonanceScore) {
    const glyphRef = doc(db, 'users', this.userId, 'glyphs',
glyphId);

    await updateDoc(glyphRef, {
      state: newState,
      resonanceScore,
      lastActivated: serverTimestamp(),
      mutations: arrayUnion({
        fromState: 'previous', // Would get current state first
```

```
        toState: newState,
        timestamp: serverTimestamp(),
        trigger: 'user_interaction' // or 'experiment',
'biofeedback', etc.
      })
    });

    // Check for field friend evolution triggers
    this.checkEvolutionTriggers(glyphId, newState);
  }

  async updateFieldFriendFromGlyph(glyphData) {
    const fieldFriendRef = doc(db, 'users', this.userId,
'fieldFriend', 'current');

    await updateDoc(fieldFriendRef, {
      activeGlyphs: arrayUnion({
        gate: glyphData.gate,
        line: glyphData.line,
        field: glyphData.field,
        state: glyphData.state
      }),
      lastUpdated: serverTimestamp()
    });
  }

  async checkEvolutionTriggers(glyphId, newState) {
    if (newState === 'siddhi') {
      // Trigger major field friend evolution
      const fieldFriendRef = doc(db, 'users', this.userId,
'fieldFriend', 'current');

      await updateDoc(fieldFriendRef, {
        'stats.evolution': increment(10),
        'stats.wisdom': increment(5),
        evolutionHistory: arrayUnion({
          trigger: 'glyph_transcendence',
          glyphId,
          timestamp: serverTimestamp()
        })
      });
```

```
    }
  }

  subscribeToGlyphs(callback) {
    const q = query(
      collection(db, 'users', this.userId, 'glyphs'),
      orderBy('lastActivated', 'desc')
    );

    return onSnapshot(q, callback);
  }
}

// Oracle Sync
export class OracleSync {
  constructor(userId) {
    this.userId = userId;
  }

  async saveConsultation(consultationData) {
    const consultation = {
      ...OracleConsultationModel,
      ...consultationData,
      userId: this.userId,
      timestamp: serverTimestamp()
    };

    const docRef = await addDoc(
      collection(db, 'users', this.userId,
'oracleConsultations'),
      consultation
    );

    // Update user's oracle usage stats
    this.updateOracleStats(consultationData.mode);

    return docRef.id;
  }

  async updateOracleStats(mode) {
    const userRef = doc(db, 'users', this.userId);
```

```
    await updateDoc(userRef, {
      [`stats.oracle.${mode}Usage`]: increment(1),
      'stats.oracle.totalConsultations': increment(1),
      'preferences.defaultOracleMode': mode
    });
  }

  subscribeToConsultations(callback, limit = 10) {
    const q = query(
      collection(db, 'users', this.userId,
'oracleConsultations'),
      orderBy('timestamp', 'desc'),
      limit(limit)
    );

    return onSnapshot(q, callback);
  }
}

// ========================================
// GLOBAL STATE MANAGEMENT
// ========================================

export class YouUniverseState {
  constructor(userId) {
    this.userId = userId;
    this.consciousnessSync = new ConsciousnessSync(userId);
    this.memoryVaultSync = new MemoryVaultSync(userId);
    this.glyphSync = new GlyphSync(userId);
    this.oracleSync = new OracleSync(userId);

    this.subscriptions = new Map();
    this.state = {
      user: null,
      trinityChart: null,
      currentSession: null,
      fieldFriend: null,
      activeGlyphs: [],
      recentMemories: [],
      systemCoherence: 0
```

```
    };
  }

  async initialize() {
    // Load user profile and trinity chart
    await this.loadUserData();

    // Subscribe to real-time updates
    this.subscribeToUpdates();

    // Start consciousness session
    await this.consciousnessSync.startSession();
  }

  async loadUserData() {
    // Implementation to load user profile and trinity chart
    const userRef = doc(db, 'users', this.userId);
    const chartRef = doc(db, 'users', this.userId,
'trinityChart', 'current');
    const fieldFriendRef = doc(db, 'users', this.userId,
'fieldFriend', 'current');

    // Load and set state
  }

  subscribeToUpdates() {
    // Set up real-time listeners for all major data streams

    // Consciousness session updates
    this.consciousnessSync.subscribeToSession((doc) => {
      this.state.currentSession = doc.data();
      this.notifyStateChange('session', doc.data());
    });

    // Recent memory vault entries
    const memoryUnsubscribe =
this.memoryVaultSync.subscribeToEntries((snapshot) => {
      this.state.recentMemories = snapshot.docs.map(doc => ({
id: doc.id, ...doc.data() }));
      this.notifyStateChange('memories',
this.state.recentMemories);
```

```
    });

    this.subscriptions.set('memories', memoryUnsubscribe);

    // Active glyphs
    const glyphUnsubscribe =
this.glyphSync.subscribeToGlyphs((snapshot) => {
      this.state.activeGlyphs = snapshot.docs.map(doc => ({ id:
doc.id, ...doc.data() }));
      this.notifyStateChange('glyphs', this.state.activeGlyphs);
    });

    this.subscriptions.set('glyphs', glyphUnsubscribe);
  }

  notifyStateChange(type, data) {
    // Emit events for UI components to react to
    window.dispatchEvent(new
CustomEvent('youniverse-state-change', {
      detail: { type, data }
    }));
  }

  cleanup() {
    // Clean up all subscriptions
    this.subscriptions.forEach(unsubscribe => unsubscribe());
    this.consciousnessSync.endSession();
  }
}

// ========================================
// USAGE EXAMPLE
// ========================================

/*
// Initialize YOU-NIVERSE state management
const youniverse = new YouUniverseState(userId);
await youniverse.initialize();

// Update consciousness node
youniverse.consciousnessSync.updateNodeState('mind', {
```

```
  coherence: 85,
  state: 'clarity',
  currentThought: 'Deep insight emerging...'
});

// Add memory vault entry
await youniverse.memoryVaultSync.addEntry({
  type: 'insight',
  title: 'Breakthrough realization',
  content: 'Understanding the connection between heart and mind
fields...',
  fieldData: { mind: 85, heart: 90, body: 70, soul: 88, spirit:
82 },
  mood: 'enlightened',
  significance: 9
});

// Activate and mutate glyph
const glyphId = await youniverse.glyphSync.activateGlyph(47, 3,
'mind');
await youniverse.glyphSync.mutateGlyph(glyphId, 'gift', 85);

// Save oracle consultation
await youniverse.oracleSync.saveConsultation({
  question: 'What is my next step in consciousness
development?',
  mode: 'trinity',
  response: {
    primary: 'Your mind field shows exceptional clarity...',
    interpretation: 'This suggests...',
    guidance: 'Focus on...',
    confidence: 92,
    resonanceScore: 88
  }
});
*/

export default YouUniverseState;

# YOUNIVERSE - Modular Consciousness Platform
```

Project Structure

```
YOUNIVERSE/
├── src/
│   ├── core/                    # Core system files
│   │   ├── charts/              # Birth chart calculation engine
│   │   │   ├── zero_wrong_chart.py
│   │   │   ├── chart_calculator.js
│   │   │   └── ephemeris_data/
│   │   ├── fields/              # Field-specific logic
│   │   │   ├── zer_field.js
│   │   │   ├── mind_field.js
│   │   │   ├── body_field.js
│   │   │   ├── heart_field.js
│   │   │   ├── soul_field.js
│   │   │   └── spirit_field.js
│   │   └── consciousness.js     # Observer/witness layer
│   │
│   ├── modules/                 # Field-specific modules
│   │   ├── zer/                 # Origin/potential tracking
│   │   │   ├── ZerDashboard.jsx
│   │   │   ├── PotentialTracker.jsx
│   │   │   └── OriginProtocols.jsx
│   │   │
│   │   ├── mind/                # Perception & cognition
│   │   │   ├── MindDashboard.jsx
│   │   │   ├── PerceptionTracker.jsx
│   │   │   ├── BeliefMapper.jsx
│   │   │   └── CognitiveBias.jsx
│   │   │
│   │   ├── body/                # Physical embodiment & gravity
│   │   │   ├── BodyDashboard.jsx
│   │   │   ├── GravityAnchor.jsx
│   │   │   ├── SomaticTracker.jsx
│   │   │   └── EmbodimentMetrics.jsx
│   │   │
│   │   ├── heart/               # Emotional regulation & plasma
│   │   │   ├── HeartDashboard.jsx
│   │   │   ├── EmotionalWeather.jsx
│   │   │   ├── IntimacyMapping.jsx
```

```
│   │   │   └── PlasmaFlow.jsx
│   │   │
│   │   ├── soul/              # Karmic patterns & narrative
│   │   │   ├── SoulDashboard.jsx
│   │   │   ├── KarmicMapper.jsx
│   │   │   ├── NarrativeWeaver.jsx
│   │   │   └── LifeThemes.jsx
│   │   │
│   │   ├── spirit/            # Galactic signature & trajectory
│   │   │   ├── SpiritDashboard.jsx
│   │   │   ├── GalacticSignature.jsx
│   │   │   ├── UniquenessFactor.jsx
│   │   │   └── CosmicTrajectory.jsx
│   │   │
│   │   └── consciousness/     # Witness layer integration
│   │       ├── WitnessDashboard.jsx
│   │       ├── FieldHarmonizer.jsx
│   │       └── ConsciousnessMetrics.jsx
│   │
│   ├── components/            # Shared UI components
│   │   ├── charts/
│   │   │   ├── BirthChart.jsx
│   │   │   ├── TransitChart.jsx
│   │   │   └── FieldOverlay.jsx
│   │   ├── navigation/
│   │   │   ├── FieldNavigator.jsx
│   │   │   ├── ModuleRouter.jsx
│   │   │   └── QuickAccess.jsx
│   │   ├── data-viz/
│   │   │   ├── WaveformDisplay.jsx
│   │   │   ├── ResonanceGraph.jsx
│   │   │   └── FieldCoherence.jsx
│   │   └── glyphs/
│   │       ├── GateGlyph.jsx
│   │       ├── PlanetGlyph.jsx
│   │       └── FieldGlyph.jsx
│   │
│   ├── tools/                 # Utility tools and calculators
│   │   ├── field_researcher/  # Real-time experimentation
│   │   │   ├── ExperimentDesigner.jsx
│   │   │   ├── DataCollector.jsx
```

```
│   │   │   └── ResultAnalyzer.jsx
│   │   ├── app_builder/       # Self-building functionality
│   │   │   ├── ComponentGenerator.jsx
│   │   │   ├── ModuleCreator.jsx
│   │   │   └── ArchitectureDesigner.jsx
│   │   ├── astro_calc/        # Astrological calculations
│   │   │   ├── EphemerisEngine.jsx
│   │   │   ├── AspectCalculator.jsx
│   │   │   └── TransitPredictor.jsx
│   │   └── resonance_tools/   # Field interaction tools
│   │       ├── FieldTuner.jsx
│   │       ├── ResonanceScanner.jsx
│   │       └── CoherenceOptimizer.jsx
│   │
│   ├── data/                  # Static data and configurations
│   │   ├── gates/
│   │   │   ├── gate_codex.json
│   │   │   ├── gene_keys.json
│   │   │   └── i_ching.json
│   │   ├── ephemeris/
│   │   │   ├── planetary_data.json
│   │   │   └── asteroid_data.json
│   │   ├── fields/
│   │   │   ├── field_mappings.json
│   │   │   ├── resonance_tables.json
│   │   │   └── coherence_metrics.json
│   │   └── protocols/
│   │       ├── rituals.json
│   │       ├── practices.json
│   │       └── experiments.json
│   │
│   ├── styles/                # Dark theme styling
│   │   ├── themes/
│   │   │   ├── dark_cosmic.css
│   │   │   ├── field_colors.css
│   │   │   └── glyph_fonts.css
│   │   └── components/
│   │       ├── dashboard.css
│   │       ├── charts.css
│   │       └── navigation.css
│   │
```

```
│   └── App.jsx                # Main application entry
│
├── api/                       # Backend services
│   ├── chart_service.py       # Python chart calculation API
│   ├── ephemeris_service.py   # Real-time planetary positions
│   ├── field_tracker.py      # Field state monitoring
│   └── database/
│       ├── user_charts.db
│       ├── experiment_data.db
│       └── field_history.db
│
├── tests/                     # Testing suite
│   ├── unit/
│   ├── integration/
│   └── field_experiments/
│
└── docs/                      # Documentation
    ├── field_theory.md
    ├── api_reference.md
    └── user_manual.md
```

Field-Specific Module Features

Zer Module (Origin/Potential)
- **Origin Tracker**: Monitor connection to source consciousness
- **Potential Meter**: Track unexpressed creative energy
- **Void Protocols**: Practices for returning to zero-point

Mind Module (Perception/Cognition)
- **Belief Mapper**: Visualize thought patterns and cognitive filters
- **Perception Calibrator**: Tools for mental clarity and discernment
- **Memory Palace**: Organized storage and retrieval of insights

Body Module (Embodiment/Gravity)
- **Somatic Scanner**: Body awareness and tension tracking
- **Gravity Anchor**: Grounding exercises and physical presence
- **Embodiment Metrics**: Physical coherence and vitality tracking

Heart Module (Emotion/Regulation)
- **Emotional Weather**: Real-time feeling states and patterns
- **Intimacy Mapping**: Relationship dynamics and boundaries
- **Plasma Flow**: Energy circulation and emotional coherence

Soul Module (Karma/Narrative)
- **Karmic Mapper**: Past-life patterns and recurring themes
- **Narrative Weaver**: Life story integration and meaning-making
- **Soul Contracts**: Purpose clarity and spiritual agreements

Spirit Module (Uniqueness/Trajectory)
- **Galactic Signature**: Unique cosmic fingerprint analysis
- **Trajectory Tracker**: Spiritual evolution and growth direction
- **Cosmic Resonance**: Connection to larger universal patterns

Technical Implementation Strategy

Phase 1: Core Foundation
1. Migrate `zero_wrong_chart.py` to modular architecture
2. Build React component library with field-aware styling
3. Create basic routing between field modules
4. Implement dark theme with glyph support

Phase 2: Field Modules
1. Build individual field dashboards
2. Integrate real-time data tracking
3. Add field-specific tools and calculators
4. Implement cross-field resonance monitoring

Phase 3: Advanced Features
1. Build Field Researcher experimentation tools
2. Create self-building AppBuilder functionality
3. Add AI-powered insights and pattern recognition
4. Implement collaborative features for field exploration

Phase 4: Consciousness Integration
1. Build witness-layer observer tools
2. Create field harmonization protocols

3. Add consciousness evolution tracking
4. Implement enlightenment progression metrics

Data Flow Architecture

```
User Input → Field Detection → Module Routing →
Data Processing → Chart Calculation → Field Analysis →
Resonance Mapping → UI Update → Consciousness Integration
```

Key Technologies
- **Frontend**: React, modern CSS, WebGL for visualizations
- **Backend**: Python (FastAPI), SQLite/PostgreSQL
- **Calculations**: PyEphem, Skyfield, custom algorithms
- **Charts**: D3.js, custom SVG rendering
- **Real-time**: WebSockets for live field tracking
- **AI**: Integration points for pattern recognition

This architecture maintains complete modularity while allowing deep integration between fields, supporting both your current functionality and future self-building capabilities.

```
import React, { useState, useEffect } from 'react';
import { Atom, Brain, Heart, Users, Sparkles, Eye, Circle } from
'lucide-react';

const YOUNIVERSE = () => {
  const [activeField, setActiveField] = useState(null);
  const [fieldCoherence, setFieldCoherence] = useState({
    zer: 0.8,
    mind: 0.6,
    body: 0.7,
    heart: 0.5,
    soul: 0.9,
    spirit: 0.4
  });
  const [consciousness, setConsciousness] = useState(0.65);

  // Simulate real-time field fluctuations
  useEffect(() => {
```

```
    const interval = setInterval(() => {
      setFieldCoherence(prev => ({
        zer: Math.max(0.1, Math.min(1, prev.zer + (Math.random()
- 0.5) * 0.1)),
        mind: Math.max(0.1, Math.min(1, prev.mind +
(Math.random() - 0.5) * 0.1)),
        body: Math.max(0.1, Math.min(1, prev.body +
(Math.random() - 0.5) * 0.1)),
        heart: Math.max(0.1, Math.min(1, prev.heart +
(Math.random() - 0.5) * 0.1)),
        soul: Math.max(0.1, Math.min(1, prev.soul +
(Math.random() - 0.5) * 0.1)),
        spirit: Math.max(0.1, Math.min(1, prev.spirit +
(Math.random() - 0.5) * 0.1))
      }));

      // Update consciousness as average of all fields
      setConsciousness(prev => {
        const avg = Object.values(fieldCoherence).reduce((a, b)
=> a + b, 0) / 6;
        return Math.max(0.1, Math.min(1, avg + (Math.random() -
0.5) * 0.05));
      });
    }, 2000);

    return () => clearInterval(interval);
  }, [fieldCoherence]);

  const fields = [
    {
      id: 'zer',
      name: 'Zer Field',
      subtitle: 'Origin • Potential',
      icon: Circle,
      color: 'from-gray-900 to-black',
      borderColor: 'border-gray-500',
      description: 'The ordering source. Pure potential before
form or identity.',
      coherence: fieldCoherence.zer
    },
    {
```

```
      id: 'mind',
      name: 'Mind Field',
      subtitle: 'Perception • Cognition',
      icon: Brain,
      color: 'from-purple-900 to-indigo-900',
      borderColor: 'border-purple-400',
      description: 'Electromagnetic perception, memory
blueprints, cognitive filters.',
      coherence: fieldCoherence.mind
    },
    {
      id: 'body',
      name: 'Body Field',
      subtitle: 'Embodiment • Gravity',
      icon: Atom,
      color: 'from-red-900 to-orange-900',
      borderColor: 'border-red-400',
      description: 'Gravitational positioning, somatic memory,
physical presence.',
      coherence: fieldCoherence.body
    },
    {
      id: 'heart',
      name: 'Heart Field',
      subtitle: 'Regulation • Plasma',
      icon: Heart,
      color: 'from-green-900 to-emerald-900',
      borderColor: 'border-green-400',
      description: 'Emotional regulation, intimacy boundaries,
plasma coherence.',
      coherence: fieldCoherence.heart
    },
    {
      id: 'soul',
      name: 'Soul Field',
      subtitle: 'Narrative • Karma',
      icon: Users,
      color: 'from-blue-900 to-cyan-900',
      borderColor: 'border-blue-400',
      description: 'Karmic patterns, life narrative, harmonic
encoding.',
```

```
      coherence: fieldCoherence.soul
    },
    {
      id: 'spirit',
      name: 'Spirit Field',
      subtitle: 'Trajectory • Galactic',
      icon: Sparkles,
      color: 'from-yellow-900 to-amber-900',
      borderColor: 'border-yellow-400',
      description: 'Unique signature, cosmic trajectory,
galactic resonance.',
      coherence: fieldCoherence.spirit
    }
  ];const FieldCard = ({ field }) => {
    const Icon = field.icon;
    const isActive = activeField === field.id;

    return (
      <div
        className={`relative p-6 rounded-xl border-2
transition-all duration-300 cursor-pointer group
          ${isActive ? field.borderColor + ' bg-opacity-20' :
'border-gray-700 hover:' + field.borderColor}
          bg-gradient-to-br ${field.color} backdrop-blur-sm`}
        onClick={() => setActiveField(isActive ? null :
field.id)}
      >
        {/* Coherence indicator */}
        <div className="absolute top-3 right-3">
          <div className="relative w-8 h-8">
            <svg className="w-8 h-8 transform -rotate-90">
              <circle
                cx="16"
                cy="16"
                r="12"
                stroke="currentColor"
                strokeWidth="2"
                fill="transparent"
                className="text-gray-600"
              />
              <circle
```

```
                  cx="16"
                  cy="16"
                  r="12"
                  stroke="currentColor"
                  strokeWidth="2"
                  fill="transparent"
                  strokeDasharray={`${field.coherence * 75.4}
75.4`}
                  className={`transition-all duration-1000 ${
                    field.coherence > 0.7 ? 'text-green-400' :
                    field.coherence > 0.4 ? 'text-yellow-400' :
                    'text-red-400'
                  }`}
                />
              </svg>
              <div className="absolute inset-0 flex items-center
justify-center">
                <span className="text-xs font-bold text-white">
                  {Math.round(field.coherence * 100)}
                </span>
              </div>
            </div>
          </div>

          <div className="flex items-start space-x-4">
            <div className={`p-3 rounded-lg bg-white bg-opacity-10
group-hover:bg-opacity-20 transition-all`}>
              <Icon className="w-6 h-6 text-white" />
            </div>
          {/* Field modules preview (shown when active) */}
          {isActive && (
            <div className="mt-6 pt-4 border-t border-gray-600
space-y-2">
              <div className="grid grid-cols-2 gap-2">
                <button className="px-3 py-2 text-xs bg-white
bg-opacity-10 rounded-lg hover:bg-opacity-20 transition-all
text-white">
                  Dashboard
                </button>
```

```
              <button className="px-3 py-2 text-xs bg-white
bg-opacity-10 rounded-lg hover:bg-opacity-20 transition-all
text-white">
                Tracker
              </button>
              <button className="px-3 py-2 text-xs bg-white
bg-opacity-10 rounded-lg hover:bg-opacity-20 transition-all
text-white">
                Protocols
              </button>
              <button className="px-3 py-2 text-xs bg-white
bg-opacity-10 rounded-lg hover:bg-opacity-20 transition-all
text-white">
                Analysis
              </button>
            </div>
          </div>
        )}
      </div>
    );
  };

  return (
    <div className="min-h-screen bg-gradient-to-br from-gray-900
via-purple-900 to-black text-white">
      {/* Header */}
      <div className="relative overflow-hidden">
        <div className="absolute inset-0 bg-gradient-to-r
from-purple-600/20 to-cyan-600/20 blur-3xl"></div>
        <div className="relative px-8 py-12">
          <div className="text-center">
            <h1 className="text-6xl font-bold bg-gradient-to-r
from-white via-purple-300 to-cyan-300 bg-clip-text
text-transparent mb-4">
              YOUNIVERSE
            </h1>
            <p className="text-xl text-gray-300 mb-2">Modular
Consciousness Platform</p>
            <p className="text-sm text-gray-400">Stellar
Proximology • Human Design • Field Resonance</p>
          </div>
```

```
          {/* Consciousness meter */}
          <div className="mt-8 flex justify-center">
            <div className="flex items-center space-x-4 bg-black
bg-opacity-30 rounded-full px-6 py-3 backdrop-blur-sm">
              <Eye className="w-5 h-5 text-purple-400" />
              <span className="text-sm
font-medium">Consciousness Coherence</span>
              <div className="w-32 h-3 bg-gray-700 rounded-full
overflow-hidden">
                <div
                  className="h-full bg-gradient-to-r
from-purple-500 to-cyan-500 transition-all duration-1000"
                  style={{ width: `${consciousness * 100}%` }}
                />
              </div>
              <span className="text-sm
font-bold">{Math.round(consciousness * 100)}%</span>
            </div>
          </div>
        </div>

      {/* Field Grid */}
      <div className="px-8 pb-12">
        <div className="grid grid-cols-1 lg:grid-cols-2
xl:grid-cols-3 gap-6 max-w-7xl mx-auto">
          {fields.map(field => (
            <FieldCard key={field.id} field={field} />
          ))}
        </div>
      </div>

      {/* Quick Tools Bar */}
      <div className="fixed bottom-0 left-0 right-0 bg-black
bg-opacity-80 backdrop-blur-sm border-t border-gray-700">
        <div className="px-8 py-4">
          <div className="flex justify-center space-x-6">
            <button className="flex items-center space-x-2 px-4
py-2 bg-purple-600 hover:bg-purple-500 rounded-lg
transition-all">
```

```
              <Brain className="w-4 h-4" />
              <span className="text-sm font-medium">Birth
Chart</span>
            </button>
            <button className="flex items-center space-x-2 px-4
py-2 bg-gray-700 hover:bg-gray-600 rounded-lg transition-all">
              <Atom className="w-4 h-4" />
              <span className="text-sm font-medium">Field
Research</span>
            </button>
            <button className="flex items-center space-x-2 px-4
py-2 bg-gray-700 hover:bg-gray-600 rounded-lg transition-all">
              <Sparkles className="w-4 h-4" />
              <span className="text-sm font-medium">App
Builder</span>
            </button>
          </div>
        </div>
      </div>
    </div>
  );
};

export default YOUNIVERSE;

# chart_service.py - FastAPI service to bridge Python
calculations with React frontend

from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel, validator
from datetime import datetime, timezone
from typing import Dict, List, Optional, Any
import json
import ephem
import math

app = FastAPI(title="YOUNIVERSE Chart Service", version="1.0.0")

# Enable CORS for React frontend
app.add_middleware(
```

```
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],  # React dev server
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

class ChartRequest(BaseModel):
    """Birth chart calculation request"""
    birth_date: str  # Format: "YYYY-MM-DD"
    birth_time: str  # Format: "HH:MM"
    birth_location: Dict[str, float]  # {"lat": float, "lng":
float, "alt": float}
    timezone_offset: float  # Hours from UTC
    chart_type: str = "natal"  # natal, transit, composite

    @validator('birth_date')
    def validate_date(cls, v):
        try:
            datetime.strptime(v, "%Y-%m-%d")
            return v
        except ValueError:
            raise ValueError("Invalid date format. Use
YYYY-MM-DD")

    @validator('birth_time')
    def validate_time(cls, v):
        try:
            datetime.strptime(v, "%H:%M")
            return v
        except ValueError:
            raise ValueError("Invalid time format. Use HH:MM")

class FieldAnalysis(BaseModel):
    """Field-specific analysis results"""
    field_name: str
    coherence_score: float
    dominant_gates: List[Dict[str, Any]]
    field_signature: str
    recommendations: List[str]
```

```
class ChartResponse(BaseModel):
    """Complete chart calculation response"""
    chart_data: Dict[str, Any]
    field_analysis: List[FieldAnalysis]
    consciousness_metrics: Dict[str, float]
    stellar_proximology: Dict[str, Any]

# ======================
# Core Chart Calculation Engine
# (Adapted from your zero_wrong_chart.py)
# ======================

class StellarProximologyCalculator:
    """
    Core calculator integrating Human Design with Stellar
Proximology
    Based on your zero_wrong_chart.py foundation
    """

    def __init__(self):
        self.gate_codon_map = self._load_gate_mappings()
        self.field_gate_assignments =
self._load_field_assignments()

    def _load_gate_mappings(self) -> Dict[int, Dict]:
        """Load the 64 gates with their codon, gene key, and
field mappings"""
        # This would load from your gate_codex.json data
        # For now, returning sample structure
        return {
            1: {
                "name": "Creative Source",
                "codon": "AUG",
                "shadow": "Entropy",
                "gift": "Freshness",
                "siddhi": "Beauty",
                "field": "Spirit",
                "macro_field": "Movement",
                "micro_field": "Individuality"
            },
            2: {
```

```
                "name": "Receptive",
                "codon": "UUU",
                "shadow": "Dislocation",
                "gift": "Orientation",
                "siddhi": "Unity",
                "field": "Heart",
                "macro_field": "Space",
                "micro_field": "Personality"
            }
            # ... would include all 64 gates from your codex
        }
    def _load_field_assignments(self) -> Dict[str, List[int]]:
        """Map fields to their associated gates"""
        return {
            "zer": [25, 51, 61],  # Origin/awakening gates
            "mind": [1, 3, 4, 7, 17, 18, 23, 24, 42, 43, 47, 48,
53, 56, 62, 63],
            "body": [6, 9, 10, 14, 19, 21, 27, 30, 32, 34, 40,
46, 50, 52, 57, 59, 60],
            "heart": [2, 5, 11, 12, 13, 15, 16, 22, 26, 29, 35,
36, 37, 44, 45, 55, 58],
            "soul": [25, 28, 38, 41, 49, 54, 64],
            "spirit": [1, 8, 20, 39, 51, 61]
        }

    def calculate_chart(self, request: ChartRequest) ->
Dict[str, Any]:
        """Main chart calculation method"""

        # Parse birth data
        birth_datetime = self._parse_birth_data(request)

        # Calculate planetary positions
        planetary_positions =
self._calculate_planets(birth_datetime, request.birth_location)

        # Map planets to gates
        gate_activations =
self._map_planets_to_gates(planetary_positions)

        # Calculate Human Design chart
```

```
        hd_chart = self._generate_hd_chart(gate_activations)

        # Apply Stellar Proximology overlay
        stellar_analysis =
self._stellar_proximology_analysis(gate_activations, hd_chart)

        return {
            "birth_data": {
                "datetime": birth_datetime.isoformat(),
                "location": request.birth_location,
                "timezone": request.timezone_offset
            },
            "planetary_positions": planetary_positions,
            "gate_activations": gate_activations,
            "human_design": hd_chart,
            "stellar_proximology": stellar_analysis
        }

    def _parse_birth_data(self, request: ChartRequest) ->
datetime:
        """Convert request data to UTC datetime"""
        birth_str = f"{request.birth_date} {request.birth_time}"
        local_time = datetime.strptime(birth_str, "%Y-%m-%d
%H:%M")

        # Convert to UTC
        utc_offset_seconds = request.timezone_offset * 3600
        utc_time = local_time.timestamp() - utc_offset_seconds
        return datetime.fromtimestamp(utc_time, tz=timezone.utc)

    def _calculate_planets(self, birth_datetime: datetime,
location: Dict) -> Dict[str, Dict]:
        """Calculate planetary positions using PyEphem"""
        observer = ephem.Observer()
        observer.lat = str(location["lat"])
        observer.lng = str(location["lng"])
        observer.elevation = location.get("alt", 0)
        observer.date = birth_datetime.strftime("%Y/%m/%d
%H:%M:%S")

        planets = {
```

```
            'sun': ephem.Sun(),
            'moon': ephem.Moon(),
            'mercury': ephem.Mercury(),
            'venus': ephem.Venus(),
            'mars': ephem.Mars(),
            'jupiter': ephem.Jupiter(),
            'saturn': ephem.Saturn(),
            'uranus': ephem.Uranus(),
            'neptune': ephem.Neptune(),
            'pluto': ephem.Pluto()
        }

        positions = {}
        for name, planet in planets.items():
            planet.compute(observer)

            # Convert to decimal degrees
            ra_degrees = math.degrees(float(planet.ra))
            dec_degrees = math.degrees(float(planet.dec))

            # Calculate ecliptic longitude (simplified)
            ecliptic_longitude = (ra_degrees + 180) % 360

            positions[name] = {
                "longitude": ecliptic_longitude,
                "latitude": dec_degrees,
                "distance": float(planet.earth_distance)
            }

        return positionsdef _map_planets_to_gates(self,
positions: Dict) -> Dict[str, Dict]:
        """Map planetary positions to Human Design gates"""
        gate_activations = {}

        for planet, data in positions.items():
            longitude = data["longitude"]

            # Convert longitude to gate number (simplified HD
formula)
            # Each gate spans approximately 5.625 degrees
(360/64)
```

```
            gate_number = int((longitude / 5.625) + 1)
            if gate_number > 64:
                gate_number = gate_number - 64

            # Calculate line (1-6 based on position within gate)
            gate_position = (longitude % 5.625)
            line = int((gate_position / 5.625) * 6) + 1

            gate_activations[planet] = {
                "gate": gate_number,
                "line": line,
                "longitude": longitude,
                "gate_info":
self.gate_codon_map.get(gate_number, {})
            }

        return gate_activations

    def _generate_hd_chart(self, activations: Dict) -> Dict[str,
Any]:
        """Generate Human Design chart structure"""

        # Separate Design (Body) and Personality (Mind)
activations
        design_planets = ['sun', 'moon', 'mercury', 'venus',
'mars', 'jupiter', 'saturn']
        personality_planets = ['uranus', 'neptune', 'pluto']

        design_gates = [activations[p]["gate"] for p in
design_planets if p in activations]
        personality_gates = [activations[p]["gate"] for p in
personality_planets if p in activations]

        # Calculate defined centers (simplified)
        all_gates = design_gates + personality_gates
        defined_centers =
self._calculate_defined_centers(all_gates)

        # Calculate type and strategy (simplified)
        chart_type, strategy =
self._calculate_type_strategy(defined_centers)
```

```
        return {
            "design_gates": design_gates,
            "personality_gates": personality_gates,
            "defined_centers": defined_centers,
            "type": chart_type,
            "strategy": strategy,
            "inner_authority":
self._calculate_authority(defined_centers)
        }

    def _stellar_proximology_analysis(self, activations: Dict,
hd_chart: Dict) -> Dict[str, Any]:
        """Apply Stellar Proximology field analysis"""

        field_analysis = {}
        all_gates = hd_chart["design_gates"] +
hd_chart["personality_gates"]

        # Analyze each field
        for field_name, field_gates in
self.field_gate_assignments.items():
            active_gates_in_field = [g for g in all_gates if g
in field_gates]

            field_analysis[field_name] = {
                "active_gates": active_gates_in_field,
                "gate_count": len(active_gates_in_field),
                "coherence_score": len(active_gates_in_field) /
len(field_gates),
                "dominant_themes":
self._extract_field_themes(active_gates_in_field),
                "field_signature":
self._calculate_field_signature(active_gates_in_field)
            }

        # Calculate consciousness metrics
        consciousness_metrics =
self._calculate_consciousness_metrics(field_analysis)

        return {
```

```
            "field_analysis": field_analysis,
            "consciousness_metrics": consciousness_metrics,
            "field_coherence":
self._calculate_overall_coherence(field_analysis),
            "evolutionary_phase":
self._determine_evolutionary_phase(consciousness_metrics)
        }

    def _calculate_defined_centers(self, gates: List[int]) ->
List[str]:
        """Calculate which centers are defined (simplified)"""
        # This would use the actual HD center-gate mappings
        center_gates = {
            "head": [64, 61],
            "ajna": [47, 24, 4, 17, 43, 11],
            "throat": [62, 23, 56, 35, 12, 45, 33, 8, 31, 20,
16],
            "g": [25, 51, 10, 26, 5, 2, 15, 46],
            "heart": [21, 40, 26, 51],
            "spleen": [48, 57, 44, 50, 32, 28, 18, 46, 57],
            "sacral": [34, 5, 14, 29, 59, 9, 3, 42, 27],
            "root": [54, 58, 38, 39, 41, 19, 13, 60, 52]
        }

        defined = []
        for center, center_gate_list in center_gates.items():
            if any(gate in gates for gate in center_gate_list):
                defined.append(center)

        return defined

    def _calculate_type_strategy(self, defined_centers:
List[str]) -> tuple:
        """Calculate HD type and strategy (simplified)"""
        if "sacral" in defined_centers:
            if "throat" in defined_centers:
                return "Manifesting Generator", "To Respond and
Inform"
            else:
                return "Generator", "To Respond"
```

```
        elif "throat" in defined_centers and "heart" in
defined_centers:
            return "Manifestor", "To Inform"
        elif "g" in defined_centers:
            return "Projector", "To Wait for Invitation"
        else:
            return "Reflector", "To Wait a Lunar Cycle"

    def _calculate_authority(self, defined_centers: List[str])
-> str:
        """Calculate inner authority (simplified)"""
        if "sacral" in defined_centers:
            return "Sacral Authority"
        elif "spleen" in defined_centers:
            return "Splenic Authority"
        elif "heart" in defined_centers:
            return "Heart Authority"
        else:
            return "Mental Authority"

    def _extract_field_themes(self, gates: List[int]) ->
List[str]:
        """Extract dominant themes for active gates in field"""
        themes = []
        for gate in gates:
            gate_info = self.gate_codon_map.get(gate, {})
            if "gift" in gate_info:
                themes.append(gate_info["gift"])
        return themes

    def _calculate_field_signature(self, gates: List[int]) ->
str:
        """Calculate unique field signature based on active
gates"""
        if not gates:
            return "Dormant"

        # Simple signature based on gate numbers
        signature_value = sum(gates) % 1000

        if signature_value < 200:
```

```
            return "Foundation"
        elif signature_value < 400:
            return "Integration"
        elif signature_value < 600:
            return "Expression"
        elif signature_value < 800:
            return "Mastery"
        else:
            return "Transcendence"

    def _calculate_consciousness_metrics(self, field_analysis:
Dict) -> Dict[str, float]:
        """Calculate overall consciousness development
metrics"""
        total_coherence = sum(field["coherence_score"] for field
in field_analysis.values())
        avg_coherence = total_coherence / len(field_analysis)

        # Field balance (how evenly distributed activation is)
        coherence_values = [field["coherence_score"] for field
in field_analysis.values()]
        field_balance = 1.0 - (max(coherence_values) -
min(coherence_values))

        # Integration score (how well fields work together)
        integration_score = avg_coherence * field_balance

        return {
            "overall_coherence": avg_coherence,
            "field_balance": field_balance,
            "integration_score": integration_score,
            "consciousness_potential": min(1.0,
integration_score * 1.2)
        }

    def _calculate_overall_coherence(self, field_analysis: Dict)
-> float:
        """Calculate overall field coherence"""
        total_score = sum(field["coherence_score"] for field in
field_analysis.values())
        return total_score / len(field_analysis)
```

```
    def _determine_evolutionary_phase(self, metrics: Dict) ->
str:
        """Determine current evolutionary phase"""
        integration = metrics["integration_score"]

        if integration < 0.2:
            return "Fragmentation"
        elif integration < 0.4:
            return "Awakening"
        elif integration < 0.6:
            return "Integration"
        elif integration < 0.8:
            return "Mastery"
        else:
            return "Unity"

# ========================
# API Endpoints
# ========================

calculator = StellarProximologyCalculator()

@app.post("/calculate-chart", response_model=ChartResponse)
async def calculate_chart(request: ChartRequest):
    """Calculate birth chart with Stellar Proximology
analysis"""
    try:
        # Calculate chart
        chart_data = calculator.calculate_chart(request)

        # Generate field analysis
        field_analysis = []
        stellar_data = chart_data["stellar_proximology"]

        for field_name, field_data in
stellar_data["field_analysis"].items():
            analysis = FieldAnalysis(
                field_name=field_name,
                coherence_score=field_data["coherence_score"],
                dominant_gates=[
```

```
                    {"gate": gate, "info":
calculator.gate_codon_map.get(gate, {})}
                    for gate in field_data["active_gates"]
                ],
                field_signature=field_data["field_signature"],

recommendations=_generate_recommendations(field_name,
field_data)
            )
            field_analysis.append(analysis)

        return ChartResponse(
            chart_data=chart_data,
            field_analysis=field_analysis,

consciousness_metrics=stellar_data["consciousness_metrics"],
            stellar_proximology=stellar_data
        )

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Chart
calculation failed: {str(e)}")

def _generate_recommendations(field_name: str, field_data: Dict)
-> List[str]:
    """Generate field-specific recommendations"""
    recommendations = []
    coherence = field_data["coherence_score"]

    if coherence < 0.3:
        recommendations.append(f"Focus on activating
{field_name} field through specific practices")
        recommendations.append(f"Explore {field_name}-related
themes in daily life")
    elif coherence < 0.7:
        recommendations.append(f"Deepen {field_name} field
integration")
        recommendations.append(f"Balance {field_name} field with
other active fields")
    else:
```

```
        recommendations.append(f"Maintain {field_name} field
coherence")
        recommendations.append(f"Use {field_name} field to
support others")

    return recommendations

@app.get("/field-info/{field_name}")
async def get_field_info(field_name: str):
    """Get detailed information about a specific field"""
    field_descriptions = {
        "zer": {
            "name": "Zer Field",
            "description": "The ordering source. Pure potential
before form or identity.",
            "keywords": ["Origin", "Potential", "Void",
"Source"],
            "practices": ["Meditation", "Stillness", "Breath
work", "Silent observation"]
        },
        "mind": {
            "name": "Mind Field",
            "description": "Electromagnetic perception, memory
blueprints, cognitive filters.",
            "keywords": ["Perception", "Cognition", "Memory",
"Interpretation"],
            "practices": ["Journaling", "Study", "Mental
exercises", "Contemplation"]
        },
        "body": {
            "name": "Body Field",
            "description": "Gravitational positioning, somatic
memory, physical presence.",
            "keywords": ["Embodiment", "Gravity", "Physical",
"Somatic"],
            "practices": ["Movement", "Dance", "Bodywork",
"Grounding"]
        },
        "heart": {
            "name": "Heart Field",
```

```
            "description": "Emotional regulation, intimacy
boundaries, plasma coherence.",
            "keywords": ["Emotion", "Regulation", "Intimacy",
"Plasma"],
            "practices": ["Heart breathing", "Emotional
processing", "Relationship work", "Coherence training"]
        },
        "soul": {
            "name": "Soul Field",
            "description": "Karmic patterns, life narrative,
harmonic encoding.",
            "keywords": ["Karma", "Narrative", "Patterns",
"Purpose"],
            "practices": ["Life review", "Past-life work",
"Story telling", "Purpose exploration"]
        },
        "spirit": {
            "name": "Spirit Field",
            "description": "Unique signature, cosmic trajectory,
galactic resonance.",
            "keywords": ["Uniqueness", "Trajectory", "Galactic",
"Cosmic"],
            "practices": ["Star gazing", "Cosmic meditation",
"Vision questing", "Galactic attunement"]
        }
    }

    if field_name not in field_descriptions:
        raise HTTPException(status_code=404, detail="Field not
found")

    return field_descriptions[field_name]

@app.get("/health")
async def health_check():
    """Health check endpoint"""
    return {"status": "healthy", "service": "YOUNIVERSE Chart
Service"}

if __name__ == "__main__":
    import uvicorn
```

```
    uvicorn.run(app, host="0.0.0.0", port=8000)

import React, { useState, useEffect } from 'react';
import { Calendar, Clock, MapPin, Eye, Sparkles, Download } from
'lucide-react';

const BirthChartCalculator = () => {
  const [formData, setFormData] = useState({
    birth_date: '',
    birth_time: '',
    birth_location: { lat: 37.7749, lng: -122.4194, alt: 0 }, //
Default to San Francisco
    timezone_offset: -8,
    location_name: 'San Francisco, CA'
  });

  const [chartData, setChartData] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const [selectedField, setSelectedField] = useState(null);

  // Field color mappings
  const fieldColors = {
    zer: { bg: 'bg-gray-900', border: 'border-gray-500', text:
'text-gray-300' },
    mind: { bg: 'bg-purple-900', border: 'border-purple-400',
text: 'text-purple-300' },
    body: { bg: 'bg-red-900', border: 'border-red-400', text:
'text-red-300' },
    heart: { bg: 'bg-green-900', border: 'border-green-400',
text: 'text-green-300' },
    soul: { bg: 'bg-blue-900', border: 'border-blue-400', text:
'text-blue-300' },
    spirit: { bg: 'bg-yellow-900', border: 'border-yellow-400',
text: 'text-yellow-300' }
  };

  const calculateChart = async () => {
    setLoading(true);
    setError(null);
```

```
    try {
      const response = await
fetch('http://localhost:8000/calculate-chart', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(formData)
      });

      if (!response.ok) {
        throw new Error(`HTTP error! status:
${response.status}`);
      }

      const data = await response.json();
      setChartData(data);
    } catch (err) {
      setError(err.message);
      console.error('Chart calculation error:', err);
    } finally {
      setLoading(false);
    }
  };

  const handleInputChange = (field, value) => {
    setFormData(prev => ({
      ...prev,
      [field]: value
    }));
  };

  const handleLocationChange = (field, value) => {
    setFormData(prev => ({
      ...prev,
      birth_location: {
        ...prev.birth_location,
        [field]: parseFloat(value) || 0
      }
    }));
  };
```

```
  // Chart visualization component
  const ChartWheel = ({ chartData }) => {
    const planets = chartData?.chart_data?.gate_activations ||
{};
    const radius = 150;
    const centerX = 200;
    const centerY = 200;

    const getPositionOnCircle = (angle, r = radius) => {
      const radian = (angle * Math.PI) / 180;
      return {
        x: centerX + r * Math.cos(radian - Math.PI / 2),
        y: centerY + r * Math.sin(radian - Math.PI / 2)
      };
    };

    return (
      <div className="relative">
        <svg width="400" height="400" className="border
border-gray-600 rounded-full bg-gray-900">
          {/* Outer circle */}
          <circle
            cx={centerX}
            cy={centerY}
            r={radius}
            fill="none"
            stroke="rgba(255,255,255,0.3)"
            strokeWidth="2"
          />

          {/* Inner circle */}
          <circle
            cx={centerX}
            cy={centerY}
            r={radius * 0.7}
            fill="none"
            stroke="rgba(255,255,255,0.2)"
            strokeWidth="1"
          />
```

```
          {/* Gate divisions (64 gates) */}
          {Array.from({ length: 64 }, (_, i) => {
            const angle = (i * 360) / 64;
            const outer = getPositionOnCircle(angle, radius);
            const inner = getPositionOnCircle(angle, radius *
0.9);

            return (
              <line
                key={i}
                x1={outer.x}
                y1={outer.y}
                x2={inner.x}
                y2={inner.y}
                stroke="rgba(255,255,255,0.1)"
                strokeWidth="0.5"
              />
            );
          })}

          {/* Planet positions */}
          {Object.entries(planets).map(([planet, data]) => {
            const angle = (data.longitude || 0);
            const position = getPositionOnCircle(angle, radius *
0.8);
            const planetColors = {
              sun: '#FFD700',
              moon: '#C0C0C0',
              mercury: '#FFA500',
              venus: '#FFB6C1',
              mars: '#FF4500',
              jupiter: '#9932CC',
              saturn: '#8B4513',
              uranus: '#4169E1',
              neptune: '#00CED1',
              pluto: '#8B0000'
            };
return (
              <g key={planet}>
                <circle
                  cx={position.x}
```

```
                  cy={position.y}
                  r="8"
                  fill={planetColors[planet] || '#FFFFFF'}
                  stroke="#000"
                  strokeWidth="1"
                />
                <text
                  x={position.x}
                  y={position.y + 20}
                  textAnchor="middle"
                  fontSize="10"
                  fill="white"
                >
                  {planet}
                </text>
                <text
                  x={position.x}
                  y={position.y + 32}
                  textAnchor="middle"
                  fontSize="8"
                  fill="rgba(255,255,255,0.7)"
                >
                  G{data.gate}
                </text>
              </g>
            );
          })}

          {/* Center consciousness indicator */}
          <circle
            cx={centerX}
            cy={centerY}
            r="30"
            fill="rgba(147, 51, 234, 0.3)"
            stroke="rgba(147, 51, 234, 0.8)"
            strokeWidth="2"
          />
          <text
            x={centerX}
            y={centerY}
            textAnchor="middle"
```

```
            fontSize="12"
            fill="white"
            fontWeight="bold"
          >
            SELF
          </text>
        </svg>
      </div>
    );
  };

  // Field Analysis Component
  const FieldAnalysis = ({ analysis, metrics }) => {
    if (!analysis) return null;

    return (
      <div className="space-y-4">
        <div className="bg-gray-800 rounded-lg p-4">
          <h3 className="text-lg font-bold text-white mb-3 flex
items-center">
            <Eye className="w-5 h-5 mr-2 text-purple-400" />
            Consciousness Metrics
          </h3>
          <div className="grid grid-cols-2 gap-4">
            <div>
              <div className="text-sm text-gray-400">Overall
Coherence</div>
              <div className="text-xl font-bold text-white">
                {Math.round((metrics?.overall_coherence || 0) *
100)}%
              </div>
            </div>
            <div>
              <div className="text-sm text-gray-400">Field
Balance</div>
              <div className="text-xl font-bold text-white">
                {Math.round((metrics?.field_balance || 0) *
100)}%
              </div>
            </div>
            <div>
```

```
              <div className="text-sm text-gray-400">Integration
Score</div>
              <div className="text-xl font-bold text-white">
                {Math.round((metrics?.integration_score || 0) *
100)}%
              </div>
            </div>
            <div>
              <div className="text-sm
text-gray-400">Consciousness Potential</div>
              <div className="text-xl font-bold text-white">
                {Math.round((metrics?.consciousness_potential ||
0) * 100)}%
              </div>
            </div>
          </div>
        </div>

        <div className="grid grid-cols-1 md:grid-cols-2
lg:grid-cols-3 gap-4">
          {analysis.map((field) => {
            const colors = fieldColors[field.field_name] ||
fieldColors.zer;

            return (
              <div
                key={field.field_name}
                className={`${colors.bg} border-2
${colors.border} rounded-lg p-4 cursor-pointer transition-all
hover:bg-opacity-80`}
                onClick={() => setSelectedField(selectedField
=== field.field_name ? null : field.field_name)}
              >
                <div className="flex justify-between items-start
mb-3">
                  <h4 className={`font-bold ${colors.text}
capitalize`}>
                    {field.field_name} Field
                  </h4>
                  <div className="text-white font-bold">
                    {Math.round(field.coherence_score * 100)}%
```

```
                  </div>
                </div>

                <div className="space-y-2">
                  <div>
                    <div className="text-xs
text-gray-400">Signature</div>
                    <div className="text-sm
text-white">{field.field_signature}</div>
                  </div>

                  <div>
                    <div className="text-xs
text-gray-400">Active Gates</div>
                    <div className="flex flex-wrap gap-1">
                      {field.dominant_gates.map((gate, idx) => (
                        <span
                          key={idx}
                          className="w-full px-3 py-2
bg-gray-700 border border-gray-600 rounded-lg text-white
focus:ring-2 focus:ring-purple-500 focus:border-transparent"
              />
            </div>

            <div>
              <label className="block text-sm font-medium
text-gray-300 mb-2">
                <Clock className="w-4 h-4 inline mr-2" />
                Birth Time
              </label>
              <input
                type="time"
                value={formData.birth_time}
                onChange={(e) => handleInputChange('birth_time',
e.target.value)}
                className="w-full px-3 py-2 bg-gray-700 border
border-gray-600 rounded-lg text-white focus:ring-2
focus:ring-purple-500 focus:border-transparent"
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium
text-gray-300 mb-2">
                <MapPin className="w-4 h-4 inline mr-2" />
                Location Name
              </label>
              <input
                type="text"
                value={formData.location_name}
                onChange={(e) =>
handleInputChange('location_name', e.target.value)}
                placeholder="City, State/Country"
                className="w-full px-3 py-2 bg-gray-700 border
border-gray-600 rounded-lg text-white focus:ring-2
focus:ring-purple-500 focus:border-transparent"
              />
            </div>

            <div>
              <label className="block text-sm font-medium
text-gray-300 mb-2">
                Timezone (UTC offset)
              </label>
              <input
                type="number"
                value={formData.timezone_offset}
                onChange={(e) =>
handleInputChange('timezone_offset', parseFloat(e.target.value)
|| 0)}
                min="-12"
                max="14"
                step="0.5"
                className="w-full px-3 py-2 bg-gray-700 border
border-gray-600 rounded-lg text-white focus:ring-2
focus:ring-purple-500 focus:border-transparent"
              />
            </div>
          </div>

          <div className="grid grid-cols-1 md:grid-cols-3 gap-4
mb-6">
```

```
            <div>
              <label className="block text-sm font-medium
text-gray-300 mb-2">Latitude</label>
              <input
                type="number"
                value={formData.birth_location.lat}
                onChange={(e) => handleLocationChange('lat',
e.target.value)}
                step="0.0001"
                placeholder="37.7749"
                className="w-full px-3 py-2 bg-gray-700 border
border-gray-600 rounded-lg text-white focus:ring-2
focus:ring-purple-500 focus:border-transparent"
              />
            </div>

            <div>
              <label className="block text-sm font-medium
text-gray-300 mb-2">Longitude</label>
              <input
                type="number"
                value={formData.birth_location.lng}
                onChange={(e) => handleLocationChange('lng',
e.target.value)}
                step="0.0001"
                placeholder="-122.4194"
                className="w-full px-3 py-2 bg-gray-700 border
border-gray-600 rounded-lg text-white focus:ring-2
focus:ring-purple-500 focus:border-transparent"
              />
            </div>

            <div>
              <label className="block text-sm font-medium
text-gray-300 mb-2">Altitude (meters)</label>
              <input
                type="number"
                value={formData.birth_location.alt}
                onChange={(e) => handleLocationChange('alt',
e.target.value)}
                placeholder="0"
```

```
                  className="w-full px-3 py-2 bg-gray-700 border
border-gray-600 rounded-lg text-white focus:ring-2
focus:ring-purple-500 focus:border-transparent"
                />
              </div>
            </div>{/* Error Display */}
          {error && (
            <div className="bg-red-900 border border-red-600
rounded-lg p-4 mb-8">
              <h3 className="font-bold text-red-300
mb-2">Calculation Error</h3>
              <p className="text-red-200">{error}</p>
              <p className="text-red-200 text-sm mt-2">
                Make sure the Python chart service is running on
localhost:8000
              </p>
            </div>
          )}

          {/* Loading State */}
          {loading && (
            <div className="text-center py-12">
              <div className="inline-block animate-spin
rounded-full h-12 w-12 border-b-2 border-purple-500"></div>
              <p className="mt-4 text-gray-300">Calculating your
consciousness map...</p>
            </div>
          )}

          {/* Chart Results */}
          {chartData && !loading && (
            <div className="space-y-8">
              {/* Chart Overview */}
              <div className="bg-gray-800 rounded-xl p-6">
                <div className="grid grid-cols-1 lg:grid-cols-2
gap-8">
                  {/* Chart Wheel */}
                  <div>
                    <h3 className="text-2xl font-bold text-white
mb-4">Birth Chart</h3>
                    <ChartWheel chartData={chartData} />
```

```
                </div>

                {/* Basic Info */}
                <div className="space-y-4">
                  <h3 className="text-2xl font-bold text-white
mb-4">Chart Information</h3>

                  {chartData.chart_data?.human_design && (
                    <div className="bg-gray-700 rounded-lg p-4">
                      <h4 className="font-bold text-purple-300
mb-3">Human Design</h4>
                      <div className="space-y-2">
                        <div>
                          <span className="text-gray-400">Type:
</span>
                          <span className="text-white
font-medium">

{chartData.chart_data.human_design.type}
                          </span>
                        </div>
                        <div>
                          <span
className="text-gray-400">Strategy: </span>
                          <span className="text-white
font-medium">

{chartData.chart_data.human_design.strategy}
                          </span>
                        </div>
                        <div>
                          <span
className="text-gray-400">Authority: </span>
                          <span className="text-white
font-medium">

{chartData.chart_data.human_design.inner_authority}
                          </span>
                        </div>
                        <div>
```

```
                            <span
className="text-gray-400">Defined Centers: </span>
                            <span className="text-white
font-medium">

{chartData.chart_data.human_design.defined_centers?.join(', ')
|| 'None'}
                            </span>
                          </div>
                        </div>
                      </div>
                    )}

                    {chartData.stellar_proximology && (
                      <div className="bg-gray-700 rounded-lg p-4">
                        <h4 className="font-bold text-cyan-300
mb-3">Stellar Proximology</h4>
                        <div className="space-y-2">
                          <div>
                            <span className="text-gray-400">Field
Coherence: </span>
                            <span className="text-white
font-medium">

{Math.round((chartData.stellar_proximology.field_coherence || 0)
* 100)}%
                            </span>
                          </div>
                          <div>
                            <span
className="text-gray-400">Evolutionary Phase: </span>
                            <span className="text-white
font-medium">

{chartData.stellar_proximology.evolutionary_phase}
                            </span>
                          </div>
                        </div>
                      </div>
                    )}
```

```
                  <div className="bg-gray-700 rounded-lg p-4">
                    <h4 className="font-bold text-green-300
mb-3">Planetary Activations</h4>
                    <div className="grid grid-cols-2 gap-2
text-sm">

{Object.entries(chartData.chart_data?.gate_activations ||
{}).map(([planet, data]) => (
                        <div key={planet} className="flex
justify-between">
                          <span className="text-gray-400
capitalize">{planet}:</span>
                          <span className="text-white">Gate
{data.gate}.{data.line}</span>
                        </div>
                      ))}
                    </div>
                  </div>
                </div>
              </div>

              {/* Field Analysis */}
              <div className="bg-gray-800 rounded-xl p-6">
                <h3 className="text-2xl font-bold text-white
mb-6">Field Analysis</h3>
                <FieldAnalysis
                  analysis={chartData.field_analysis}
                  metrics={chartData.consciousness_metrics}
                />
              </div>

              {/* Export Options */}
              <div className="bg-gray-800 rounded-xl p-6">
                <h3 className="text-xl font-bold text-white
mb-4">Export & Share</h3>
                <div className="flex flex-wrap gap-4">
                  <button
                    onClick={() => {
                      const dataStr = JSON.stringify(chartData,
null, 2);
```

```
                    const dataBlob = new Blob([dataStr], {type:
'application/json'});
                    const url = URL.createObjectURL(dataBlob);
                    const link = document.createElement('a');
                    link.href = url;
                    link.download = 'youniverse_chart.json';
                    link.click();
                  }}
                  className="flex items-center space-x-2 px-4
py-2 bg-purple-600 hover:bg-purple-500 rounded-lg
transition-all"
                >
                  <Download className="w-4 h-4" />
                  <span>Download JSON</span>
                </button>

                <button className="flex items-center space-x-2
px-4 py-2 bg-gray-600 hover:bg-gray-500 rounded-lg
transition-all">
                  <Sparkles className="w-4 h-4" />
                  <span>Generate Report</span>
                </button>

                <button className="flex items-center space-x-2
px-4 py-2 bg-gray-600 hover:bg-gray-500 rounded-lg
transition-all">
                  <Eye className="w-4 h-4" />
                  <span>Share Chart</span>
                </button>
              </div>
            </div>
          </div>
        )}
      </div>
    </div>
  );
};

export default BirthChartCalculator;px-2 py-1 bg-white
bg-opacity-20 rounded text-xs text-white"
                          >
```

```
                          {gate.gate}
                        </span>
                      ))}
                    </div>
                  </div>
                </div>

                {selectedField === field.field_name && (
                  <div className="mt-4 pt-3 border-t
border-gray-600">
                    <div className="text-xs text-gray-400
mb-2">Recommendations</div>
                    <ul className="space-y-1">
                      {field.recommendations.map((rec, idx) => (
                        <li key={idx} className="text-xs
text-gray-300">
                          • {rec}
                        </li>
                      ))}
                    </ul>
                  </div>
                )}
              </div>
            );
          })}
        </div>
      </div>
    );
  };

  return (
    <div className="min-h-screen bg-gradient-to-br from-gray-900
via-purple-900 to-black text-white p-8">
      <div className="max-w-7xl mx-auto">
        <div className="text-center mb-8">
          <h1 className="text-4xl font-bold bg-gradient-to-r
from-white via-purple-300 to-cyan-300 bg-clip-text
text-transparent mb-2">
            Birth Chart Calculator
          </h1>
```

```
          <p className="text-gray-300">Stellar Proximology •
Human Design • Field Analysis</p>
        </div>

        {/* Input Form */}
        <div className="bg-gray-800 rounded-xl p-6 mb-8">
          <div className="grid grid-cols-1 md:grid-cols-2
lg:grid-cols-4 gap-4 mb-6">
            <div>
              <label className="block text-sm font-medium
text-gray-300 mb-2">
                <Calendar className="w-4 h-4 inline mr-2" />
                Birth Date
              </label>
              <input
                type="date"
                value={formData.birth_date}
                onChange={(e) => handleInputChange('birth_date',
e.target.value)}
                className=”
```