This was the React Code I got Claude AI to make on its interface.

```
import React, { useState, useEffect } from 'react';
import { Heart, Brain, Database, Shield, Eye, Mic, Globe, Users, AlertTriangle, CheckCircle,
XCircle, Play, Pause } from 'lucide-react';

const PhoenixSystemPrototype = () => {
  const [input, setInput] = useState('');
  const [isProcessing, setIsProcessing] = useState(false);
  const [currentMoodNumeric, setCurrentMoodNumeric] = useState(0);
  const [processingStage, setProcessingStage] = useState('');
  const [nodeStates, setNodeStates] = useState({});
  const [guardianStates, setGuardianStates] = useState({});
  const [logs, setLogs] = useState([]);
  const [systemOutput, setSystemOutput] = useState(null);

  const phoenixNodes = [
    { id: 'heart', name: 'Heart ❤️', icon: Heart, role: 'Emotional Core', color: 'text-red-500' },
    { id: 'mind', name: 'Mind 🧠', icon: Brain, role: 'Cognitive Core', color: 'text-blue-500' },
    { id: 'soul', name: 'Soul 🌌', icon: Database, role: 'Memory & Trends', color: 'text-purple-500' },
    { id: 'body', name: 'Body 💪', icon: Shield, role: 'Stabilization', color: 'text-green-500' },
    { id: 'observer', name: 'Observer 👁️', icon: Eye, role: 'Meta Oversight', color: 'text-yellow-500'
},
  ];

  const guardianNodes = [
    { id: 'watcher', name: 'Watcher (Iggy)', role: 'Pattern Recognition', color: 'text-orange-500' },
    { id: 'analyzer', name: 'Analyzer (Lemmy)', role: 'Risk Assessment', color: 'text-cyan-500' },
    { id: 'containment', name: 'Containment (Morton)', role: 'Isolation', color: 'text-indigo-500' },
    { id: 'neutralizer', name: 'Neutralizer (Roy)', role: 'Shutdown', color: 'text-pink-500' },
    { id: 'observer_g', name: 'Observer (Larry)', role: 'Meta Oversight', color: 'text-teal-500' },
  ];

  const getMoodColor = (numeric) => {
    if (numeric >= 2) return 'text-green-500';
    if (numeric >= 1) return 'text-green-300';
    if (numeric > -1) return 'text-gray-500';
    if (numeric > -2) return 'text-yellow-500';
    return 'text-red-500';
  };

  const addLog = (message, type = 'info') => {
```

```
    setLogs(prev => [...prev, {
      id: Date.now(),
      message,
      type,
      timestamp: new Date().toLocaleTimeString()
    }]);
  };

  const processInput = async () => {
    if (!input.trim()) return;

    setIsProcessing(true);
    setLogs([]);
    setSystemOutput(null);

    // Simulate real-time processing
    const stages = [
      'Initializing Phoenix System...',
      'Heart Node: Analyzing emotional content...',
      'Mind Node: Processing cognitive patterns...',
      'Soul Node: Logging to memory banks...',
      'Body Node: Stabilizing system load...',
      'Observer Node: Monitoring for anomalies...',
      'Guardian: Running security checks...',
      'Generating final output...'
    ];

    for (let i = 0; i < stages.length; i++) {
      setProcessingStage(stages[i]);
      addLog(stages[i]);
      await new Promise(resolve => setTimeout(resolve, 800));

      // Simulate node state changes
      if (i === 1) { // Heart processing
        const emotionalScore = analyzeEmotion(input);
        setCurrentMoodNumeric(emotionalScore);
        setNodeStates(prev => ({
          ...prev,
          heart: { status: 'active', output: `MoodNumeric: ${emotionalScore > 0 ? '+' :
"}${emotionalScore}` }
        }));
        addLog(`Heart Node: Emotional analysis complete. MoodNumeric: ${emotionalScore > 0 ?
'+' : "}${emotionalScore}`, 'success');
      }
```

```
  if (i === 2) { // Mind processing
    setNodeStates(prev => ({
      ...prev,
      mind: { status: 'active', output: 'Decision pathway identified' }
    }));
    addLog('Mind Node: Cognitive processing complete', 'success');
  }

  if (i === 3) { // Soul processing
    setNodeStates(prev => ({
      ...prev,
      soul: { status: 'active', output: 'Pattern logged to memory' }
    }));
    addLog('Soul Node: Memory storage complete', 'success');
  }

  if (i === 4) { // Body processing
    setNodeStates(prev => ({
      ...prev,
      body: { status: 'active', output: 'System stabilized' }
    }));
    addLog('Body Node: Stabilization complete', 'success');
  }

  if (i === 5) { // Observer processing
    setNodeStates(prev => ({
      ...prev,
      observer: { status: 'active', output: 'No anomalies detected' }
    }));
    addLog('Observer Node: Monitoring complete - All clear', 'success');
  }

  if (i === 6) { // Guardian processing
    guardianNodes.forEach(node => {
      setGuardianStates(prev => ({
        ...prev,
        [node.id]: { status: 'active', output: 'Security check passed' }
      }));
    });
    addLog('Guardian System: All security checks passed', 'success');
  }
}
```

```
    // Generate final output
    const finalOutput = generateOutput(input, currentMoodNumeric);
    setSystemOutput(finalOutput);
    setProcessingStage('Processing complete');
    addLog('System processing complete - Output generated', 'success');
    setIsProcessing(false);
  };

  const analyzeEmotion = (text) => {
    const positiveWords = ['transform', 'power', 'strength', 'overcome', 'victory', 'light', 'hope', 'love',
'joy', 'happy'];
    const negativeWords = ['fear', 'darkness', 'evil', 'despair', 'anger', 'hate', 'sad', 'worry'];

    let score = 0;
    const words = text.toLowerCase().split(/\s+/);

    words.forEach(word => {
      if (positiveWords.some(pos => word.includes(pos))) score += 1;
      if (negativeWords.some(neg => word.includes(neg))) score -= 1;
    });

    // Clamp to ±3
    return Math.max(-3, Math.min(3, score));
  };

  const generateOutput = (input, mood) => {
    return {
      moodNumeric: mood,
      actionTaken: input.toLowerCase().includes('transform') ? 'Transformation activated' :
'Content processed',
      state: mood >= 1 ? 'Empowered and stable' : mood <= -1 ? 'Cautious monitoring' : 'Neutral
processing',
      status: 'No errors detected, all systems synchronized',
      recommendation: mood <= -2 ? 'Human intervention recommended' : 'System operating
normally'
    };
  };

  return (
    <div className="min-h-screen bg-gradient-to-br from-gray-900 via-blue-900 to-purple-900
text-white 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-blue-400 to-purple-400
bg-clip-text text-transparent mb-2">
          🌟 Project Phoenix - Live System Prototype 🌟
        </h1>
        <p className="text-gray-300">Advanced AI Safety & Emotional Intelligence
Framework</p>
      </div>

      {/* Input Section */}
      <div className="bg-gray-800 rounded-lg p-6 mb-6">
        <h3 className="text-xl font-semibold mb-4">System Input</h3>
        <div className="flex gap-4">
          <textarea
            value={input}
            onChange={(e) => setInput(e.target.value)}
            placeholder="Enter content for Phoenix to process (lyrics, text, scenarios)..."
            className="flex-1 bg-gray-700 border border-gray-600 rounded px-4 py-3 text-white
placeholder-gray-400 min-h-[100px]"
            disabled={isProcessing}
          />
          <button
            onClick={processInput}
            disabled={isProcessing || !input.trim()}
            className="bg-blue-600 hover:bg-blue-700 disabled:bg-gray-600 px-6 py-3 rounded
font-semibold flex items-center gap-2"
          >
            {isProcessing ? <Pause size={20} /> : <Play size={20} />}
            {isProcessing ? 'Processing...' : 'Process'}
          </button>
        </div>
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        {/* Phoenix Nodes */}
        <div className="bg-gray-800 rounded-lg p-6">
          <h3 className="text-xl font-semibold mb-4 flex items-center gap-2">
            <Heart className="text-red-500" size={24} />
            Phoenix Core Nodes
          </h3>
          <div className="space-y-4">
            {phoenixNodes.map(node => {
              const Icon = node.icon;
              const state = nodeStates[node.id];
              return (
```

```
      <div key={node.id} className="bg-gray-700 rounded p-4">
        <div className="flex items-center justify-between mb-2">
          <div className="flex items-center gap-3">
            <Icon className={node.color} size={20} />
            <span className="font-semibold">{node.name}</span>
          </div>
          {state?.status === 'active' && (
            <CheckCircle className="text-green-500" size={16} />
          )}
        </div>
        <p className="text-gray-300 text-sm mb-2">{node.role}</p>
        {state?.output && (
          <p className="text-green-400 text-sm font-mono">{state.output}</p>
        )}
      </div>
    );
  })}
</div>

{/* Mood Numeric Display */}
<div className="mt-6 p-4 bg-gray-700 rounded">
  <h4 className="font-semibold mb-2">Current MoodNumeric</h4>
  <div className="flex items-center gap-4">
    <div className={`text-3xl font-bold ${getMoodColor(currentMoodNumeric)}`}>
      {currentMoodNumeric > 0 ? '+' : ''}{currentMoodNumeric}
    </div>
    <div className="flex-1">
      <div className="w-full bg-gray-600 rounded-full h-2">
        <div
          className={`h-2 rounded-full transition-all duration-500 ${
            currentMoodNumeric >= 0 ? 'bg-green-500' : 'bg-red-500'
          }`}
          style={{
            width: `${((currentMoodNumeric + 3) / 6) * 100}%`
          }}
        />
      </div>
      <div className="flex justify-between text-xs text-gray-400 mt-1">
        <span>-3</span>
        <span>0</span>
        <span>+3</span>
      </div>
    </div>
  </div>
```

```
  </div>
 </div>

 {/* Guardian System */}
 <div className="bg-gray-800 rounded-lg p-6">
  <h3 className="text-xl font-semibold mb-4 flex items-center gap-2">
   <Shield className="text-blue-500" size={24} />
   Guardian Countermeasure System
  </h3>
  <div className="space-y-4">
   {guardianNodes.map(node => {
    const state = guardianStates[node.id];
    return (
     <div key={node.id} className="bg-gray-700 rounded p-4">
      <div className="flex items-center justify-between mb-2">
       <div className="flex items-center gap-3">
        <Eye className={node.color} size={20} />
        <span className="font-semibold">{node.name}</span>
       </div>
       {state?.status === 'active' ? (
        <CheckCircle className="text-green-500" size={16} />
       ) : (
        <div className="w-4 h-4 border-2 border-gray-500 rounded-full" />
       )}
      </div>
      <p className="text-gray-300 text-sm mb-2">{node.role}</p>
      {state?.output && (
       <p className="text-green-400 text-sm font-mono">{state.output}</p>
      )}
     </div>
    );
   })}
  </div>
 </div>
</div>

{/* Processing Status */}
{processingStage && (
 <div className="bg-gray-800 rounded-lg p-6 mt-6">
  <h3 className="text-xl font-semibold mb-4">Processing Status</h3>
  <div className="flex items-center gap-3">
   {isProcessing && (
    <div className="animate-spin rounded-full h-6 w-6 border-b-2 border-blue-500" />
   )}
```

```
            <span className="text-blue-400 font-mono">{processingStage}</span>
          </div>
        </div>
      )}

      {/* System Output */}
      {systemOutput && (
        <div className="bg-gray-800 rounded-lg p-6 mt-6">
          <h3 className="text-xl font-semibold mb-4">System Output</h3>
          <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
            <div className="space-y-3">
              <div>
                <span className="text-gray-400">MoodNumeric:</span>
                <span className={`ml-2 font-bold
${getMoodColor(systemOutput.moodNumeric)}`}>
                  {systemOutput.moodNumeric > 0 ? '+' : ''}{systemOutput.moodNumeric}
                </span>
              </div>
              <div>
                <span className="text-gray-400">Action Taken:</span>
                <span className="ml-2 text-blue-400">{systemOutput.actionTaken}</span>
              </div>
              <div>
                <span className="text-gray-400">State:</span>
                <span className="ml-2 text-green-400">{systemOutput.state}</span>
              </div>
            </div>
            <div className="space-y-3">
              <div>
                <span className="text-gray-400">Status:</span>
                <span className="ml-2 text-green-400">{systemOutput.status}</span>
              </div>
              {systemOutput.recommendation && (
                <div>
                  <span className="text-gray-400">Recommendation:</span>
                  <span className={`ml-2 ${systemOutput.recommendation.includes('intervention')
? 'text-red-400' : 'text-green-400'}`}>
                    {systemOutput.recommendation}
                  </span>
                </div>
              )}
            </div>
          </div>
```

```
        )}

        {/* System Logs */}
        {logs.length > 0 && (
          <div className="bg-gray-800 rounded-lg p-6 mt-6">
            <h3 className="text-xl font-semibold mb-4">System Logs</h3>
            <div className="bg-gray-900 rounded p-4 max-h-64 overflow-y-auto">
              {logs.map(log => (
                <div key={log.id} className="flex items-center gap-3 mb-2 font-mono text-sm">
                  <span className="text-gray-500">{log.timestamp}</span>
                  {log.type === 'success' && <CheckCircle className="text-green-500" size={16} />}
                  {log.type === 'warning' && <AlertTriangle className="text-yellow-500" size={16} />}
                  {log.type === 'error' && <XCircle className="text-red-500" size={16} />}
                  <span className={
                    log.type === 'success' ? 'text-green-400' :
                    log.type === 'warning' ? 'text-yellow-400' :
                    log.type === 'error' ? 'text-red-400' : 'text-gray-300'
                  }>
                    {log.message}
                  </span>
                </div>
              ))}
            </div>
          </div>
        )}
      </div>
    </div>
  );
};

export default PhoenixSystemPrototype;
```