

Cathedral Final Components and Registries

Complete Registry Files

packages/data/labs/racks.json (complete)

json

```
{
  "merge_strategy": "append",
  "racks": [
    {
      "id": "rack-arp-2500",
      "name": "ARP 2500 Lab",
      "map": ["osc", "vcf", "vca", "seq"],
      "scale": "Pythagorean",
      "fib": true,
      "tones": ["55", "110", "220"],
      "docs": ["/docs/synthesis_pd.pdf"],
      "lock": true
    },
    {
      "id": "rack-modular-moog",
      "name": "Moog Modular Lab",
      "map": ["osc", "ladder", "env", "noise"],
      "scale": "Platonic",
      "tones": ["65.4", "130.8", "261.6"],
      "docs": ["/docs/moog_pd.pdf"],
      "lock": true
    },
    {
      "id": "rack-emulator-ii",
      "name": "Emulator II Lab",
      "map": ["sample", "filter", "chorus"],
      "scale": "Fibonacci",
      "tones": ["100", "200"],
      "docs": ["/docs/sampler_pd.pdf"],
      "lock": true
    }
  ]
}
```

packages/data/labs/labs.json

```
json

{
  "merge_strategy": "append",
  "labs": [
    {
      "id": "lab-sound",
      "name": "Sound Lab",
      "racks": ["rack-arp-2500", "rack-modular-moog"],
      "docs": ["/docs/acoustics_pd.pdf"],
      "tags": ["harmonics", "healing"]
    },
    {
      "id": "lab-fractal",
      "name": "Fractal Lab",
      "engines": ["fractal-engine"],
      "docs": ["/docs/mandelbrot_pd.html"],
      "tags": ["geometry", "ifs"]
    },
    {
      "id": "lab-reiki",
      "name": "Reiki Grid Lab",
      "grid": ["R1", "R2", "R3", "R4", "R5", "R6", "R7", "Octarine"],
      "docs": ["/docs/reiki_pd.pdf"],
      "tags": ["energy"]
    }
  ]
}
```

packages/data/cosmos/tracks.json

```
json
```

```
{
  "merge_strategy": "append",
  "tracks": [
    {
      "id": "seed-fool",
      "steps": ["seed", "spiral", "reflect", "fuse", "publish"],
      "arcana": ["fool"],
      "rooms": ["nave"],
      "nd_safe": true
    },
    {
      "id": "spiral-hierophant",
      "steps": ["seed", "spiral", "reflect"],
      "arcana": ["hierophant"],
      "rooms": ["garden"],
      "nd_safe": true
    },
    {
      "id": "crypt-death",
      "steps": ["seed", "spiral", "reflect", "fuse"],
      "arcana": ["death"],
      "rooms": ["crypt"],
      "nd_safe": true
    },
    {
      "id": "pillar-strength",
      "steps": ["spiral", "reflect"],
      "arcana": ["strength"],
      "rooms": ["apprentice-pillar"],
      "nd_safe": true
    },
    {
      "id": "tower-shadow",
      "steps": ["reflect", "fuse", "publish"],
```

```
"arcana": ["tower"],  
"rooms": ["tower"],  
"nd_safe": true  
}  
]  
}
```

Additional Engine Components

packages/engines/fractal-engine.ts

typescript

```
export interface FractalConfig {  
  canvas: HTMLCanvasElement  
  type: 'mandelbrot' | 'julia' | 'sierpinski' | 'fibonacci' | 'platonic'  
  seed?: number  
  respectMotion?: boolean  
}
```

```
export class FractalEngine {  
  private canvas: HTMLCanvasElement  
  private ctx: CanvasRenderingContext2D  
  private type: string  
  private seed: number  
  private respectMotion: boolean  
  
  constructor(config: FractalConfig) {  
    this.canvas = config.canvas  
    this.ctx = this.canvas.getContext('2d')!  
    this.type = config.type  
    this.seed = config.seed || Date.now()  
    this.respectMotion = config.respectMotion ?? true  
  
    this.setupCanvas()  
  }  
  
  private setupCanvas() {  
    const dpr = window.devicePixelRatio || 1  
    const rect = this.canvas.getBoundingClientRect()  
  
    this.canvas.width = rect.width * dpr  
    this.canvas.height = rect.height * dpr  
  
    this.ctx.scale(dpr, dpr)  
  }  
}
```

```
render() {  
  const prefersReducedMotion = window.matchMedia('(prefers-reduced-motion: reduce)').m  
  
  if (prefersReducedMotion && this.respectMotion) {  
    this.renderStatic()  
  } else {  
    this.renderAnimated()  
  }  
}  
  
private renderStatic() {  
  switch (this.type) {  
    case 'mandelbrot':  
      this.renderMandelbrot(100) // Lower iterations for static  
      break  
    case 'fibonacci':  
      this.renderFibonacciSpiral()  
      break  
    case 'platonic':  
      this.renderPlatonicSolid()  
      break  
    default:  
      this.renderMandelbrot(100)  
  }  
}  
  
private renderAnimated() {  
  this.renderStatic() // Start with static then add gentle animation  
  // Animation would be implemented here with requestAnimationFrame  
}  
  
private renderMandelbrot(maxIterations: number) {  
  const width = this.canvas.width  
  const height = this.canvas.height  
  const imageData = this.ctx.createImageData(width, height)
```

```

for (let x = 0; x < width; x++) {
  for (let y = 0; y < height; y++) {
    const zx = (x - width / 2) / (width / 4)
    const zy = (y - height / 2) / (height / 4)

    let cx = zx
    let cy = zy
    let i = 0

    while (cx * cx + cy * cy < 4 && i < maxIterations) {
      const tmp = cx * cx - cy * cy + zx
      cy = 2 * cx * cy + zy
      cx = tmp
      i++
    }

    const pixelIndex = (y * width + x) * 4
    const value = (i / maxIterations) * 255

    // Purple/violet gradient
    imageData.data[pixelIndex] = value * 0.6 // R
    imageData.data[pixelIndex + 1] = value * 0.2 // G
    imageData.data[pixelIndex + 2] = value * 0.9 // B
    imageData.data[pixelIndex + 3] = 255 // A
  }
}

this.ctx.putImageData(imageData, 0, 0)
}

private renderFibonacciSpiral() {
  const centerX = this.canvas.width / 2
  const centerY = this.canvas.height / 2
  const scale = 5

```



```
this.ctx.strokeStyle = 'rgba(147, 51, 234, 0.6)'
```

```
this.ctx.lineWidth = 2
```

```
this.ctx.beginPath()
```

```
let theta = 0
```

```
let r = 0
```

```
for (let i = 0; i < 500; i++) {
```

```
  theta += 0.1
```

```
  r = scale * Math.pow(1.618, theta / (2 * Math.PI))
```

```
  const x = centerX + r * Math.cos(theta)
```

```
  const y = centerY + r * Math.sin(theta)
```

```
  if (i === 0) {
```

```
    this.ctx.moveTo(x, y)
```

```
  } else {
```

```
    this.ctx.lineTo(x, y)
```

```
  }
```

```
}
```

```
this.ctx.stroke()
```

```
}
```

```
private renderPlatonicSolid() {
```

```
  // Render dodecahedron projection
```

```
  const centerX = this.canvas.width / 2
```

```
  const centerY = this.canvas.height / 2
```

```
  const radius = Math.min(this.canvas.width, this.canvas.height) / 3
```

```
this.ctx.strokeStyle = 'rgba(125, 211, 252, 0.6)'
```

```
this.ctx.lineWidth = 1
```

```
// Pentagon vertices
```

```
for (let i = 0; i < 5; i++) {  
  const angle = (i * 2 * Math.PI) / 5 - Math.PI / 2  
  const x = centerX + radius * Math.cos(angle)  
  const y = centerY + radius * Math.sin(angle)  
  
  this.ctx.beginPath()  
  this.ctx.arc(x, y, 3, 0, 2 * Math.PI)  
  this.ctx.fill()  
  
  // Connect vertices  
  for (let j = i + 1; j < 5; j++) {  
    const angle2 = (j * 2 * Math.PI) / 5 - Math.PI / 2  
    const x2 = centerX + radius * Math.cos(angle2)  
    const y2 = centerY + radius * Math.sin(angle2)  
  
    this.ctx.beginPath()  
    this.ctx.moveTo(x, y)  
    this.ctx.lineTo(x2, y2)  
    this.ctx.stroke()  
  }  
}  
}  
}
```

packages/engines/tarot-engine.ts

typescript

```
import majorsData from '@cathedral/data/arcana/majors.json'

export interface SpreadConfig {
  type: 'spine-33' | 'double-tree' | 'unity-hexagram' | 'mirror-5' | 'crystal-grid'
  cards?: string[]
  seed?: number
}

export class TarotEngine {
  private majors: any[]
  private seed: number

  constructor(seed?: number) {
    this.majors = majorsData.majors
    this.seed = seed || Date.now()
  }

  drawCards(count: number): any[] {
    const shuffled = this.shuffle([...this.majors])
    return shuffled.slice(0, count)
  }

  generateSpread(config: SpreadConfig) {
    const cardCounts = {
      'spine-33': 33,
      'double-tree': 32,
      'unity-hexagram': 6,
      'mirror-5': 5,
      'crystal-grid': 8
    }

    const count = cardCounts[config.type]
    const cards = config.cards || this.drawCards(count)
  }
}
```

```
return this.layoutSpread(config.type, cards)
}

private layoutSpread(type: string, cards: any[]) {
  switch (type) {
    case 'spine-33':
      return this.layoutSpine33(cards)
    case 'double-tree':
      return this.layoutDoubleTree(cards)
    case 'unity-hexagram':
      return this.layoutUnityHexagram(cards)
    case 'mirror-5':
      return this.layoutMirror5(cards)
    case 'crystal-grid':
      return this.layoutCrystalGrid(cards)
    default:
      return { type, cards }
  }
}

private layoutSpine33(cards: any[]) {
  const chakras = [
    'Root', 'Sacral', 'Solar Plexus', 'Heart',
    'Throat', 'Third Eye', 'Crown'
  ]

  return {
    type: 'spine-33',
    positions: cards.map((card, i) => ({
      card,
      position: i + 1,
      chakra: chakras[Math.floor(i / 5)] || 'Beyond',
      aspect: this.getSpineAspect(i)
    })))
  }
}
```

```
}
```

```
private layoutDoubleTree(cards: any[]) {
```

```
  return {
```

```
    type: 'double-tree',
```

```
    day: {
```

```
      sephiroth: cards.slice(0, 10),
```

```
      paths: cards.slice(10, 21)
```

```
    },
```

```
    night: {
```

```
      qliphoth: cards.slice(21, 31),
```

```
      tunnels: cards.slice(31, 32)
```

```
    }
```

```
  }
```

```
}
```

```
private layoutUnityHexagram(cards: any[]) {
```

```
  const positions = [
```

```
    'Above', 'Below', 'East', 'West', 'North', 'South'
```

```
  ]
```

```
  return {
```

```
    type: 'unity-hexagram',
```

```
    cards: cards.map((card, i) => ({
```

```
      card,
```

```
      direction: positions[i]
```

```
    })))
```

```
  }
```

```
}
```

```
private layoutMirror5(cards: any[]) {
```

```
  return {
```

```
    type: 'mirror-5',
```

```
    self: cards[0],
```

```
    shadow: cards[1],
```

```

    anima: cards[2],
    animus: cards[3],
    synthesis: cards[4]
  }
}

private layoutCrystalGrid(cards: any[]) {
  return {
    type: 'crystal-grid',
    center: cards[0],
    vertices: cards.slice(1, 7),
    amplifier: cards[7]
  }
}

private getSpineAspect(index: number): string {
  const aspects = [
    'Foundation', 'Flow', 'Will', 'Love', 'Expression',
    'Vision', 'Connection', 'Shadow', 'Light'
  ]
  return aspects[index % aspects.length]
}

private shuffle(array: any[]): any[] {
  let m = array.length
  let t, i

  // Use seeded random
  const random = () => {
    this.seed = (this.seed * 9301 + 49297) % 233280
    return this.seed / 233280
  }

  while (m) {
    i = Math.floor(random() * m--)

```

```
t = array[m]
array[m] = array[i]
array[i] = t
}

return array
}

exportSpread(spread: any): string {
  return JSON.stringify(spread, null, 2)
}
}
```

Additional UI Components

packages/ui/TarotBar.tsx

tsx

```
import React, { useState, useEffect } from 'react'
import { bridge } from '@cathedral/engines/tesseract'

interface Guardian {
  id: string
  name: string
  hz: number
  color: string
}

const guardians: Guardian[] = [
  { id: 'fool', name: 'The Fool', hz: 396, color: '#9333ea' },
  { id: 'magus', name: 'The Magus', hz: 417, color: '#3b82f6' },
  { id: 'priestess', name: 'High Priestess', hz: 528, color: '#7dd3fc' },
  { id: 'hierophant', name: 'Hierophant', hz: 639, color: '#10b981' },
  { id: 'strength', name: 'Strength', hz: 741, color: '#f59e0b' },
  { id: 'hermit', name: 'Hermit', hz: 852, color: '#6366f1' },
  { id: 'justice', name: 'Justice', hz: 432, color: '#ec4899' },
  { id: 'death', name: 'Death', hz: 963, color: '#8b5cf6' }
]

export const TarotBar: React.FC = () => {
  const [selectedGuardian, setSelectedGuardian] = useState<Guardian | null>(null)

  const handleGuardianSelect = (guardian: Guardian) => {
    setSelectedGuardian(guardian)
    bridge.arcanaSelect(guardian.id)
    bridge.toneChange(guardian.hz)
  }

  return (
    <div style={{
      position: 'fixed',
      bottom: '60px',

```



```

left: '50%',
transform: 'translateX(-50%)',
display: 'flex',
gap: '12px',
padding: '16px',
background: 'rgba(11, 11, 15, 0.95)',
border: '1px solid rgba(147, 51, 234, 0.3)',
borderRadius: '12px',
backdropFilter: 'blur(10px)',
zIndex: 100
}}>
{guardians.map(guardian => (
  <button
    key={guardian.id}
    onClick={() => handleGuardianSelect(guardian)}
    style={{
      width: '60px',
      height: '80px',
      background: selectedGuardian?.id === guardian.id
        ? `linear-gradient(135deg, ${guardian.color}44, ${guardian.color}22)`
        : 'rgba(147, 51, 234, 0.1)',
      border: `1px solid ${guardian.color}66`,
      borderRadius: '8px',
      color: guardian.color,
      cursor: 'pointer',
      display: 'flex',
      flexDirection: 'column',
      alignItems: 'center',
      justifyContent: 'center',
      transition: 'all 0.3s ease',
      fontSize: '10px'
    }}
    title={` ${guardian.name} - ${guardian.hz} Hz`}
  >
    <div style={{

```

```
    fontSize: '24px',  
    marginBottom: '4px'  
  }}>  
    ☺  
  </div>  
  <div>{guardian.hz}</div>  
</button>  
  )}  
</div>  
)  
}
```

packages/ui/Rack.tsx

tsx

```
import React, { useState } from 'react'
```

```
interface Control {
```

```
  id: string
```

```
  type: 'knob' | 'slider' | 'toggle' | 'selector'
```

```
  label: string
```

```
  min?: number
```

```
  max?: number
```

```
  step?: number
```

```
  options?: string[]
```

```
  default: any
```

```
  unit?: string
```

```
}
```

```
interface RackProps {
```

```
  id: string
```

```
  name: string
```

```
  controls: Control[]
```

```
  onChange?: (id: string, values: Record<string, any>) => void
```

```
}
```

```
export const Rack: React.FC<RackProps> = ({ id, name, controls, onChange }) => {
```

```
  const [values, setValues] = useState<Record<string, any>>(<
```

```
    controls.reduce((acc, control) => ({
```

```
      ...acc,
```

```
      [control.id]: control.default
```

```
    })), {})
```

```
)
```

```
const handleChange = (controlId: string, value: any) => {
```

```
  const newValues = { ...values, [controlId]: value }
```

```
  setValues(newValues)
```

```
  onChange?.(id, newValues)
```

```
}
```

```

const renderControl = (control: Control) => {
  switch (control.type) {
    case 'knob':
    case 'slider':
      return (
        <div key={control.id} style={{ marginBottom: '16px' }}>
          <label style={{ display: 'block', marginBottom: '4px', fontSize: '12px', color: '#9ca3af' }}>
            {control.label}
            {control.unit} && <span> {control.unit}</span>
          </label>
          <input
            type="range"
            min={control.min}
            max={control.max}
            step={control.step || 'any'}
            value={values[control.id]}
            onChange={e => handleChange(control.id, parseFloat(e.target.value))}
            style={{
              width: '100%',
              height: '4px',
              background: 'rgba(147, 51, 234, 0.2)',
              outline: 'none',
              borderRadius: '2px'
            }}
          />
          <div style={{ textAlign: 'center', marginTop: '4px', fontSize: '14px' }}>
            {values[control.id]}
          </div>
        </div>
      )

    case 'toggle':
      return (
        <div key={control.id} style={{ marginBottom: '16px' }}>

```

```

<label style={{ display: 'flex', alignItems: 'center', cursor: 'pointer' }}>
  <input
    type="checkbox"
    checked={values[control.id]}
    onChange={e => handleChange(control.id, e.target.checked)}
    style={{ marginRight: '8px' }}
  />
  <span style={{ fontSize: '14px' }}>{control.label}</span>
</label>
</div>
)

```

case 'selector':

```

return (
  <div key={control.id} style={{ marginBottom: '16px' }}>
    <label style={{ display: 'block', marginBottom: '4px', fontSize: '12px', color: '#9ca3af' }}>
      {control.label}
    </label>
    <select
      value={values[control.id]}
      onChange={e => handleChange(control.id, e.target.value)}
      style={{
        width: '100%',
        padding: '8px',
        background: 'rgba(147, 51, 234, 0.1)',
        border: '1px solid rgba(147, 51, 234, 0.4)',
        borderRadius: '8px',
        color: '#fff',
        cursor: 'pointer',
        fontSize: '16px'
      }}
    >
      Generate Spread
    </button>
  </div>
)

```

```
{currentSpread && (  
  <div style={{  
    padding: '30px',  
    background: 'rgba(20, 20, 30, 0.95)',  
    border: '1px solid rgba(147, 51, 234, 0.3)',  
    borderRadius: '12px'  
  }}>  
    <h3>Current Spread: {spreadType}</h3>  
    <pre style={{ whiteSpace: 'pre-wrap', fontSize: '14px' }}>  
      {JSON.stringify(currentSpread, null, 2)}  
    </pre>  
  </div>  
)}  
</div>
```

```
{selectedCard && (  
  <div style={{  
    position: 'fixed',  
    right: '20px',  
    top: '100px',  
    width: '300px',  
    padding: '20px',  
    background: 'rgba(20, 20, 30, 0.95)',  
    border: '1px solid rgba(147, 51, 234, 0.3)',  
    borderRadius: '12px',  
    maxHeight: '70vh',  
    overflowY: 'auto'  
  }}>  
    <h3>{selectedCard.name}</h3>  
    <p><strong>Guardian:</strong> {selectedCard.guardian}</p>  
    <p><strong>Glyph:</strong> {selectedCard.glyph}</p>  
    <p><strong>Portal:</strong> {selectedCard.portal.form}</p>  
    <div>  
      <strong>Lineage:</strong>
```

```

        <ul>
          {selectedCard.lineage.map((item: string, i: number) => (
            <li key={i}>{item}</li>
          ))}
        </ul>
      </div>
    </div>
  )}
</div>
)
}, 0.3)',
    borderRadius: '4px',
    color: '#fff',
    fontSize: '14px'
  }}
>
  {control.options?.map(option => (
    <option key={option} value={option}>{option}</option>
  ))}
</select>
</div>
)

default:
  return null
}
}

return (
  <div style={{
    padding: '24px',
    background: 'linear-gradient(135deg, rgba(20,20,30,0.95), rgba(30,20,40,0.95))',
    border: '1px solid rgba(147, 51, 234, 0.3)',
    borderRadius: '12px'
  }}>

```

```
<h3 style={{ marginBottom: '24px', fontSize: '20px' }}>{name}</h3>
<div style={{
  display: 'grid',
  gridTemplateColumns: 'repeat(auto-fit, minmax(200px, 1fr))',
  gap: '20px'
}}>
  {controls.map(renderControl)}
</div>
</div>
)
}
```

packages/ui/DocViewer.tsx

tsx


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

```
interface DocViewerProps {
```

```
  docPath: string
```

```
  title?: string
```

```
}
```

```
export const DocViewer: React.FC<DocViewerProps> = ({ docPath, title }) => {
```

```
  const [content, setContent] = useState<string>('')
```

```
  const [loading, setLoading] = useState(true)
```

```
  const [error, setError] = useState<string | null>(null)
```

```
  useEffect(() => {
```

```
    fetch(docPath)
```

```
    .then(res => {
```

```
      if (!res.ok) throw new Error(`Failed to load document: ${res.status}`)
```

```
      return res.text()
```

```
    })
```

```
    .then(text => {
```

```
      setContent(text)
```

```
      setLoading(false)
```

```
    })
```

```
    .catch(err => {
```

```
      setError(err.message)
```

```
      setLoading(false)
```

```
    })
```

```
  }, [docPath])
```

```
  if (loading) {
```

```
    return (
```

```
      <div style={{ padding: '40px', textAlign: 'center' }}>
```

```
        <div>Loading document...</div>
```

```
      </div>
```

```
    )
```

```

}

if (error) {
  return (
    <div style={{ padding: '40px', textAlign: 'center', color: '#ef4444' }}>
      <div>Error: {error}</div>
    </div>
  )
}

```

```

const getProvenance = () => {
  const filename = docPath.split('/').pop()
  const sources: Record<string, string> = {
    'sefer_yetzirah_pd.pdf': 'Sefer Yetzirah - Public Domain translation (1877)',
    'corpus_hermeticum_pd.pdf': 'Corpus Hermeticum - Mead translation (1906)',
    'agrippa_occult_pd.pdf': 'Three Books of Occult Philosophy - Public Domain (1651)',
    'dee_monas.pdf': 'Monas Hieroglyphica - John Dee (1564)',
    'bruno_cosmology.pdf': 'On the Infinite Universe - Giordano Bruno (1584)',
    'hildegard_pd.pdf': 'Scivias - Hildegard of Bingen (12th century)',
    'pythagoras_pd.pdf': 'Golden Verses - Pythagorean texts (Public Domain)',
    'hypatia_pd.pdf': 'Commentary on Apollonius - fragments (Public Domain)'
  }

  return sources[filename!] || 'Public Domain source'
}

```

```

return (
  <div style={{
    maxWidth: '800px',
    margin: '0 auto',
    padding: '40px'
  }}>
    {title && (
      <h1 style={{
        fontSize: '32px',

```

```
marginBottom: '24px',
textAlign: 'center'
}}>
{title}
</h1>
)}
```

```
<div style={{
background: 'rgba(20, 20, 30, 0.5)',
border: '1px solid rgba(147, 51, 234, 0.2)',
borderRadius: '8px',
padding: '32px',
lineHeight: '1.8',
fontSize: '16px',
whiteSpace: 'pre-wrap'
}}>
{content}
</div>
```

```
<footer style={{
marginTop: '32px',
padding: '16px',
background: 'rgba(147, 51, 234, 0.1)',
border: '1px solid rgba(147, 51, 234, 0.3)',
borderRadius: '8px',
fontSize: '14px',
color: '#9ca3af'
}}>
<strong>Provenance:</strong> {getProvenance()}
<br />
<strong>Access:</strong> Open archive. Museum-grade. ND-safe.
<br />
<strong>Note:</strong> This document is part of the public domain collection maintained
</footer>
</div>
```

```
)  
}
```

Final Pages

apps/web/src/pages/Arcana.tsx

tsx

```
import React, { useState } from 'react'
import { TarotEngine } from '@cathedral/engines/tarot'
import majorsData from '@cathedral/data/arcana/majors.json'

export default function Arcana() {
  const [selectedCard, setSelectedCard] = useState<any>(null)
  const [spreadType, setSpreadType] = useState<string>('mirror-5')
  const [currentSpread, setCurrentSpread] = useState<any>(null)

  const engine = new TarotEngine()

  const generateSpread = () => {
    const spread = engine.generateSpread({ type: spreadType as any })
    setCurrentSpread(spread)
  }

  return (
    <div style={{ padding: '40px', maxWidth: '1400px', margin: '0 auto' }}>
      <h1 style={{ fontSize: '48px', marginBottom: '40px', textAlign: 'center' }}>
        Living Arcanae
      </h1>

      <div style={{ marginBottom: '40px' }}>
        <h2>Major Arcana</h2>
        <div style={{
          display: 'grid',
          gridTemplateColumns: 'repeat(auto-fill, minmax(150px, 1fr))',
          gap: '20px'
        }}>
          {majorsData.majors.map(card => (
            <div
              key={card.id}
              onClick={() => setSelectedCard(card)}
              style={{
```



```
onChange={e => setSpreadType(e.target.value)}
style={{
  padding: '8px 16px',
  background: 'rgba(147, 51, 234, 0.1)',
  border: '1px solid rgba(147, 51, 234, 0.3)',
  borderRadius: '8px',
  color: '#fff',
  fontSize: '16px'
}}
>
<option value="mirror-5">Mirror 5</option>
<option value="unity-hexagram">Unity Hexagram</option>
<option value="crystal-grid">Crystal Grid</option>
<option value="double-tree">Double Tree</option>
<option value="spine-33">Spine 33</option>
</select>
<button
  onClick={generateSpread}
  style={{
    padding: '8px 24px',
    background: 'rgba(147, 51, 234, 0.2)',
    border: '1px solid rgba(147, 51, 234
```