IRISVOICE Level 4 (Mini Node Stack) - Complete PRD

Version: 1.0

Date: February 3, 2026
Status: Ready for Implementation
Priority: HIGH

🎯 Executive Summary

Level 4 transforms the navigation into an interactive field input system. When a subnode is clicked (e.g., “Input” from Voice category), the center IrisOrb shrinks and blurs, the subnode label becomes the new orb label, and a stack of mini nodes (field wrappers) appears connected by a glass morphism line. Users navigate through the stack carousel-style, confirming inputs which then orbit around the shrinking orb.

📊 Visual Flow

```
LEVEL 3 (Subnodes visible):
           [VOICE]
              ↓
    [Input] ← User clicks this
  [Output] [Processing]
       [Model]

             ↓ TRANSITION ↓

LEVEL 4 (Mini Node Stack):

   Mini Stack (4 visible)          IrisOrb (shrunk to 60px)
   ┌──────────────────┐                   ┌─────┐
   │ #4 Input Test    │ 0.6 opacity        │  I  │ ← "INPUT" label
   ├──────────────────┤                    │  N  │ ← Blur effect
   │ #3 VAD           │ 0.6 opacity        │  P  │
   ├──────────────────┤                    │  U  │
   │ #2 Noise Gate    │ 0.6 opacity        │  T  │
   ├──────────────────┤═════════════════════┘     │
   │ #1 Sensitivity  │ ← ACTIVE (1.0)            │
   │  [Slider 50%]   │   180px × 180px   Glass line
   │  [Save] button  │                   1.5px thick
   └─────────────────┘

User fills field → Clicks Save → Node flies to orbit

AFTER CONFIRMATION:
                              ┌───┐
                         ┌──┬─┤ I ├─┬──┐
                         │#1│ │ N │ │  │ ← Orb grows
                         └──┴─┤ P ├─┴──┘   with each
   Next Active:               │ U │        confirmed
   ┌─────────────┐            │ T │        node
   │ #3 VAD      │            └───┘
   ├─────────────┤              │
   │ #2 Noise Gt │              │
   ├─────────────┤══════════════╛
   │   ACTIVE    │
   └─────────────┘
```

🎨 Detailed Specifications

1. Center IrisOrb Transformation

1.1 Size Animation

Initial State (L3):

- Size: 120px (current default)
- Label: “IRIS” or parent node label
- Opacity: 1.0
- Blur: none

Transition to L4:

```
const orbTransition = {
  size: {
    from: 120,
    to: 60,
    duration: 800,
    ease: [0.4, 0, 0.2, 1]
  },
  blur: {
    from: 0,
    to: 8, // 8px backdrop blur on scene behind
    duration: 600,
    ease: 'easeOut'
  },
  label: {
    fadeOutDuration: 300,
    fadeInDuration: 300,
    gap: 100 // ms between fade out and fade in
  }
}
```

Shrink Behavior:

- Shrinks in place at center (50%, 50%)
- Scales down to 60px
- As it shrinks, backdrop blur on entire scene increases to 8px
- Creates “depth of field” effect - orb in focus, background soft

Label Transition:

```
1. Fade out "IRIS" (300ms)
2. Wait 100ms
3. Fade in "INPUT" (300ms)
4. Total: 700ms
```

Dynamic Orb Growth (Important!):

- Each confirmed mini node → Orb grows by 10px
- Formula: `orbSize = 60 + (confirmedCount * 10)`
- Example:
    - 0 confirmed: 60px
    - 1 confirmed: 70px
    - 2 confirmed: 80px
    - 3 confirmed: 90px

    - 4 confirmed: 100px (if all 4 in stack)
- When node returns to stack → Orb shrinks by 10px

---

2. Glass Morphism Connecting Line

2.1 Visual Specification

Already Implemented: `edge-to-edge-line.tsx` provides the foundation

Modifications Needed:

```
const L4_LINE_CONFIG = {
  thickness: 1.5, // Updated from 1.5 (already correct!)
  glowThickness: 4,
  animationDuration: 800, // Sync with orb shrink
  gradient: true, // Animated gradient along line
  turbulence: true, // Liquid metal effect
  glassEffect: {
    backdropFilter: 'blur(8px)',
    opacity: 0.6
  }
}
```

Connection Points:

```
interface LineConnection {
  start: {
    // IrisOrb border edge
    x: orbX + (Math.cos(angleToStack) * (orbSize / 2)),
    y: orbY + (Math.sin(angleToStack) * (orbSize / 2))
  },
  end: {
    // Active mini node border edge
    x: stackX - (Math.cos(angleToStack) * (miniNodeSize / 2)),
    y: stackY - (Math.sin(angleToStack) * (miniNodeSize / 2))
  }
}
```

Multiple Lines (Key Feature!):

- **ONE line to active card in stack** (always visible)
- **ADDITIONAL lines to confirmed nodes in orbit** (one per confirmed node)

Line States:

```
type LineState =
  | 'drawing'      // Entering L4 - line draws from orb to stack
  | 'active'       // Connected to active mini node
  | 'confirmed'    // Connected to confirmed orbit node (dimmer)
  | 'retracting'   // Exiting L4 - line retracts back to orb
```

Visual Hierarchy:

```
const lineOpacity = {
  active: 1.0,        // Bright, pulsing
  confirmed: 0.4,     // Dim, steady
  retracting: 0.0     // Fading out
}

const lineAnimation = {
  active: {
    // Gradient flows along line
    // Subtle pulsing (scale 1.0 → 1.05 → 1.0)
  },
  confirmed: {
    // Static gradient
    // No pulsing
  }
}
```

---

3. Mini Node Stack System

3.1 Stack Positioning

Position Relative to Clicked Subnode:

```
interface StackPosition {
  // Remember where subnode was clicked
  subnodeAngle: number  // e.g., -90° for top, 0° for right

  // Stack appears at same angle but farther out
  distance: 320,  // Increased from 260 to accommodate 2x size

  // Calculate position
  x: Math.cos(subnodeAngle * Math.PI / 180) * distance,
  y: Math.sin(subnodeAngle * Math.PI / 180) * distance
}
```

Example Positions:

```
If user clicked "Input" at angle -90° (top):
→ Stack appears at top (0, -320)

If user clicked "Wake" at angle 0° (right):
→ Stack appears at right (320, 0)
```

Why This Matters:

- Maintains spatial relationship
- User's mental model preserved
- Feels natural and oriented

3.2 Stack Visual Structure

Stack Depth & Offset:

```
const STACK_CONFIG = {
  maxVisible: 4,           // Show max 4 cards
  baseSize: 180,           // Active card size (2x normal)
  offsetX: 12,             // Horizontal offset per card
  offsetY: 8,              // Vertical offset per card
  scaleReduction: 0.05,    // Each card slightly smaller
  zIndexBase: 100          // Z-index management
}
```

Card Positions (from front to back):

```
// Card 1 (Active - Front)
{
  x: 0,
  y: 0,
  scale: 1.0,
  opacity: 1.0,
  zIndex: 104
}

// Card 2 (Behind active)
{
  x: -12,  // Peek out to left
  y: 8,    // Peek out below
  scale: 0.95,
  opacity: 0.6,
  zIndex: 103
}

// Card 3
{
  x: -24,
  y: 16,
  scale: 0.90,
  opacity: 0.6,
  zIndex: 102
}

// Card 4
{
  x: -36,
  y: 24,
  scale: 0.85,
  opacity: 0.6,
  zIndex: 101
}

// Cards 5+ (if exist)
// Hidden, only shown when stack rotates
```

Visual Effect:

```
  ┌──────────────────┐
  │ Card 4           │  ← Peeking top-left
  ├──────────────────┤
  │ Card 3           │  ← Peeking top-left
  ├──────────────────┤
  │ Card 2           │  ← Peeking top-left
  ├──────────────────┤═════╗
  │ Card 1 (Active)  │     ║ Glass line
  │  [Field Input]   │     ║ to orb
  │  [Save Button]   │     ║
  └──────────────────┘═════╝
```

3.3 Stack Content Structure

Each Mini Node Contains:

Max 3 Fields Per Subnode (as requested):

```
interface MiniNodeContent {
  id: string
  label: string
  icon: string | ElementType
  fields: Field[]  // Max 3 fields
}

type Field =
  | TextFieldConfig
  | SliderFieldConfig
  | DropdownFieldConfig
  | ToggleFieldConfig
  | ColorFieldConfig
```

Example - “Input” Subnode (4 mini nodes):

```
const inputSubnodeMiniNodes = [
  {
    id: 'input-sensitivity',
    label: 'Input Sensitivity',
    icon: 'Mic',
    fields: [
      {
        id: 'sensitivity',
        type: 'slider',
        label: 'Sensitivity',
        min: 0,
        max: 100,
        unit: '%',
        default: 50
      }
    ]
  },
  {
    id: 'noise-gate',
    label: 'Noise Gate',
    icon: 'Mic',
    fields: [
      {
        id: 'noise_gate_enabled',
        type: 'toggle',
        label: 'Enable Noise Gate'
      },
      {
        id: 'gate_threshold',
        type: 'slider',
        label: 'Threshold',
        min: -60,
        max: 0,
        unit: 'dB'
      }
    ]
  },
  {
    id: 'vad',
    label: 'VAD',
    icon: 'Waveform',
    fields: [
      {
        id: 'vad_enabled',
        type: 'toggle',
        label: 'Voice Activity Detection'
      }
    ]
  },
  {
    id: 'input-test',
    label: 'Input Test',
    icon: 'Mic',
    fields: [
      {
        id: 'test_input',
        type: 'text',
        label: 'Test Phrase',
        placeholder: 'Say something...'
      }
    ]
  }
]
```

Field Display in Stack:

```
┌──────────────────────────┐
│ INPUT SENSITIVITY        │ ← Label + Icon
├──────────────────────────┤
│                          │
│  Sensitivity             │ ← Field label
│  [━━━━━●━━━━━━━━] 50%    │ ← Slider (visible!)
│                          │
│        [Save]            │ ← Save button
│                          │
└──────────────────────────┘
    180px × 180px
```

All content visible at 180px size (that's why you made it 2x!)

3.4 Save Button Design

Why Save Button (Not Auto-Blur):

- Prevents accidental confirms
- User has clear control
- Can review before saving
- Feels intentional

Button Specification:

```
const saveButton = {
  position: 'bottom center',
  width: '80%', // 144px at 180px card width
  height: 32,
  style: {
    background: 'rgba(255, 255, 255, 0.1)',
    backdropFilter: 'blur(12px)',
    border: '1px solid rgba(255, 255, 255, 0.2)',
    borderRadius: '8px',
    fontSize: '11px',
    fontWeight: 600,
    textTransform: 'uppercase',
    letterSpacing: '0.05em',
    color: glowColor,
    cursor: 'pointer'
  },
  hover: {
    background: 'rgba(255, 255, 255, 0.15)',
    border: `1px solid ${glowColor}40`,
    boxShadow: `0 0 12px ${glowColor}30`
  }
}
```

Button States:

```
type SaveButtonState =
  | 'idle'         // Ready to save
  | 'saving'       // Animating to orbit
  | 'disabled'     // Invalid input
```

---

4. Stack Interaction Model

4.1 Navigation Methods

Method A: Confirm Current (Primary Flow)

```
User fills field → Clicks "Save" → Stack rotates forward
```

Method B: Click Behind Cards

```
User clicks Card #3 → Stack rotates to bring #3 forward
```

Method C: Click Confirmed Orbit Node

```
User clicks orbiting node → Node flies back to stack as active card
```

Keyboard Navigation (Nice to Have):

```
Arrow Up/Down → Rotate stack
Enter → Save current
Escape → Cancel and go back to L3
```

4.2 Stack Rotation Animation (Carousel)

Rotation Direction:

```
Forward: Next card slides to front
Backward: Previous card slides to front
```

Animation Sequence (Forward):

```
const rotateStackForward = {
  // Step 1: Current active card exits
  activeCard: {
    from: { x: 0, y: 0, scale: 1.0, opacity: 1.0, zIndex: 104 },
    to: {
      x: -48,      // Slides to back-left
      y: 32,       // Slides to back-bottom
      scale: 0.80,
      opacity: 0.6,
      zIndex: 100  // Now at back
    },
    duration: 400,
    ease: [0.4, 0, 0.2, 1]
  },

  // Step 2: All other cards shift forward
  otherCards: {
    // Each card moves to position of card in front
    stagger: 50, // ms between each card
    duration: 400,
    ease: [0.4, 0, 0.2, 1]
  },

  // Step 3: Next card becomes active
  nextCard: {
    from: { x: -12, y: 8, scale: 0.95, opacity: 0.6, zIndex: 103 },
    to: { x: 0, y: 0, scale: 1.0, opacity: 1.0, zIndex: 104 },
    duration: 400,
    ease: [0.4, 0, 0.2, 1]
  }
}
```

Total Duration: 400ms (feels snappy, not sluggish)

Visual During Rotation:

- Glass line stays connected to card as it moves
- Line smoothly tracks active card position
- Blur effect on moving cards (motion blur 2px)

4.3 Clicking Behind Cards

Scenario: User clicks Card #3 (2 cards behind active)

Behavior:

```
if (clickedCardIndex > activeCardIndex) {
  // Clicked card is behind - rotate forward multiple times
  const rotations = clickedCardIndex - activeCardIndex

  // Rotate stack 'rotations' times
  for (let i = 0; i < rotations; i++) {
    await rotateStackForward()
  }
} else if (clickedCardIndex < activeCardIndex) {
  // Clicked card is ahead (shouldn't happen in normal flow)
  // Rotate backward
  const rotations = activeCardIndex - clickedCardIndex

  for (let i = 0; i < rotations; i++) {
    await rotateStackBackward()
  }
}
```

Animation:

- If 2 rotations needed → Do 2 consecutive rotations
- Each rotation takes 400ms
- Total: 800ms for 2 jumps
- Fast enough to feel responsive
- Slow enough to see carousel effect

Alternative (Instant Jump):

- Could make clicked card instantly swap to front
- Would be faster but less satisfying
- **Recommendation:** Use carousel rotation (feels better)

---

5. Confirmation & Orbit System

5.1 Save Button Click Flow

User clicks "Save" on active mini node:

```
1. Validate Input
   ↓
2. Save to State
   ↓
3. Trigger Orbit Animation
   ↓
4. Grow Orb Size (+10px)
   ↓
5. Add Confirmed Line
   ↓
6. Rotate Stack Forward
```

Detailed Sequence:

```
async function handleSaveClick(miniNodeId: string, values: Record<string, any>) {
  console.log('[L4] Save clicked:', miniNodeId, values)

  // 1. Validate
  const isValid = validateFields(values)
  if (!isValid) {
    showValidationError()
    return
  }

  // 2. Save to state
  const confirmed = {
    id: miniNodeId,
    values: values,
    timestamp: Date.now()
  }
  addConfirmedNode(confirmed)

  // 3. Calculate orbit position
  const orbitAngle = calculateNextOrbitAngle()
  const orbitRadius = 200 // Same as L5

  // 4. Trigger animations simultaneously
  Promise.all([
    animateMiniNodeToOrbit(miniNodeId, orbitAngle, orbitRadius),
    growIrisOrb(10), // +10px
    drawConfirmedLine(miniNodeId, orbitAngle),
    rotateStackForward()
  ])

  console.log('[L4] Node confirmed and in orbit')
}
```

5.2 Orbit Animation

From Stack to Orbit:

```
const stackToOrbitAnimation = {
  // Phase 1: Lift from stack (0-200ms)
  lift: {
    from: { x: stackX, y: stackY, scale: 1.0, opacity: 1.0 },
    to: { x: stackX, y: stackY - 20, scale: 0.95, opacity: 0.95 },
    duration: 200,
    ease: 'easeOut'
  },

  // Phase 2: Arc to orbit (200-800ms)
  arc: {
    // Bezier curve from stack to orbit
    path: calculateArcPath(
      { x: stackX, y: stackY - 20 },
      { x: orbitX, y: orbitY }
    ),
    duration: 600,
    ease: [0.34, 1.56, 0.64, 1] // Bouncy
  },

  // Phase 3: Scale down while moving (simultaneous with arc)
  scale: {
    from: 1.0,    // 180px
    to: 0.5,      // 90px (normal orbit size)
    duration: 600,
    ease: 'easeOut'
  },

  // Phase 4: Settle into orbit (800-1000ms)
  settle: {
    // Small bounce
    scale: [0.5, 0.55, 0.5],
    duration: 200,
    ease: 'easeInOut'
  }
}
```

Total Duration: 1000ms (1 second)

Orbit Position Calculation:

```
function calculateNextOrbitAngle(confirmedNodes: ConfirmedNode[]): number {
  // Distribute evenly around circle
  const count = confirmedNodes.length + 1 // +1 for this new node
  const angleStep = 360 / Math.max(count, 4) // Min 4 positions

  // Start at top (-90°) and go clockwise
  const baseAngle = -90
  const index = confirmedNodes.length

  return baseAngle + (index * angleStep)
}

// Examples:
// 1st confirmed: -90° (top)
// 2nd confirmed: 0° (right)
// 3rd confirmed: 90° (bottom)
// 4th confirmed: 180° (left)
```

5.3 Confirmed Node in Orbit

Appearance:

```
const confirmedOrbitNode = {
  size: 90, // Normal size (not 2x)
  position: {
    radius: 200,
    angle: calculatedAngle
  },
  style: {
    background: 'rgba(255, 255, 255, 0.08)',
    backdropFilter: 'blur(12px)',
    border: '1px solid rgba(255, 255, 255, 0.1)',
    borderRadius: '16px'
  },
  content: {
    icon: miniNode.icon, // Show icon
    label: miniNode.label, // Show label (small, 8px)
    preview: firstFieldValue // Show first field value
  },
  interactivity: {
    cursor: 'pointer',
    onClick: bringBackToStack
  }
}
```

Glass Line to Orbit Node:

```
const confirmedLine = {
  start: orbEdge,
  end: confirmedNodeEdge,
  thickness: 1.5,
  opacity: 0.4, // Dimmer than active line
  gradient: true,
  animation: 'none' // Static, no pulsing
}
```

5.4 Bringing Confirmed Node Back to Stack

User clicks orbiting confirmed node:

```
1. Remove from orbit
   ↓
2. Shrink orb (-10px)
   ↓
3. Remove confirmed line
   ↓
4. Fly back to stack
   ↓
5. Become active card
   ↓
6. Shift other cards back
```

Animation:

```
const orbitToStackAnimation = {
  // Phase 1: Lift from orbit (0-150ms)
  lift: {
    scale: [0.5, 0.55],
    duration: 150
  },

  // Phase 2: Arc back to stack (150-700ms)
  arc: {
    path: calculateArcPath(
      { x: orbitX, y: orbitY },
      { x: stackX, y: stackY }
    ),
    duration: 550,
    ease: [0.4, 0, 0.2, 1]
  },

  // Phase 3: Grow back to 2x (simultaneous with arc)
  scale: {
    from: 0.5,   // 90px
    to: 1.0,     // 180px
    duration: 550
  },

  // Phase 4: Insert into stack front
  insert: {
    // Becomes active card (front of stack)
    x: 0,
    y: 0,
    scale: 1.0,
    opacity: 1.0,
    zIndex: 104
  }
}
```

Stack Adjustment:

```
// When node returns to front:
// 1. Current active becomes 2nd position
// 2. All other cards shift back one position
// 3. Last visible card (4th) gets hidden

// Net effect: Stack shifts backward to make room
```

---

6. Entry & Exit Transitions

6.1 Level 3 → Level 4 (Entry)

Trigger: User clicks a subnode (e.g., “Input”)

Animation Sequence:

```
Timeline:
0ms ──────────────────────────── 1200ms
```

```
0-400ms:   Subnodes exit (shrink + fade)
300-800ms: IrisOrb shrinks (120px → 60px)
300-800ms: Blur increases (0px → 8px)
400-700ms: Label fades out/in
600-1200ms: Mini stack appears + line draws
```

Detailed Steps:

Step 1: Subnodes Exit (0-400ms)

```
const subnodeExit = {
  // All subnodes except clicked one
  otherSubnodes: {
    scale: [1, 0.8, 0],
    opacity: [1, 0.5, 0],
    duration: 400,
    stagger: 50,
    ease: 'easeIn'
  },

  // Clicked subnode (stays briefly)
  clickedSubnode: {
    // Pulses once before disappearing
    scale: [1, 1.1, 0],
    opacity: [1, 1, 0],
    duration: 400,
    ease: 'easeInOut'
  }
}
```

Step 2: IrisOrb Transformation (300-800ms)

```
const orbShrink = {
  size: {
    from: 120,
    to: 60,
    duration: 500,
    ease: [0.4, 0, 0.2, 1]
  },
  label: {
    fadeOut: {
      duration: 200,
      ease: 'easeOut'
    },
    wait: 100,
    fadeIn: {
      duration: 200,
      ease: 'easeIn'
    }
  }
}
```

Step 3: Blur Effect (300-800ms)

```
  const sceneBlur = {
    // Apply to container behind everything
    target: '.hexagonal-container',
    filter: {
      from: 'blur(0px)',
      to: 'blur(8px)',
      duration: 500,
      ease: 'easeOut'
    }
  }
```

Step 4: Mini Stack Appearance (600-1200ms)

```
  const stackEntrance = {
    // Calculate position based on clicked subnode angle
    position: {
      angle: clickedSubnodeAngle,
      distance: 320
    },

    // All cards appear together
    cards: {
      initial: {
        scale: 0,
        opacity: 0,
        // Start at center
        x: 0,
        y: 0
      },
      animate: {
        // Move to stack position
        // Front card first, then others staggered
        stagger: 80,
        duration: 600,
        ease: [0.34, 1.56, 0.64, 1] // Bouncy
      }
    }
  }
```

Step 5: Line Drawing (600-1200ms)

```
const lineDraw = {
  // Use existing LiquidMetalLine component
  pathLength: {
    from: 0,
    to: 1,
    duration: 600,
    ease: 'easeInOut'
  },
  opacity: {
    from: 0,
    to: 1,
    duration: 400
  }
}
```

6.2 Level 4 → Level 3 (Exit)

Trigger: User clicks IrisOrb (back button)

What Happens to Confirmed Nodes:

- **They disappear** (as you specified)
- **Values are saved** in state
- When user returns to L4, confirmed nodes don't re-appear
- Values persist in backend/context

Animation Sequence:

```
Timeline:
0ms ─────────────────────────── 1000ms

0-400ms:   Stack collapses to center
0-400ms:   Lines retract to orb
0-400ms:   Confirmed orbit nodes fade out
300-800ms: IrisOrb grows (current → 120px)
300-800ms: Blur decreases (8px → 0px)
400-700ms: Label fades out/in
600-1000ms: Subnodes reappear
```

Detailed Steps:

Step 1: Stack Collapse (0-400ms)

```
const stackExit = {
  cards: {
    // All cards collapse to center
    to: { x: 0, y: 0, scale: 0, opacity: 0 },
    duration: 400,
    stagger: 50, // Back to front
    ease: 'easeIn'
  }
}
```

Step 2: Orbit Nodes Fade (0-400ms)

```
const orbitNodesFade = {
  // All confirmed nodes in orbit
  to: { opacity: 0, scale: 0.8 },
  duration: 400,
  ease: 'easeOut'
}
```

Step 3: Lines Retract (0-400ms)

```
const linesRetract = {
  // All lines (active + confirmed)
  pathLength: {
    from: 1,
    to: 0,
    duration: 400,
    ease: 'easeIn'
  }
}
```

Step 4: IrisOrb Growth (300-800ms)

```
const orbGrow = {
  size: {
    from: currentOrbSize, // Could be 60-100px depending on confirmed nodes
    to: 120,
    duration: 500,
    ease: [0.4, 0, 0.2, 1]
  },
  label: {
    // "INPUT" → "IRIS" or parent label
    fadeOut: 200,
    wait: 100,
    fadeIn: 200
  }
}
```

Step 5: Blur Removal (300-800ms)

```
const blurRemove = {
  filter: {
    from: 'blur(8px)',
    to: 'blur(0px)',
    duration: 500,
    ease: 'easeOut'
  }
}
```

Step 6: Subnodes Return (600-1000ms)

```
const subnodesReturn = {
  // Same as L2→L3 transition
  // Use existing spiral/radial animation
  from: { scale: 0, opacity: 0 },
  to: { scale: 1, opacity: 1 },
  duration: 400,
  stagger: 100
}
```

---

7. State Management

7.1 Navigation State Extension

Add to NavigationContext:

```
interface NavigationState {
  // Existing
  level: 1 | 2 | 3 | 4 | 5
  selectedMain: string | null
  selectedSub: string | null
  selectedMini: string | null
  isTransitioning: boolean
  history: HistoryEntry[]

  // NEW for Level 4
  miniNodeStack: MiniNode[]           // All mini nodes for current subnode
  activeMiniNodeIndex: number         // Index of active card in stack
  confirmedMiniNodes: ConfirmedNode[] // Nodes in orbit
  miniNodeValues: Record<string, Record<string, any>> // Field values
}

interface MiniNode {
  id: string
  label: string
  icon: string | ElementType
  fields: FieldConfig[]
}

interface ConfirmedNode {
  id: string
  label: string
  icon: string | ElementType
  values: Record<string, any>
  orbitAngle: number
  timestamp: number
}

interface FieldConfig {
  id: string
  type: 'text' | 'slider' | 'dropdown' | 'toggle' | 'color'
  label: string
  // ... type-specific props
}
```

7.2 New Actions

```
type NavigationAction =
  | { type: 'SELECT_SUB', payload: { subnodeId: string } }
  | { type: 'ROTATE_STACK_FORWARD' }
  | { type: 'ROTATE_STACK_BACKWARD' }
  | { type: 'JUMP_TO_MINI_NODE', payload: { index: number } }
  | { type: 'CONFIRM_MINI_NODE', payload: { id: string, values: Record<string, an
  | { type: 'RECALL_CONFIRMED_NODE', payload: { id: string } }
  | { type: 'UPDATE_MINI_NODE_VALUE', payload: { nodeId: string, fieldId: string,
  | { type: 'GO_BACK' }
```

7.3 Value Persistence

Where to Store:

Option A: Context + LocalStorage (Recommended)

```
// In NavigationContext
const [miniNodeValues, setMiniNodeValues] = useState<Record<string, Record<string
  // Load from localStorage on init
  const saved = localStorage.getItem('iris-mini-node-values')
  return saved ? JSON.parse(saved) : {}
})

// Save to localStorage on update
useEffect(() => {
  localStorage.setItem('iris-mini-node-values', JSON.stringify(miniNodeValues))
}, [miniNodeValues])
```

Why LocalStorage:

- Persists across sessions
- User doesn't lose work on refresh
- Simple to implement
- Works offline

Option B: Backend API (Future)

```
// Save to backend when confirmed
async function confirmMiniNode(id: string, values: Record<string, any>) {
  await api.post('/user/settings', {
    category: selectedMain,
    subcategory: selectedSub,
    miniNode: id,
    values: values
  })
}
```

Recommendation: Start with Option A (localStorage), migrate to Option B later

---

7.4 Going Back L4 → L3

Behavior:

- Confirmed values are SAVED (persist in state + localStorage)
- Confirmed orbit nodes DISAPPEAR (visual cleanup)
- Stack and lines RETRACT
- When user returns to this subnode later, they start fresh (empty stack)
- But values are pre-filled if they were previously confirmed

Why This Design:

- Clean exit (no clutter)
- Values persist (user doesn't lose work)
- Fresh start on re-entry (clear mental model)

---

8. Data Structure & Configuration

8.1 Mini Node Definitions

Structure:

```
// Structure: SUB_NODES → fields become mini nodes
const SUB_NODES_WITH_MINI = {
  voice: [
    {
      id: 'input',
      label: 'INPUT',
      icon: Mic,
      miniNodes: [
        {
          id: 'input-device',
          label: 'Input Device',
          icon: Mic,
          fields: [
            {
              id: 'input_device',
              type: 'dropdown',
              label: 'Device',
              options: ['Default', 'USB Mic', 'Headset', 'Webcam']
            }
          ]
        },
        {
          id: 'input-sensitivity',
          label: 'Sensitivity',
          icon: Mic,
          fields: [
            {
              id: 'input_sensitivity',
              type: 'slider',
              label: 'Sensitivity',
              min: 0,
              max: 100,
              unit: '%',
              default: 50
            }
          ]
        },
        {
          id: 'noise-gate',
          label: 'Noise Gate',
          icon: Mic,
          fields: [
            {
              id: 'noise_gate',
              type: 'toggle',
              label: 'Enable'
            },
            {
              id: 'gate_threshold',
              type: 'slider',
              label: 'Threshold',
              min: -60,
              max: 0,
              unit: 'dB'
            }
          ]
        },
        {
          id: 'vad',
          label: 'VAD',
          icon: Waveform,
          fields: [
            {
              id: 'vad',
              type: 'toggle',
              label: 'Voice Activity Detection'
            }
```

```
          ]
        }
      ]
    },
    // ... other subnodes
  ]
}
```

Max 3 Fields Per Mini Node (as requested)

8.2 Field Type Mapping

You have these field components:

- `TextField.tsx` ✅
- `SliderField.tsx` ✅
- `DropdownField.tsx` ✅
- `ToggleField.tsx` ✅
- `ColorField.tsx` ✅

Field Rendering:

```
function renderField(field: FieldConfig, value: any, onChange: (value: any) => vo
  switch(field.type) {
    case 'text':
      return <TextField {...field} value={value} onChange={onChange} />
    case 'slider':
      return <SliderField {...field} value={value} onChange={onChange} />
    case 'dropdown':
      return <DropdownField {...field} value={value} onChange={onChange} />
    case 'toggle':
      return <ToggleField {...field} value={value} onChange={onChange} />
    case 'color':
      return <ColorField {...field} value={value} onChange={onChange} />
  }
}
```

---

9. Performance Considerations

9.1 Rendering Optimization

Stack Cards:

- Only render visible 4 cards (hide cards 5+)
- Use `React.memo` on MiniNodeCard component
- Use `will-change: transform` on animating elements

Lines:

- Reuse `LiquidMetalLine` component (already optimized)
- Max lines: 1 active + 4 confirmed = 5 total
- Acceptable performance impact

Blur Effect:

- Backdrop blur on container (not individual elements)
- 8px is reasonable (not too heavy)
- Applied to one element only

9.2 Animation Performance

GPU Acceleration:

```
// All animations use transform (GPU-accelerated)
const animatedProps = {
  transform: 'translate3d()', // ✅
  scale: '',                  // ✅
  opacity: '',                // ✅
  rotate: ''                  // ✅
}

// Avoid these (CPU-heavy)
const avoid = {
  width: '',       // ❌
  height: '',      // ❌
  top: '',         // ❌
  left: ''         // ❌
}
```

Target FPS: 60fps

Expected Performance:

- Entry animation: 60fps
- Stack rotation: 60fps
- Orbit animation: 55-60fps (slight drop acceptable)
- Exit animation: 60fps

---

10. Implementation Priority

High Priority (Must Have)

1. ✅ **IrisOrb shrink/grow with blur**
2. ✅ **Mini node stack with 4 visible cards**
3. ✅ **Glass line to active card**
4. ✅ **Stack rotation (carousel)**
5. ✅ **Save button + confirmation**
6. ✅ **Orbit animation**
7. ✅ **Value persistence (localStorage)**

Medium Priority (Should Have)

8. ✅ **Lines to confirmed orbit nodes**
9. ✅ **Click orbit node to recall**
10. ✅ **Click behind cards to navigate**
11. ✅ **Entry/exit transitions**

Low Priority (Nice to Have)

12. ⭐ **Keyboard navigation**
13. ⭐ **Sound effects (click, confirm)**
14. ⭐ **Validation errors display**

15. ⭐ **Field-specific help text**

---

11. Open Questions / Design Decisions

Q1: Empty Stack Behavior

What happens when all 4 mini nodes are confirmed?

Option A: Stack disappears, only orbiting nodes remain **Option B:** Show “All Complete” message **Option C:** Auto-advance to next subnode

Recommendation: Option B with auto-back to L3 after 2 seconds

---

Q2: Max Confirmed Nodes

What if user tries to confirm more than can fit in orbit?

Current orbit can fit: ~8-12 nodes comfortably at 90px size

Options:

- A: Limit to 8 confirmed nodes max
- B: Make orbit nodes smaller if > 8
- C: Second orbit ring (probably overkill)

Recommendation: Option A (limit to 8, disable save button after)

Q3: Field Validation

When should validation happen?

Options:

- A: On save click (show error message if invalid)
- B: Real-time (show error as user types)
- C: No validation (accept anything)

Recommendation: Option A (validate on save, show inline error)

---

12. Success Metrics

Visual Quality

- IrisOrb shrink/grow is smooth (no jank)
- Glass lines look polished (gradient flows)
- Stack rotation feels satisfying (carousel effect)
- Orbit animation is elegant (arc path)
- Blur effect creates depth (focus on orb)

Interaction Quality

- Save button is clear and responsive
- Stack navigation is intuitive
- Clicking orbit nodes works as expected
- Entry/exit transitions are seamless
- No navigation breaks or stuck states

Technical Quality

- 60fps on all animations
- Values persist across navigation
- No console errors
- Clean code structure
- Well-commented

🚀 Ready for Implementation!

This PRD covers:

- ✅ All visual specifications
- ✅ All interaction models
- ✅ All animations (entry, rotation, orbit, exit)

- ✅ State management
- ✅ Data structures
- ✅ Performance targets
- ✅ Implementation priorities

Next Step: Create implementation phases document?

Estimated Time: 15-20 hours

- Infrastructure: 3h
- Stack system: 5h
- Orbit system: 4h
- Transitions: 4h
- Polish: 2-3h

END OF LEVEL 4 PRD