Scene 1: Constellation View - Implementation Prompt

ConstellationView - Network Visualization Implementation

Visualization Purpose
Create a force-directed network graph that shows how attendees cluster differently based on which survey attribute is being examined. This visualization should clearly demonstrate that people who seem different in one dimension may be similar in others.

Pre-Implementation Investigation
Before coding, investigate:
1. Check existing D3 implementations in the codebase for patterns
2. Look for any force-layout utilities already created
3. Examine how other components handle window resizing
4. Find existing color constants or theme configurations
5. Check for any existing tooltip or hover state patterns

Core Concept
- Each attendee is a node in space
- Nodes cluster based on the currently selected attribute (learning style, motivation, etc.)
- Lines connect nodes that share other attributes
- The network reorganizes smoothly when switching between different attributes
- Visual emphasis on how the same people group differently based on the lens

Visual Design Specifications

Layout
- Full screen with appropriate padding for the header/UI elements
- Force-directed layout with boundaries to keep nodes on screen
- Nodes should have enough space to not feel cramped but close enough to see

patterns

Node Design

- Size: Base size + slight variation based on years at Medtronic
- Color: Mapped to years_at_medtronic (use existing color scheme if found)
- Border: Subtle glow effect on hover
- Opacity: Full opacity for active nodes, reduced for non-selected in interactive mode

Connection Lines

- Default: Thin, semi-transparent lines showing shared attributes
- Thickness: Increases with more shared attributes
- Color: Subtle gradient from source to target node
- Animation: Gentle pulse along the line in auto-play mode

Key Features to Implement

1. **Attribute Selector**
   - UI element to switch between: learning_style, shaped_by, peak_performance, motivation
   - Smooth transition between layouts (1.2s duration)
   - Clear labeling of what's currently being shown

2. **Auto-Play Mode**
   - Cycles through each attribute every 30-45 seconds
   - Smooth transitions with a brief pause on each view
   - Optional: Highlight interesting clusters with subtle animations

3. **Interactive Mode**
   - Click nodes to see details

- Hover to highlight all connections
- Filter options to show only certain segments
- Zoom and pan capabilities

4. **Insight Display**
   - Show key statistics like:
     - "X% of attendees share this [attribute]"
     - "Most common connection: [attribute1] + [attribute2]"
     - Number of unique clusters formed
   - Update dynamically based on current view

D3.js Implementation Approach

Force Simulation Setup
```javascript
// Investigate if these D3 modules are already imported
// d3-force, d3-selection, d3-transition, d3-scale

// Basic force setup (adapt based on existing patterns)
const simulation = d3.forceSimulation()
  .force("charge", d3.forceManyBody().strength(-50))
  .force("link", d3.forceLink().id(d ⇒ d.id))
  .force("center", d3.forceCenter(width/2, height/2))
  .force("collision", d3.forceCollide().radius(d ⇒ nodeRadius(d)))
  .force("boundary", boundaryForce()) // Custom force to keep nodes in view
```

Data Structure Needs

```
// Transform survey data into network format
// Nodes: Each survey response
// Links: Created based on shared attributes
// Groups: Based on current attribute selection
```

```
// Calculate connections dynamically based on selected attribute
// Store precalculated connections for performance
```

Animation Patterns

1. **Entry Animation**:
   - Nodes appear from center and expand outward
   - Links fade in after nodes are positioned
   - UI elements slide in from edges
2. **Transition Animation** (between attributes):
   - Nodes smoothly move to new positions
   - Links fade out/in as connections change
   - Brief "gathering" at center before reorganizing
3. **Idle Animations** (auto-play):
   - Subtle node breathing effect
   - Occasional highlight of interesting connections
   - Gentle force adjustments to prevent static layouts

React Integration Pattern

```
// Component structure suggestion (follow existing patterns)
const ConstellationView = () => {
  // Hooks for data, following existing patterns
  // State for mode, selected attribute, filters
  // Ref for D3 container

  // D3 initialization in useEffect
  // Cleanup on unmount
  // Resize handling
```

```
  // Return container with consistent layout structure
}
```

Performance Optimization Strategies

1. Use canvas rendering if SVG becomes slow with 600 nodes
2. Implement level-of-detail (show fewer connections when zoomed out)
3. Debounce force calculations during transitions
4. Pre-calculate connection data to avoid runtime computation
5. Use React.memo and useMemo appropriately

Specific Interactions to Implement

Auto-Play Sequence

1. Start with learning_style clustering (15s)
2. Transition to shaped_by (15s)
3. Transition to peak_performance (15s)
4. Transition to motivation (15s)
5. Show "all connections" view briefly (10s)
6. Loop back to start

Interactive Features

- Click cluster background to isolate that group
- Shift+click to select multiple nodes
- Double-click background to reset view
- Hover node to see full details
- Hover connection to see what attributes are shared

Accessibility Considerations

- Keyboard navigation (Tab through clusters, Enter to select)

- Screen reader descriptions of current clustering
- High contrast mode option
- Reduced motion mode (instant transitions)

Testing Checklist

- [ ] Loads smoothly with 600 nodes
- [ ] Transitions are smooth (60fps)
- [ ] Auto-play runs indefinitely without memory leaks
- [ ] Works on 16:9, 16:10, and other aspect ratios
- [ ] Touch interactions work (for interactive displays)
- [ ] Insights update correctly
- [ ] No overlapping nodes that obscure data

Questions to Resolve During Implementation

1. How should isolated nodes (no connections) be handled visually?
2. Should cluster labels appear automatically or on hover?
3. What's the maximum number of connections to show at once?
4. How to handle real-time updates if new responses come in?
5. Should there be a legend explaining the color coding?

Remember

- The goal is to show "different yet similar" - make sure this is obvious
- The visualization should be mesmerizing in auto-play mode
- But also informative and explorable in interactive mode
- Keep checking existing code patterns and reuse where possible