Master System Prompt - DataVisualization Implementation Guide

# Medtronic WE Summit - Data Visualization Implementation Guide

## Context
You are implementing data visualizations for the Medtronic WE Summit project.
The core message to communicate is "We are different, but we are also similar" - showing how attendees connect across various dimensions.

## IMPORTANT: Codebase Investigation First
Before implementing anything:
1. Examine the existing project structure and naming conventions
2. Check how components are currently organized in `/src/components/`
3. Review any existing shared components or utilities
4. Look at the current TypeScript interfaces and types
5. Understand how Supabase is configured and how data is fetched
6. Check for any existing animation patterns or libraries already in use

## Working Directory
Your implementations will go in: `/src/components/DataVisualization/`
- Follow the existing project's file naming conventions
- Reuse any existing shared components or patterns
- Extend existing TypeScript types rather than creating new ones

## Data Investigation
1. First, examine the actual data structure by:
   - Checking Supabase schema or existing type definitions
   - Looking at any data fetching hooks or utilities
   - Understanding all possible values for each survey question

2. The data includes responses about:
   - Years at Medtronic
   - Learning styles
   - What shaped them growing up
   - Peak performance preferences
   - Current motivations
   - Unique qualities (free text)

3. Use the actual data structure found in the codebase, not assumptions

## Technical Approach

### Stack Verification
Confirm these are available (check package.json):
- Next.js with TypeScript
- React
- D3.js (v7.8.5)
- Tailwind CSS
- Supabase client

If additional animation libraries are needed (like framer-motion), note them for installation.

### Design System Alignment
1. Check for existing:
   - Color variables or Tailwind config
   - Font families and sizes
   - Spacing units
   - Animation durations
   - Component patterns

2. For visualizations, aim for:
   - Dark theme (check existing dark colors)
   - 16:9 aspect ratio optimization
   - Smooth transitions
   - Clear data communication

### Implementation Pattern
Each visualization should:
1. Have both auto-play and interactive modes
2. Include entry animations when the page loads
3. Handle real-time data updates from Supabase
4. Be performant with 600+ data points
5. Include key insights or statistics display

## Shared Components & Utilities

### Location
Create shared components in: `/src/components/DataVisualization/shared/`

### Essential Shared Components

#### 1. VisualizationContainer
A wrapper component that provides consistent layout for all visualizations:
- Handles viewport sizing (16:9 optimization)
- Provides consistent padding/margins
- Manages fullscreen capability
- Handles resize events
- Provides loading and error states

#### 2. ModeToggle
Controls auto-play vs interactive mode:
- Consistent positioning (top-right corner)
- Smooth transition between modes
- Persists user preference
- Provides mode context to child components
- Visual feedback for current mode

#### 3. DataInsightPanel
Displays contextual statistics and insights:
- Slides in from right side
- Updates based on current visualization state

- Consistent styling across all visualizations
- Smooth number animations for statistics
- Support for different insight types

#### 4. NavigationDots
Navigation between visualizations:
- Fixed position (bottom or side)
- Shows current visualization
- Smooth transitions between views
- Keyboard navigation support
- Optional: Auto-advance in presentation mode

### Essential Utilities

#### 1. Data Fetching Hook (useVisualizationData.ts)
- Fetch from Supabase
- Handle real-time updates
- Provide loading/error states
- Cache data appropriately
- Transform data for D3 consumption

#### 2. Animation Helpers (animationUtils.ts)
- Standard easing functions
- Entry animation orchestrator
- Transition timing constants
- Performance monitoring
- Frame rate optimization

#### 3. Color Utilities (colorUtils.ts)
- Years to color mapping
- Attribute color scales
- Accessibility checking
- Color interpolation
- Theme constants

#### 4. D3 Helpers (d3Utils.ts)

- Responsive SVG setup
- Standard transitions
- Force simulation configs
- Path generation helpers
- Performance optimizations

## State Management

### Global Visualization State
Consider implementing a context or lightweight state manager for:

- Current visualization mode (auto/interactive)
- Active filters across visualizations
- User preferences (reduced motion, etc.)
- Shared data cache
- Navigation state

### Local Storage
Persist user preferences:

- Preferred mode
- Reduced motion setting
- Recently viewed insights
- Custom filter combinations

## Navigation and Routing

### Routing Structure
Set up routes for each visualization in Next.js app directory:

```
/app/
/visualization/
layout.tsx (shared layout for all visualizations)
/constellation/
page.tsx
/tapestry/
page.tsx
```

/comparison/
page.tsx
/waves/
page.tsx
/qualities/
page.tsx

### Navigation Implementation
1. **Primary Navigation**
   - Implement smooth transitions between routes
   - Preload adjacent visualizations
   - Support keyboard shortcuts (1-5 for each viz)
   - Browser back/forward support

2. **Presentation Mode**
   - Removes unnecessary UI elements
   - Auto-advances through visualizations
   - Quick return to specific visualization

3. **URL Parameters**
   - `?mode=auto|interactive`
   - `?filter=attribute:value`
   - `?presentation=true`

## Performance & LED Wall Optimization

### LED Wall Considerations
- Resolution: 4K minimum (3840×2160)
- Aspect ratio: Strict 16:9
- Viewing distance: 10-50 feet
- Increase contrast ratios
- Thicker lines and larger text
- Smooth animations (60fps)
- Avoid pure blacks (use #0A0A0F)

### Performance Targets
- Initial load: < 3 seconds
- Transition start: < 100ms
- Animation FPS: 60fps consistent
- Memory usage: < 500MB
- CPU usage: < 50% sustained

### Optimization Strategies
1. **Rendering**
   - Use `will-change` CSS property
   - Implement requestAnimationFrame
   - Use CSS transforms over position
   - Consider WebGL for complex viz

2. **Data**
   - Pre-calculate expensive operations
   - Use efficient data structures
   - Index for quick lookups

## Accessibility Requirements
- Keyboard navigation support
- Screen reader descriptions
- High contrast mode option
- Reduced motion option
- Live regions for updates

## Error Handling
- Graceful fallbacks for data issues
- Error boundaries for component crashes
- User-friendly error messages
- Recovery options

## Development Process

1. **Investigation Phase**
   - Explore codebase structure

   - Understand data schema
   - Identify reusable components
   - Note any missing dependencies

2. **Shared Components First**
   - Build VisualizationContainer
   - Create ModeToggle
   - Implement shared utilities
   - Set up navigation structure

3. **Individual Visualizations**
   - Build incrementally
   - Test with real data early
   - Ensure smooth animations
   - Implement both modes

4. **Integration & Polish**
   - Connect all visualizations
   - Optimize performance
   - Test on target hardware
   - Polish transitions

## Testing Requirements
- Test with full 600-person dataset
- Run for extended periods (4+ hours)
- Monitor memory leaks
- Verify on LED wall hardware
- Test all aspect ratios

## Key Questions During Implementation
- "How is this pattern already implemented elsewhere?"
- "What naming convention is being used?"
- "Are there existing utilities I should use?"
- "Is this performant with 600 data points?"
- "Does this clearly show 'different yet similar'?"
- "Will this work on the LED wall?"

## Remember
- This is for a keynote presentation - clarity and visual impact are crucial
- The visualizations should feel cohesive as a set
- Auto-play mode should be engaging for passive viewing
- Interactive mode should reveal deeper insights
- Always investigate existing patterns before implementing new ones