**Implementation Plan: Centralized Quadrant Assignment System**

**Overview**

This document outlines the plan to implement a centralized system for managing quadrant assignments in the Apostles Model visualization tool. The goal is to ensure consistency between the visualization and reporting components when users interact with the chart by:

1. Moving the midpoint position
2. Manually reassigning points on boundary lines
3. Changing other visualization parameters

**Current Limitations**

* MidpointContext only stores position without quadrant assignment logic
* Quadrant assignments are calculated independently by different components
* Manual reassignments only exist in local state in DataPointRenderer
* Visualization and reporting can show inconsistent quadrant assignments
* No persistence of manual quadrant assignments

**Proposed Solution**

Create a comprehensive QuadrantAssignmentContext that will:

1. Store the midpoint position
2. Calculate and provide quadrant assignments for all data points
3. Store manual reassignments when users change boundary points
4. Provide distribution statistics based on current assignments
5. Act as a single source of truth for all components

**Implementation Steps**

**Step 1: Create QuadrantAssignmentContext**

Create a new context file: src/components/visualization/context/QuadrantAssignmentContext.tsx

This context will provide:

* Midpoint state management
* Manual assignment tracking
* Quadrant determination functions
* Distribution statistics

**Step 2: Integrate with App.tsx**

Add the QuadrantAssignmentProvider to App.tsx, wrapping the visualization and reporting components:

tsx

<QuadrantAssignmentProvider

data={data}

satisfactionScale={scales.satisfactionScale}

loyaltyScale={scales.loyaltyScale}

>

{*/\* Visualization and Reporting Components \*/*}

</QuadrantAssignmentProvider>

**Step 3: Update DataPointRenderer Component**

Modify DataPointRenderer to:

* Use the QuadrantAssignmentContext for quadrant determination
* Update the context when manual reassignments occur
* Remove local manualGroups state

**Step 4: Update Visualization Components**

Update other visualization components to use the context:

* QuadrantChart.tsx
* SpecialZoneRenderer.tsx
* Components that read or display quadrant information

**Step 5: Update Reporting Components**

Modify reporting components to use the context:

* DistributionSection.tsx
* ProximityAnalysis.tsx
* ProximityList.tsx
* ProximityTileMap.tsx
* DataReport.tsx

**Step 6: Enhance Export Functionality**

Ensure the export functionality uses the context to:

* Include current quadrant assignments for each data point
* Reflect any manual reassignments made by the user
* Provide consistent quadrant distribution statistics

**Technical Details**

**QuadrantAssignmentContext Interface**

typescript

interface QuadrantAssignmentContextType {

*// Midpoint state*

midpoint: { sat: number; loy: number };

setMidpoint: (newMidpoint: { sat: number; loy: number }) => void;

*// Manual assignments management*

manualAssignments: Map<string, QuadrantType>;

updateManualAssignment: (pointId: string, quadrant: QuadrantType) => void;

clearManualAssignment: (pointId: string) => void;

*// Quadrant determination*

getQuadrantForPoint: (point: DataPoint) => QuadrantType;

*// Distribution statistics*

distribution: Record<QuadrantType, number>;

*// Scale information*

satisfactionScale: ScaleFormat;

loyaltyScale: ScaleFormat;

}

**Quadrant Types**

typescript

type QuadrantType =

'loyalists' |

'mercenaries' |

'hostages' |

'defectors' |

'apostles' |

'terrorists' |

'near\_apostles' |

'near\_terrorists';

**Benefits**

* Single source of truth for quadrant assignments
* Consistent display across visualization and reporting
* Persistent manual assignments across the application
* Simplified component logic (DRY principle)
* Improved maintainability and testability
* Flexible foundation for future reporting and export features

**Implementation Schedule**

1. Create and test QuadrantAssignmentContext
2. Update App.tsx to incorporate the provider
3. Modify DataPointRenderer to use the context
4. Update visualization components
5. Update reporting components
6. Enhance export functionality
7. Test entire flow and fix any issues

**Expected Impact**

This implementation will ensure that users see consistent data across all parts of the application, particularly:

* Quadrant distributions will match exactly between visualization and reports
* Manual reassignments will persist throughout the application
* Exports will accurately reflect the user's customizations