Dashboard Template Architecture: Storage & Data Merging

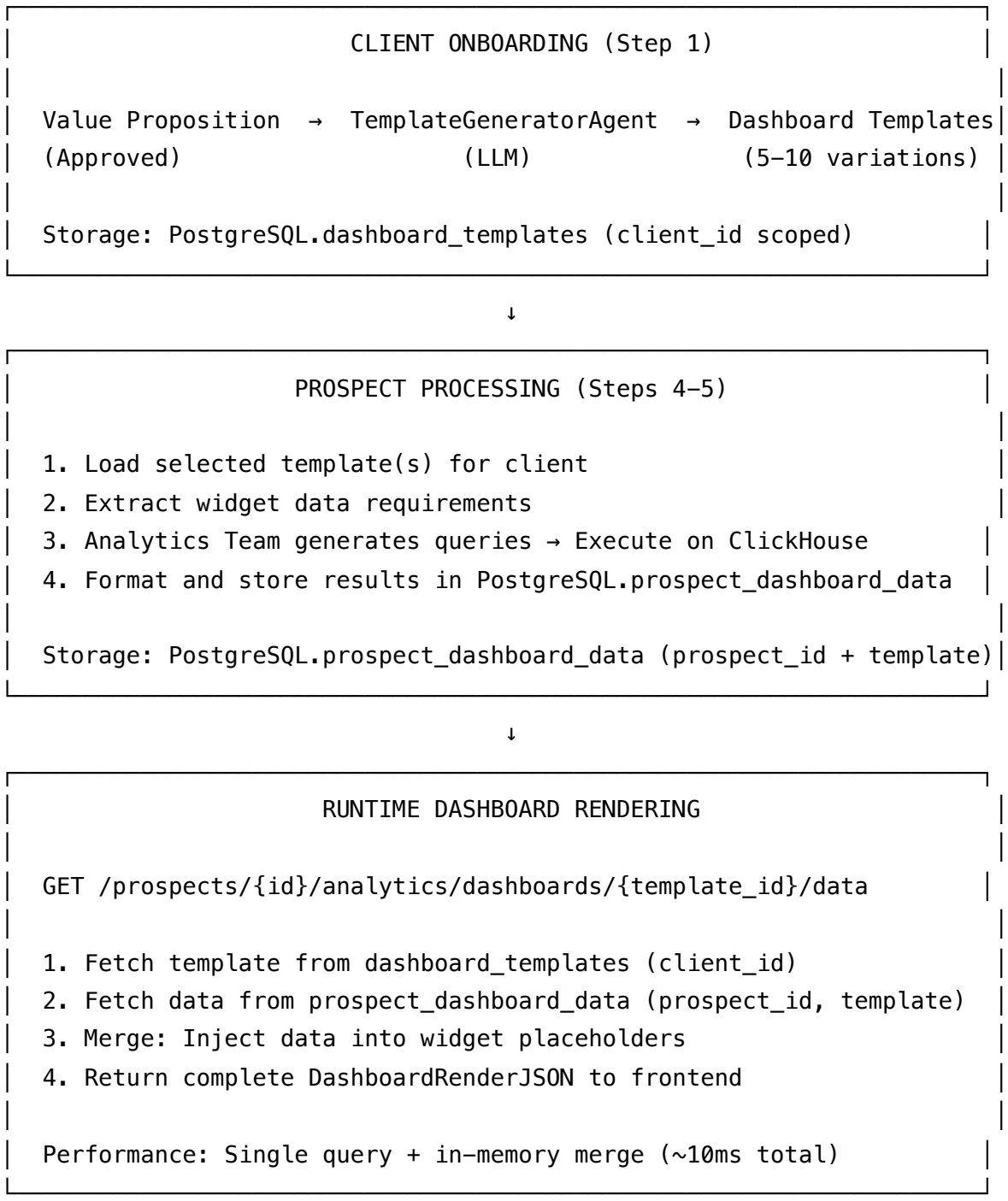
Executive Summary

This document outlines the architecture for storing LLM-generated dashboard templates and merging them with prospect-specific analytics data at runtime. The system enables a clear separation between **template definitions** (client-level, reusable) and **data instances** (prospect-specific, generated from ClickHouse queries).

Key Design Principles:

1. **Template Reusability**: Dashboard templates are client-scoped and reusable across multiple prospects
2. **Data Separation**: Template structure is separate from prospect data
3. **Pre-computation**: Analytics data is generated once and cached in PostgreSQL JSONB
4. **Fast Runtime**: Dashboard rendering uses simple template + data merge (~10ms retrieval)
5. **Schema-Driven**: Widget metadata drives analytics query generation
6. **LLM-Driven Design**: Templates are dynamically generated by LLM based on client value propositions, allowing flexibility in dashboard design per client

Architecture Overview



1. Template Storage Schema

1.1 Database Schema

```sql
CREATE TABLE dashboard_templates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    client_id UUID NOT NULL REFERENCES clients(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    category VARCHAR(100) NOT NULL,
    target_audience VARCHAR(100) NOT NULL,
    visual_style JSONB NOT NULL,
    widgets JSONB NOT NULL,  -- Array of WidgetConfiguration
    metadata JSONB,
    status VARCHAR(50) DEFAULT 'generated',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT chk_category CHECK (category IN (
        'roi-focused', 'clinical-outcomes', 'operational-efficiency',
        'competitive-positioning', 'comprehensive'
    )),
    CONSTRAINT chk_status CHECK (status IN ('generated', 'approved', 'removed'))
);

CREATE INDEX idx_dt_client_id ON dashboard_templates(client_id);
CREATE INDEX idx_dt_category ON dashboard_templates(category);
CREATE INDEX idx_dt_status ON dashboard_templates(status);
CREATE INDEX idx_dt_widgets_gin ON dashboard_templates USING GIN(widgets);
```

Template Versioning: Templates are updated in-place (full replacement). When a template is modified, all existing prospect data remains unchanged. New data generation will use the updated template definition.

1.2 Template JSON Structure

```
interface DashboardTemplate {
  id: string                    // UUID generated by backend
  client_id: string             // Links to client
  name: string                  // "Diabetes ROI Executive Dashboard"
  description: string           // Brief overview
  category: DashboardCategory
  target_audience: string       // "Health Plan", "Broker", etc.
  visual_style: VisualStyle
  widgets: WidgetConfiguration[]
  metadata: TemplateMetadata
  status: 'generated' | 'approved' | 'removed'
  created_at: string            // ISO timestamp
  updated_at: string            // ISO timestamp
}

interface VisualStyle {
  primary_color: string         // "#1976d2"
  accent_color: string          // "#2e7d32"
  layout: 'dense' | 'balanced' | 'spacious'
  theme?: 'light' | 'dark'      // Optional theme preference
}

interface TemplateMetadata {
  generated_by: 'llm' | 'manual'
  llm_model?: string            // "claude-3-5-sonnet-20241022"
  generation_timestamp?: string
  recommended_use_case?: string
  key_features?: string[]       // Bullet points for preview
}
```

1.3 Widget Configuration Schema

This is the critical bridge between templates and analytics.

```
interface WidgetConfiguration {
  widget_id: string              // "w_hba1c_trend" - unique within template
  widget_type: WidgetType        // Determines rendering component (see Section 1.4)
  title: string                  // Display title
  description: string            // Tooltip/help text

  // LAYOUT
  position: GridPosition         // Where to place in grid
  size: WidgetSize              // Responsive sizing

  // DATA BINDING
  data_requirements: DataRequirement  // ← KEY: What data is needed
  analytics_question: string     // Natural language query for analytics team

  // VISUALIZATION
  chart_config?: ChartConfig    // Chart-specific settings (Chart.js options)
  formatting?: FormattingRules  // Number formatting, colors, etc.

  // METADATA
  priority_mapping?: string     // Links to value prop priority ID
}

interface GridPosition {
  row: number               // Grid row (1-indexed)
  col: number               // Grid column (1-indexed)
  row_span: number          // How many rows to occupy
  col_span: number          // How many columns to occupy
}

interface WidgetSize {
  min_width?: number        // Minimum width in grid units
  min_height?: number       // Minimum height in grid units
  aspect_ratio?: string     // "16:9", "4:3", etc.
}
```

1.4 Widget Type Registry (Chart.js Based)

The frontend uses **Chart.js v4.5** for visualizations. The LLM can select from these predefined widget types when generating templates:

```
type WidgetType =
  // ===== KPI & METRICS =====
  | 'kpi-card'              // Single metric with icon, trend indicator, subtitle
  | 'kpi-grid'              // Multiple related KPIs in a grid
  | 'metric-comparison'     // Side-by-side metric comparison
  | 'scorecard'             // Multiple metrics with progress indicators

  // ===== CHART.JS CHARTS =====
  // Line Charts
  | 'line-chart'            // Standard line chart (trends over time)
  | 'multi-line-chart'      // Multiple series on same chart
  | 'area-chart'            // Filled area under line
  | 'stacked-area-chart'    // Multiple stacked areas

  // Bar Charts
  | 'bar-chart'             // Vertical bars (comparisons)
  | 'horizontal-bar-chart'  // Horizontal bars
  | 'stacked-bar-chart'     // Stacked vertical bars
  | 'grouped-bar-chart'     // Grouped bars for multi-series

  // Pie & Donut Charts
  | 'pie-chart'             // Circular pie chart (composition)
  | 'donut-chart'           // Donut chart (composition with center label)

  // Specialized Charts
  | 'waterfall-chart'       // Cumulative changes (savings breakdown)
  | 'scatter-plot'          // X-Y scatter plot (correlations)
  | 'bubble-chart'          // 3D scatter (size dimension)
  | 'radar-chart'           // Multi-dimensional comparison
  | 'polar-area-chart'      // Circular bars

  // ===== CUSTOM WIDGETS =====
  | 'progress-bar'          // Single progress indicator
  | 'gauge-chart'           // Semi-circle gauge (progress to goal)
  | 'timeline-chart'        // ROI timeline with breakeven marker
  | 'quality-progression'   // Multi-metric progress bars with targets
  | 'ranked-list'           // Ordered list with metadata (opportunities, conditions)
  | 'data-table'            // Sortable, filterable table
  | 'heatmap'               // Grid-based heatmap
  | 'trend-sparkline'       // Mini trend indicator

  // ===== COMPOSITE WIDGETS =====
  | 'roi-summary'           // ROI percentage + payback + savings
```

```
  | 'clinical-metrics'      // Health outcome metrics with trends
  | 'cost-breakdown'        // Multi-level cost analysis
```

Widget Type Usage Guidelines for LLM:

```
# Provide this to TemplateGeneratorAgent in prompt
widget_type_guidelines:
  roi_dashboards:
    - kpi-card: "ROI %, payback period, total savings, cost per member"
    - waterfall-chart: "Savings breakdown by category"
    - timeline-chart: "Cumulative ROI projection with breakeven"
    - bar-chart: "Cost comparisons, scenario analysis"

  clinical_dashboards:
    - kpi-card: "Key clinical metrics (HbA1c, BP control, adherence)"
    - line-chart: "Clinical trends over time"
    - multi-line-chart: "Compare multiple clinical measures"
    - quality-progression: "Progress toward clinical targets"
    - ranked-list: "Top conditions, high-risk members"

  operational_dashboards:
    - kpi-card: "Efficiency metrics, utilization rates"
    - multi-line-chart: "ED visits, admissions trends"
    - stacked-bar-chart: "Service utilization breakdown"
    - heatmap: "Risk stratification, regional patterns"

  comprehensive_dashboards:
    - Use mix of types across all categories
    - Limit to 8-12 widgets maximum
    - Balance KPIs (4), charts (4-6), lists/tables (1-2)
```

1.5 Data Requirements Schema

This is what the Analytics Team uses to generate queries.

```
interface DataRequirement {
  query_type: QueryType
  dimensions: string[]          // Group by fields
  metrics: MetricDefinition[]   // What to calculate
  filters?: FilterCondition[]   // WHERE clauses
  time_range?: TimeRange        // Date filtering
  aggregation?: AggregationType
  limit?: number               // Top N results
  sort?: SortDefinition
}

type QueryType =
  | 'aggregate'      // SUM, AVG, COUNT
  | 'time-series'    // Monthly trends
  | 'distribution'   // Histogram/grouping
  | 'comparison'     // Before/after, cohorts
  | 'ranking'        // Top conditions, etc.

interface MetricDefinition {
  name: string                  // "avg_hba1c", "total_cost"
  expression: string            // "AVG(hba1c_value)", "SUM(paid_amount)"
  data_type: 'number' | 'currency' | 'percentage' | 'count'
  format?: string               // "$0,0.00", "0.0%", etc.
}

interface FilterCondition {
  field: string
  operator: 'eq' | 'gt' | 'lt' | 'in' | 'between' | 'contains'
  value: any
}

interface TimeRange {
  type: 'relative' | 'absolute'
  value: string                 // "last_12_months", "2024-01-01|2024-12-31"
}

interface ChartConfig {
  // Chart.js configuration options
  chart_type?: 'line' | 'bar' | 'pie' | 'doughnut' | 'radar' | 'polarArea' | 'bubble' |
  x_axis?: string               // Field name for x-axis
  y_axis?: string | string[]    // Field name(s) for y-axis
  color?: string | string[]     // Color(s) for series
  show_legend?: boolean
```

```
  show_grid?: boolean
  show_tooltip?: boolean
  show_target_line?: boolean    // For goals/thresholds
  target_value?: number
  stacked?: boolean             // For stacked charts
  tension?: number              // Line smoothing (0-1)
  fill?: boolean                // Fill area under line
  point_radius?: number
  border_width?: number
  // Additional Chart.js options can be passed through
  [key: string]: any
}
```

1.6 Example Template JSON


```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "client_id": "client-abc-123",
  "name": "Diabetes ROI Executive Dashboard",
  "description": "Financial impact dashboard emphasizing 3-year diabetes complication s
  "category": "roi-focused",
  "target_audience": "Broker",
  "visual_style": {
    "primary_color": "#1976d2",
    "accent_color": "#2e7d32",
    "layout": "balanced"
  },
  "widgets": [
    {
      "widget_id": "w_roi_percentage",
      "widget_type": "kpi-card",
      "title": "Total ROI",
      "description": "Return on investment over 24 months",
      "position": { "row": 1, "col": 1, "row_span": 1, "col_span": 1 },
      "data_requirements": {
        "query_type": "aggregate",
        "metrics": [
          {
            "name": "roi_percentage",
            "expression": "(SUM(savings) / SUM(costs) - 1) * 100",
            "data_type": "percentage",
            "format": "0.0%"
          }
        ],
        "time_range": { "type": "relative", "value": "last_24_months" }
      },
      "analytics_question": "What is the total ROI percentage for diabetes program over
      "chart_config": {
        "icon": "tabler-trending-up",
        "show_trend": true
      }
    },
    {
      "widget_id": "w_utilization_trends",
      "widget_type": "multi-line-chart",
      "title": "Utilization Trends",
```

```
  "description": "Monthly ED visits and hospitalizations for diabetic members",
  "position": { "row": 2, "col": 1, "row_span": 2, "col_span": 3 },
  "data_requirements": {
    "query_type": "time-series",
    "dimensions": ["month"],
    "metrics": [
      {
        "name": "ed_visits",
        "expression": "COUNT(DISTINCT claim_id) FILTER (WHERE service_type = 'ED')"
        "data_type": "count"
      },
      {
        "name": "hospitalizations",
        "expression": "COUNT(DISTINCT claim_id) FILTER (WHERE service_type = 'Inpat
        "data_type": "count"
      }
    ],
    "filters": [
      { "field": "diagnosis_code", "operator": "in", "value": ["E11.*"] }
    ],
    "time_range": { "type": "relative", "value": "last_12_months" }
  },
  "analytics_question": "What are the monthly ED visits and hospitalizations for di
  "chart_config": {
    "chart_type": "line",
    "x_axis": "month",
    "y_axis": ["ed_visits", "hospitalizations"],
    "color": ["#1976d2", "#f57c00"],
    "tension": 0.4,
    "fill": true,
    "show_legend": true,
    "show_grid": true
  }
},
{
  "widget_id": "w_savings_waterfall",
  "widget_type": "waterfall-chart",
  "title": "Savings Waterfall",
  "description": "Total savings breakdown by category",
  "position": { "row": 2, "col": 4, "row_span": 2, "col_span": 1 },
  "data_requirements": {
    "query_type": "aggregate",
    "dimensions": ["savings_category"],
```

```
        "metrics": [
          {
            "name": "total_savings",
            "expression": "SUM(avoided_cost)",
            "data_type": "currency"
          }
        ],
        "filters": [
          { "field": "savings_category", "operator": "in", "value": [
            "ED Visit Reduction", "Hospitalization Avoidance",
            "Medication Adherence", "Complication Prevention"
          ]}
        ]
      },
      "analytics_question": "What are the total savings by category?",
      "chart_config": {
        "chart_type": "bar",
        "x_axis": "savings_category",
        "y_axis": "total_savings",
        "color": "#2e7d32"
      }
    }
  ],
  "metadata": {
    "generated_by": "llm",
    "llm_model": "claude-3-5-sonnet-20241022",
    "generation_timestamp": "2025-01-15T10:30:00Z",
    "key_features": [
      "2.6x to 4.1x ROI scorecard with three scenarios",
      "Three-year savings per member ($2,828-$4,477)",
      "Payback timeline (12-18 months)"
    ],
    "recommended_use_case": "Best for: CFO and financial decision-makers"
  },
  "status": "approved",
  "created_at": "2025-01-15T10:30:00Z",
  "updated_at": "2025-01-15T14:20:00Z"
}
```

2. Data Storage Schema (Prospect-Specific)

2.1 Database Schema

```
CREATE TABLE prospect_dashboard_data (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    prospect_id UUID NOT NULL REFERENCES prospects(id) ON DELETE CASCADE,
    template_id UUID NOT NULL REFERENCES dashboard_templates(id),
    dashboard_data JSONB NOT NULL,  -- Complete widget data
    generated_at TIMESTAMP NOT NULL,
    generation_duration_ms INTEGER, -- Performance tracking
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(prospect_id, template_id)  -- One data instance per template per prospect
);

CREATE INDEX idx_prospect_dashboard_lookup
    ON prospect_dashboard_data(prospect_id, template_id);
CREATE INDEX idx_prospect_dashboard_jsonb
    ON prospect_dashboard_data USING GIN(dashboard_data);
```

2.2 Dashboard Data JSON Structure

```
interface ProspectDashboardData {
  prospect_id: string
  template_id: string
  generated_at: string           // ISO timestamp
  widgets_data: WidgetData[]     // Data for each widget
  metadata: DataGenerationMetadata
}

interface WidgetData {
  widget_id: string              // Links to WidgetConfiguration.widget_id
  data_points: DataPoint[]       // Actual data
  summary_stats?: SummaryStats   // Min, max, avg, etc.
  query_metadata: QueryMetadata
}

interface DataPoint {
  timestamp?: string             // For time-series
  label?: string                 // For categorical data
  value: number                  // Primary numeric value
  formatted_value?: string       // "$1.2M", "82.5%", etc.
  metadata?: Record<string, any> // Additional context
  breakdown?: DataPoint[]        // For nested/drilldown data
}

interface SummaryStats {
  min: number
  max: number
  avg: number
  total?: number
  count: number
  std_dev?: number
}

interface QueryMetadata {
  query_used: string             // SQL query (for debugging/audit)
  execution_time_ms: number      // Performance tracking
  row_count: number              // How many rows returned
  generated_at: string
}

interface DataGenerationMetadata {
```

```
  total_widgets: number
  successful_widgets: number
  failed_widgets: WidgetFailure[]
  total_execution_time_ms: number
  queries_executed: number
  data_freshness: string         // "2024-12-31" - latest data date
}

interface WidgetFailure {
  widget_id: string
  error_type: 'query_error' | 'timeout' | 'validation_error' | 'data_not_found'
  error_message: string
  retry_count?: number
}
```

2.3 Example Dashboard Data JSON


```
{
  "prospect_id": "prospect-xyz-789",
  "template_id": "550e8400-e29b-41d4-a716-446655440000",
  "generated_at": "2025-01-15T15:30:00Z",
  "widgets_data": [
    {
      "widget_id": "w_roi_percentage",
      "data_points": [
        {
          "value": 245.7,
          "formatted_value": "245.7%",
          "metadata": {
            "total_savings": 12500000,
            "total_costs": 5100000,
            "time_period": "24 months"
          }
        }
      ],
      "query_metadata": {
        "query_used": "SELECT (SUM(savings) / SUM(costs) - 1) * 100 AS roi...",
        "execution_time_ms": 145,
        "row_count": 1,
        "generated_at": "2025-01-15T15:30:12Z"
      }
    },
    {
      "widget_id": "w_utilization_trends",
      "data_points": [
        {
          "timestamp": "2024-01-01",
          "label": "Jan",
          "value": 850,
          "formatted_value": "850",
          "metadata": { "series": "ed_visits" }
        },
        {
          "timestamp": "2024-01-01",
          "label": "Jan",
          "value": 320,
          "formatted_value": "320",
          "metadata": { "series": "hospitalizations" }
```

```
      },
      {
        "timestamp": "2024-02-01",
        "label": "Feb",
        "value": 820,
        "formatted_value": "820",
        "metadata": { "series": "ed_visits" }
      },
      {
        "timestamp": "2024-02-01",
        "label": "Feb",
        "value": 310,
        "formatted_value": "310",
        "metadata": { "series": "hospitalizations" }
      }
    ],
    "summary_stats": {
      "min": 310,
      "max": 850,
      "avg": 575,
      "count": 4
    },
    "query_metadata": {
      "query_used": "SELECT toStartOfMonth(service_date) AS month, COUNT(DISTINCT cla
      "execution_time_ms": 280,
      "row_count": 24,
      "generated_at": "2025-01-15T15:30:15Z"
    }
  }
],
"metadata": {
  "total_widgets": 8,
  "successful_widgets": 7,
  "failed_widgets": [
    {
      "widget_id": "w_advanced_analytics",
      "error_type": "timeout",
      "error_message": "Query exceeded 30s timeout",
      "retry_count": 3
    }
  ],
  "total_execution_time_ms": 2450,
  "queries_executed": 7,
```

```
    "data_freshness": "2024-12-31"
  }
}
```

3. Data Generation Process (Steps 4-5)

3.1 Analytics Team Workflow

```
# Step 5: Dashboard Data Generation
async def generate_dashboard_data(prospect_id: str, template_id: str) -> ProspectDashbo
    """
    Generate analytics data for all widgets in a dashboard template.

    Process:
    1. Load template configuration
    2. Extract widget data requirements
    3. For each widget, use Analytics Team to generate SQL query
    4. Execute queries against ClickHouse in parallel
    5. Format results according to widget specifications
    6. Store complete dashboard data in PostgreSQL
    """

    # 1. Load template
    template = await db.get_dashboard_template(template_id)
    prospect = await db.get_prospect(prospect_id)

    # 2. Initialize Analytics Team
    analytics_team = AnalyticsTeamCoordinator(
        clinical_agent=ClinicalAgent(),
        utilization_agent=UtilizationAgent(),
        pricing_agent=PricingAgent(),
        pharmacy_agent=PharmacyAgent(),
        sql_agent=QuestionToSQLAgent()
    )

    # 3. Process each widget in parallel
    widget_tasks = []
    for widget_config in template.widgets:
        task = generate_widget_data(
            widget_config=widget_config,
            prospect=prospect,
            analytics_team=analytics_team
        )
        widget_tasks.append(task)

    # Execute all queries concurrently
```

```python
widgets_data = await asyncio.gather(*widget_tasks, return_exceptions=True)

# 4. Separate successful and failed widgets
successful_widgets = []
failed_widgets = []

for widget_result in widgets_data:
    if isinstance(widget_result, Exception):
        failed_widgets.append({
            "widget_id": widget_result.widget_id,
            "error_type": classify_error(widget_result),
            "error_message": str(widget_result),
            "retry_count": 0
        })
    else:
        successful_widgets.append(widget_result)

# 5. Build final data structure
dashboard_data = {
    "prospect_id": prospect_id,
    "template_id": template_id,
    "generated_at": datetime.utcnow().isoformat(),
    "widgets_data": successful_widgets,
    "metadata": {
        "total_widgets": len(template.widgets),
        "successful_widgets": len(successful_widgets),
        "failed_widgets": failed_widgets,
        "total_execution_time_ms": sum(w.query_metadata.execution_time_ms for w in
        "queries_executed": len(successful_widgets),
        "data_freshness": await get_latest_data_date(prospect_id)
    }
}

# 6. Store in PostgreSQL (even with partial failures)
await db.upsert_prospect_dashboard_data(
    prospect_id=prospect_id,
    template_id=template_id,
    data=dashboard_data
)

return dashboard_data
```

```python
async def generate_widget_data(
    widget_config: WidgetConfiguration,
    prospect: Prospect,
    analytics_team: AnalyticsTeamCoordinator
) -> WidgetData:
    """
    Generate data for a single widget.

    Process:
    1. Route to appropriate specialist agent based on data requirements
    2. Agent formulates ClickHouse query
    3. QuestionToSQLAgent validates and executes query
    4. Format results according to widget type
    """

    start_time = time.time()

    # 1. Determine which specialist agent to use
    specialist = analytics_team.route_to_specialist(widget_config.data_requirements)

    # 2. Generate analytics question with context
    enriched_question = f"""
    Prospect: {prospect.name}
    Data available: {prospect.data_date_range}

    Widget: {widget_config.title}
    Question: {widget_config.analytics_question}

    Requirements:
    {json.dumps(widget_config.data_requirements, indent=2)}
    """

    # 3. Specialist generates query
    query_plan = await specialist.analyze_and_query(
        question=enriched_question,
        prospect_id=prospect.id
    )

    # 4. Execute query via SQL agent
    query_result = await analytics_team.sql_agent.execute_query(
        query=query_plan.sql,
        prospect_id=prospect.id
    )
```

```
# 5. Format results for widget type
formatted_data = format_data_for_widget(
    raw_data=query_result.rows,
    widget_type=widget_config.widget_type,
    formatting_rules=widget_config.formatting
)

execution_time = int((time.time() - start_time) * 1000)

return WidgetData(
    widget_id=widget_config.widget_id,
    data_points=formatted_data,
    query_metadata=QueryMetadata(
        query_used=query_plan.sql,
        execution_time_ms=execution_time,
        row_count=len(query_result.rows),
        generated_at=datetime.utcnow().isoformat()
    )
)
```

3.2 Agent Routing Logic

```python
class AnalyticsTeamCoordinator:
    """Routes widget data requirements to appropriate specialist agents."""

    def route_to_specialist(self, data_req: DataRequirement) -> SpecialistAgent:
        """
        Route based on metrics and dimensions in data requirement.

        Examples:
        - HbA1c, glucose, complications → ClinicalAgent
        - ED visits, admissions, LOS → UtilizationAgent
        - Claims costs, PMPM, ROI → PricingAgent
        - Prescriptions, adherence, GDR → PharmacyAgent
        """

        # Check metric names for keywords
        metric_names = " ".join([m.name for m in data_req.metrics]).lower()

        if any(term in metric_names for term in [
            "hba1c", "glucose", "bp", "cholesterol", "bmi", "complication"
        ]):
            return self.clinical_agent

        elif any(term in metric_names for term in [
            "admission", "ed_visit", "emergency", "hospitalization", "length_of_stay"
        ]):
            return self.utilization_agent

        elif any(term in metric_names for term in [
            "cost", "pmpm", "roi", "savings", "paid_amount", "allowed"
        ]):
            return self.pricing_agent

        elif any(term in metric_names for term in [
            "prescription", "rx", "pharmacy", "adherence", "pdc", "gdr"
        ]):
            return self.pharmacy_agent

        else:
            # Default to pricing for general financial metrics
            return self.pricing_agent
```

3.3 Partial Failure Handling

Strategy: Store partial results and allow individual widget retry.

Design Decisions:

1. **Store Partial Results** ✅
   - Dashboard data is stored even if some widgets fail
   - Failed widgets are tracked in `metadata.failed_widgets`
   - Successful widgets render normally
   - Failed widgets show error state in UI
2. **Error Classification**

```
def classify_error(error: Exception) -> str:
    """Classify error for better handling and retry logic."""
    if isinstance(error, QueryTimeoutError):
        return "timeout"
    elif isinstance(error, ValidationError):
        return "validation_error"
    elif isinstance(error, DataNotFoundError):
        return "data_not_found"
    else:
        return "query_error"
```

3. **Retry Logic**
   - Automatic retry: 3 attempts with exponential backoff
   - Manual retry: API endpoint to regenerate specific widget
   - Full regeneration: API endpoint to regenerate entire dashboard
4. **API Endpoints for Retry**

```
# Retry single widget
POST /api/v1/prospects/{prospect_id}/analytics/dashboards/{template_id}/widgets/{wi

# Retry all failed widgets
POST /api/v1/prospects/{prospect_id}/analytics/dashboards/{template_id}/retry-faile

# Full regeneration
POST /api/v1/prospects/{prospect_id}/analytics/dashboards/{template_id}/regenerate
```

5. **UI Error States**

```
interface WidgetErrorDisplay {
  widget_id: string
  error_message: string
  error_type: string
  actions: [
    { label: "Retry", endpoint: "/regenerate" },
    { label: "View Query", action: "show_debug_info" },
    { label: "Skip Widget", action: "mark_as_ignored" }
  ]
}
```

Recommended Approach:

- **Timeouts**: Retry up to 3 times, then fail gracefully
- **Query Errors**: Log for debugging, allow manual retry
- **Data Not Found**: Show helpful message, suggest data upload
- **Validation Errors**: Alert developers (template issue)

4. Runtime Merging & Rendering

4.1 API Endpoint Design

```python
@router.get("/prospects/{prospect_id}/analytics/dashboards/{template_id}/data")
async def get_dashboard_render_data(
    prospect_id: str,
    template_id: str,
    force_refresh: bool = False
) -> DashboardRenderJSON:
    """
    Get complete dashboard ready for frontend rendering.

    This endpoint performs the template + data merge and returns
    a single JSON payload that the frontend can directly render.

    Performance target: < 50ms (cached data)
    """

    # 1. Fetch template (client-scoped)
    template = await db.get_dashboard_template(template_id)

    # 2. Fetch or generate prospect data
    if force_refresh:
        # Regenerate data (async job)
        job = await queue.enqueue_job(
            "analytics_dashboard_generation",
            prospect_id=prospect_id,
            template_id=template_id
        )
        raise HTTPException(
            status_code=202,
            detail={"message": "Regenerating dashboard data", "job_id": job.id}
        )

    # Try to fetch cached data
    prospect_data = await db.get_prospect_dashboard_data(
        prospect_id=prospect_id,
        template_id=template_id
    )

    if not prospect_data:
```

```
        # Data doesn't exist - trigger generation
        job = await queue.enqueue_job(
            "analytics_dashboard_generation",
            prospect_id=prospect_id,
            template_id=template_id
        )
        raise HTTPException(
            status_code=404,
            detail={"message": "Dashboard data not ready", "job_id": job.id}
        )

    # 3. Merge template + data
    render_json = merge_template_and_data(template, prospect_data)

    return render_json


def merge_template_and_data(
    template: DashboardTemplate,
    prospect_data: ProspectDashboardData
) -> DashboardRenderJSON:
    """
    Merge template configuration with prospect-specific data.

    This creates a complete JSON structure that contains:
    - Layout and styling from template
    - Widget configurations from template
    - Actual data from prospect_data

    Frontend can render this directly without additional API calls.
    """

    # Create data lookup map
    data_map = {
        wd.widget_id: wd for wd in prospect_data.widgets_data
    }

    # Create failure lookup map
    failure_map = {
        wf.widget_id: wf for wf in prospect_data.metadata.failed_widgets
    }

    # Build render widgets
```

```
    render_widgets = []
    for widget_config in template.widgets:
        widget_data = data_map.get(widget_config.widget_id)
        widget_failure = failure_map.get(widget_config.widget_id)

        render_widget = {
            # Template config
            "id": widget_config.widget_id,
            "type": widget_config.widget_type,
            "title": widget_config.title,
            "description": widget_config.description,
            "position": widget_config.position,
            "chart_config": widget_config.chart_config,

            # Prospect data
            "data": widget_data.data_points if widget_data else [],
            "summary": widget_data.summary_stats if widget_data else None,

            # Status
            "status": "success" if widget_data else "error",
            "error_message": widget_failure.error_message if widget_failure else None,
            "error_type": widget_failure.error_type if widget_failure else None
        }

        render_widgets.append(render_widget)

    return DashboardRenderJSON(
        template=DashboardTemplateSummary(
            id=template.id,
            name=template.name,
            description=template.description,
            category=template.category,
            visual_style=template.visual_style
        ),
        widgets=render_widgets,
        metadata=DashboardRenderMetadata(
            prospect_id=prospect_data.prospect_id,
            generated_at=prospect_data.generated_at,
            data_freshness=prospect_data.metadata.data_freshness,
            cache_status="hit",
            total_widgets=prospect_data.metadata.total_widgets,
            successful_widgets=prospect_data.metadata.successful_widgets,
            failed_widgets_count=len(prospect_data.metadata.failed_widgets)
```

)
)

4.2 Frontend Rendering

```
// Frontend component for rendering dashboards
interface DashboardRenderJSON {
  template: {
    id: string
    name: string
    description: string
    category: string
    visual_style: VisualStyle
  }
  widgets: RenderWidget[]
  metadata: {
    prospect_id: string
    generated_at: string
    data_freshness: string
    cache_status: 'hit' | 'miss' | 'regenerating'
    total_widgets: number
    successful_widgets: number
    failed_widgets_count: number
  }
}

interface RenderWidget {
  id: string
  type: WidgetType
  title: string
  description: string
  position: GridPosition
  chart_config: any
  data: DataPoint[]              // Ready to render
  summary: SummaryStats | null
  status: 'success' | 'error' | 'loading'
  error_message: string | null
  error_type: string | null
}

// React component
const DashboardRenderer: React.FC<{ prospectId: string; templateId: string }> = ({
  prospectId,
  templateId
}) => {
  const [dashboard, setDashboard] = useState<DashboardRenderJSON | null>(null)
```

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

useEffect(() => {
  const fetchDashboard = async () => {
    try {
      const response = await fetch(
        `/api/v1/prospects/${prospectId}/analytics/dashboards/${templateId}/data`
      )

      if (response.status === 202 || response.status === 404) {
        // Data is being generated - poll for completion
        const { job_id } = await response.json()
        pollForCompletion(job_id)
        return
      }

      const data = await response.json()
      setDashboard(data)
    } catch (error) {
      console.error('Failed to load dashboard:', error)
    } finally {
      setLoading(false)
    }
  }

  fetchDashboard()
}, [prospectId, templateId])

if (loading || !dashboard) {
  return <LoadingState />
}

return (
  <DashboardLayout visualStyle={dashboard.template.visual_style}>
    <DashboardHeader template={dashboard.template} metadata={dashboard.metadata} />

    <WidgetGrid>
      {dashboard.widgets.map((widget) => (
        <WidgetRenderer
          key={widget.id}
          widget={widget}
          visualStyle={dashboard.template.visual_style}
        />
```

```
        ))}
      </WidgetGrid>
    </DashboardLayout>
  )
}

// Widget renderer - routes to appropriate component based on Chart.js
const WidgetRenderer: React.FC<{ widget: RenderWidget; visualStyle: VisualStyle }> = ({
  widget,
  visualStyle
}) => {
  if (widget.status === 'error') {
    return <WidgetErrorState message={widget.error_message} type={widget.error_type} />
  }

  // Map widget types to Chart.js components
  switch (widget.type) {
    case 'kpi-card':
      return <KPICard data={widget.data[0]} config={widget.chart_config} color={visualS

    case 'line-chart':
    case 'area-chart':
      return <LineChart data={widget.data} config={widget.chart_config} color={visualSt

    case 'multi-line-chart':
      return <MultiLineChart data={widget.data} config={widget.chart_config} colors={vi

    case 'bar-chart':
    case 'horizontal-bar-chart':
      return <BarChart data={widget.data} config={widget.chart_config} color={visualSty

    case 'waterfall-chart':
      return <WaterfallChart data={widget.data} config={widget.chart_config} color={vis

    case 'pie-chart':
    case 'donut-chart':
      return <PieChart data={widget.data} config={widget.chart_config} />

    case 'timeline-chart':
      return <ROITimelineChart data={widget.data} config={widget.chart_config} color={v

    case 'quality-progression':
      return <QualityProgressionWidget data={widget.data} config={widget.chart_config}
```

```
    case 'ranked-list':
      return <RankedListWidget data={widget.data} config={widget.chart_config} color={v

    case 'data-table':
      return <DataTable data={widget.data} config={widget.chart_config} />

    // ... other widget types

    default:
      return <UnsupportedWidgetType type={widget.type} />
  }
}
```

5. Performance Considerations

5.1 Query Optimization

ClickHouse Query Best Practices:

1. **Use Materialized Views for Common Aggregations**

```
-- Pre-compute monthly diabetes metrics
CREATE MATERIALIZED VIEW diabetes_monthly_metrics
ENGINE = SummingMergeTree()
ORDER BY (prospect_id, month)
AS SELECT
    prospect_id,
    toStartOfMonth(service_date) AS month,
    countIf(diagnosis_code LIKE 'E11%') AS diabetic_claims,
    avgIf(hba1c_value, hba1c_value > 0) AS avg_hba1c,
    sumIf(paid_amount, diagnosis_code LIKE 'E11%') AS total_cost
FROM medical_claims
GROUP BY prospect_id, month
```

2. **Partition by Prospect**

```
-- Partition tables for faster prospect-specific queries
CREATE TABLE medical_claims (
    prospect_id UUID,
    service_date Date,
    ...
)
ENGINE = MergeTree()
PARTITION BY prospect_id
ORDER BY (prospect_id, service_date)
```

3. **Use Query Result Cache**

```
-- Enable query result caching for repeated queries
SET use_query_cache = 1;
SET query_cache_ttl = 3600;
```

5.2 PostgreSQL Optimization

1. **JSONB Indexing**

```
-- Index frequently queried JSONB paths
CREATE INDEX idx_widget_data_roi
ON prospect_dashboard_data ((dashboard_data->'widgets_data'->0->'data_points'->0->>
WHERE (dashboard_data->'widgets_data'->0->>'widget_id') = 'w_roi_percentage';
```

2. **Partial Indexes**

```
-- Index only active/approved templates
CREATE INDEX idx_active_templates
ON dashboard_templates(client_id, category)
WHERE status = 'approved';
```

5.3 Performance Targets

Operation	Target	Notes
Get template (no data)	< 10ms	Single PostgreSQL query
Get dashboard data (cached in PG)	< 20ms	PostgreSQL JSONB query
Generate single widget data	< 500ms	ClickHouse query + formatting
Generate full dashboard (8 widgets)	< 3s	Parallel execution

Operation	Target	Notes
Frontend render (complete JSON)	< 100ms	Client-side only

6. API Reference

6.1 Template Management APIs

Generate Templates (Client Onboarding - Step 1.4)

```
POST /api/v1/clients/{client_id}/dashboard-templates/generate
```

Request:

```
{
  "value_proposition_id": "vp-uuid-123",
  "template_count": 8,
  "categories": ["roi-focused", "clinical-outcomes", "operational-efficiency"]
}
```

Response (202 Accepted):

```
{
  "job_id": "job-uuid-456",
  "status": "pending",
  "estimated_duration_seconds": 120
}
```

List Templates

```
GET /api/v1/clients/{client_id}/dashboard-templates
```

Query Parameters:

- `category` (optional): Filter by category
- `status` (optional): Filter by status

- `target_audience` (optional): Filter by audience

Response (200 OK):

```
{
  "templates": [
    {
      "id": "template-uuid-1",
      "name": "Diabetes ROI Executive Dashboard",
      "category": "roi-focused",
      "target_audience": "Broker",
      "widget_count": 8,
      "status": "approved",
      "created_at": "2025-01-15T10:30:00Z"
    }
  ],
  "total": 8
}
```

Get Template Details

```
GET /api/v1/clients/{client_id}/dashboard-templates/{template_id}
```

Response (200 OK):

```
{
  "id": "template-uuid-1",
  "client_id": "client-abc-123",
  "name": "Diabetes ROI Executive Dashboard",
  "description": "...",
  "category": "roi-focused",
  "target_audience": "Broker",
  "visual_style": { ... },
  "widgets": [ ... ],
  "metadata": { ... },
  "status": "approved",
  "created_at": "2025-01-15T10:30:00Z",
  "updated_at": "2025-01-15T14:20:00Z"
}
```

Approve Template

```
PATCH /api/v1/clients/{client_id}/dashboard-templates/{template_id}/approve
```

Response (200 OK):

```
{
  "id": "template-uuid-1",
  "status": "approved",
  "approved_at": "2025-01-15T14:20:00Z"
}
```

Update Template (Full Replacement)

```
PUT /api/v1/clients/{client_id}/dashboard-templates/{template_id}
```

Request:

```
{
  "name": "Updated Dashboard Name",
  "description": "Updated description",
  "widgets": [ ... ]  // Complete widget array
}
```

Response (200 OK):

```
{
  "id": "template-uuid-1",
  "status": "generated",
  "updated_at": "2025-01-16T10:00:00Z",
  "message": "Template updated. Existing prospect data will use previous template until
}
```

6.2 Dashboard Data APIs

Generate Dashboard Data (Prospect Processing - Step 5)

```
POST /api/v1/prospects/{prospect_id}/analytics/dashboards/generate
```

Request:

```
{
  "template_ids": ["template-uuid-1", "template-uuid-2"],
  "force_regenerate": false
}
```

Response (202 Accepted):

```
{
  "job_id": "job-uuid-789",
  "status": "pending",
  "templates_count": 2,
  "estimated_duration_seconds": 180
}
```

Get Dashboard Render Data

```
GET /api/v1/prospects/{prospect_id}/analytics/dashboards/{template_id}/data
```

Query Parameters:

- `force_refresh` (optional): Regenerate data even if cached

Response (200 OK):

```
{
  "template": {
    "id": "template-uuid-1",
    "name": "Diabetes ROI Executive Dashboard",
    "category": "roi-focused",
    "visual_style": { ... }
  },
  "widgets": [
    {
      "id": "w_roi_percentage",
      "type": "kpi-card",
      "title": "Total ROI",
      "position": { ... },
      "data": [ { "value": 245.7, "formatted_value": "245.7%" } ],
      "status": "success"
    }
  ],
  "metadata": {
    "prospect_id": "prospect-xyz-789",
    "generated_at": "2025-01-15T15:30:00Z",
    "data_freshness": "2024-12-31",
    "cache_status": "hit",
    "total_widgets": 8,
    "successful_widgets": 7,
    "failed_widgets_count": 1
  }
}
```


Response (404 Not Found) - Data not ready:


```
{
  "message": "Dashboard data not ready",
  "job_id": "job-uuid-789",
  "status": "processing",
  "progress": 45
}
```


Retry Failed Widgets

```
POST /api/v1/prospects/{prospect_id}/analytics/dashboards/{template_id}/retry-failed
```

Response (202 Accepted):

```
{
  "job_id": "job-uuid-890",
  "widgets_to_retry": ["w_advanced_analytics"],
  "status": "pending"
}
```

List Available Dashboards for Prospect

```
GET /api/v1/prospects/{prospect_id}/analytics/dashboards
```

Response (200 OK):

```
{
  "dashboards": [
    {
      "template_id": "template-uuid-1",
      "name": "Diabetes ROI Executive Dashboard",
      "category": "roi-focused",
      "data_status": "ready",
      "generated_at": "2025-01-15T15:30:00Z",
      "successful_widgets": 7,
      "failed_widgets": 1,
      "preview_url": "/analytics/dashboards/template-uuid-1/data"
    }
  ],
  "total": 5
}
```

7. Next Steps

7.1 Immediate Actions (Week 1)

1. **Review & Approve Architecture**
   - Stakeholder review of this document
   - Lock schema definitions

2. **Create Database Migrations**

```
migrations/
├── 001_create_dashboard_templates.sql
└── 002_create_prospect_dashboard_data.sql
```

3. **Define Pydantic Schemas**

```
backend/schemas/
├── dashboard_template.py
├── widget_configuration.py
├── data_requirement.py
└── dashboard_data.py
```

4. **Create Test Fixtures**

```
tests/fixtures/dashboard_templates/
├── diabetes_roi_executive.json
├── clinical_outcomes.json
└── invalid_template.json
```

7.2 Implementation Sequence

Week 1: Foundation

- [ ] Database schema migration
- [ ] Pydantic schema definitions
- [ ] Basic CRUD APIs for templates
- [ ] Template validation tests

Week 2: Template Generation

- [ ] TemplateGeneratorAgent implementation
- [ ] Template generation API
- [ ] LLM prompt engineering for templates
- [ ] Widget data requirement parsing

Week 3: Data Generation

- [ ] Analytics Team routing logic
- [ ] Widget data generation
- [ ] ClickHouse query generation

- [ ] Data formatting and storage

Week 4: Runtime Merging

- [ ] Dashboard render API
- [ ] Template + data merge logic
- [ ] Frontend integration

Week 5: Polish

- [ ] Performance optimization
- [ ] Error handling and retries
- [ ] Monitoring and logging
- [ ] Documentation

Appendix A: Complete Dashboard Example - "Three-Scenario ROI Comparison"

A.1 Dashboard Overview

Name: Three-Scenario ROI Comparison
Category: roi-focused
Target Audience: Health Plan
Description: Side-by-side comparison of conservative, moderate, and optimistic ROI scenarios with clinical assumptions and outcomes for each scenario

Visual Style:

- Primary Color: `#c62828` (Red)
- Accent Color: `#6a1b9a` (Purple)
- Layout: Balanced

Key Features:

1. Conservative scenario: 2.6x ROI, $2,828 savings
2. Moderate scenario: 3.4x ROI, $3,624 savings
3. Optimistic scenario: 4.1x ROI, $4,477 savings
4. Clinical assumptions and outcomes for each scenario

A.2 Complete Template JSON


```
{
  "id": "scenario-comparison-template-001",
  "client_id": "client-livongo-diabetes",
  "name": "Three-Scenario ROI Comparison",
  "description": "Side-by-side comparison of conservative, moderate, and optimistic ROI
  "category": "roi-focused",
  "target_audience": "Health Plan",
  "visual_style": {
    "primary_color": "#c62828",
    "accent_color": "#6a1b9a",
    "layout": "balanced"
  },
  "widgets": [
    {
      "widget_id": "w_scenario_1_roi",
      "widget_type": "kpi-card",
      "title": "Conservative Scenario ROI",
      "description": "Conservative ROI estimate based on minimal adoption",
      "position": { "row": 1, "col": 1, "row_span": 1, "col_span": 1 },
      "size": { "min_width": 1, "min_height": 1 },
      "data_requirements": {
        "query_type": "aggregate",
        "metrics": [
          {
            "name": "roi_conservative",
            "expression": "(SUM(savings_conservative) / SUM(program_costs) - 1) * 100",
            "data_type": "percentage",
            "format": "0.0x"
          }
        ],
        "filters": [
          { "field": "scenario", "operator": "eq", "value": "conservative" }
        ]
      },
      "analytics_question": "What is the ROI for the conservative scenario?",
      "chart_config": {
        "icon": "tabler-trending-up",
        "show_trend": false,
        "subtitle": "2.6x ROI"
      }
    },
```

```
{
  "widget_id": "w_scenario_2_roi",
  "widget_type": "kpi-card",
  "title": "Moderate Scenario ROI",
  "description": "Moderate ROI estimate based on typical adoption",
  "position": { "row": 1, "col": 2, "row_span": 1, "col_span": 1 },
  "size": { "min_width": 1, "min_height": 1 },
  "data_requirements": {
    "query_type": "aggregate",
    "metrics": [
      {
        "name": "roi_moderate",
        "expression": "(SUM(savings_moderate) / SUM(program_costs) - 1) * 100",
        "data_type": "percentage",
        "format": "0.0x"
      }
    ],
    "filters": [
      { "field": "scenario", "operator": "eq", "value": "moderate" }
    ]
  },
  "analytics_question": "What is the ROI for the moderate scenario?",
  "chart_config": {
    "icon": "tabler-trending-up",
    "show_trend": false,
    "subtitle": "3.4x ROI"
  }
},
{
  "widget_id": "w_scenario_3_roi",
  "widget_type": "kpi-card",
  "title": "Optimistic Scenario ROI",
  "description": "Optimistic ROI estimate based on high adoption",
  "position": { "row": 1, "col": 3, "row_span": 1, "col_span": 1 },
  "size": { "min_width": 1, "min_height": 1 },
  "data_requirements": {
    "query_type": "aggregate",
    "metrics": [
      {
        "name": "roi_optimistic",
        "expression": "(SUM(savings_optimistic) / SUM(program_costs) - 1) * 100",
        "data_type": "percentage",
        "format": "0.0x"
```

```
      }
    ],
    "filters": [
      { "field": "scenario", "operator": "eq", "value": "optimistic" }
    ]
  },
  "analytics_question": "What is the ROI for the optimistic scenario?",
  "chart_config": {
    "icon": "tabler-trending-up",
    "show_trend": false,
    "subtitle": "4.1x ROI"
  }
},
{
  "widget_id": "w_scenario_1_savings",
  "widget_type": "kpi-card",
  "title": "Conservative Savings per Member",
  "description": "3-year savings per member (conservative)",
  "position": { "row": 2, "col": 1, "row_span": 1, "col_span": 1 },
  "data_requirements": {
    "query_type": "aggregate",
    "metrics": [
      {
        "name": "savings_per_member",
        "expression": "SUM(total_savings) / COUNT(DISTINCT member_id)",
        "data_type": "currency",
        "format": "$0,0"
      }
    ],
    "filters": [
      { "field": "scenario", "operator": "eq", "value": "conservative" }
    ],
    "time_range": { "type": "relative", "value": "last_36_months" }
  },
  "analytics_question": "What are the 3-year savings per member for conservative sc
  "chart_config": {
    "icon": "tabler-coin",
    "subtitle": "$2,828 over 3 years"
  }
},
{
  "widget_id": "w_scenario_2_savings",
  "widget_type": "kpi-card",
```

```
  "title": "Moderate Savings per Member",
  "description": "3-year savings per member (moderate)",
  "position": { "row": 2, "col": 2, "row_span": 1, "col_span": 1 },
  "data_requirements": {
    "query_type": "aggregate",
    "metrics": [
      {
        "name": "savings_per_member",
        "expression": "SUM(total_savings) / COUNT(DISTINCT member_id)",
        "data_type": "currency",
        "format": "$0,0"
      }
    ],
    "filters": [
      { "field": "scenario", "operator": "eq", "value": "moderate" }
    ],
    "time_range": { "type": "relative", "value": "last_36_months" }
  },
  "analytics_question": "What are the 3-year savings per member for moderate scenar
  "chart_config": {
    "icon": "tabler-coin",
    "subtitle": "$3,624 over 3 years"
  }
},
{
  "widget_id": "w_scenario_3_savings",
  "widget_type": "kpi-card",
  "title": "Optimistic Savings per Member",
  "description": "3-year savings per member (optimistic)",
  "position": { "row": 2, "col": 3, "row_span": 1, "col_span": 1 },
  "data_requirements": {
    "query_type": "aggregate",
    "metrics": [
      {
        "name": "savings_per_member",
        "expression": "SUM(total_savings) / COUNT(DISTINCT member_id)",
        "data_type": "currency",
        "format": "$0,0"
      }
    ],
    "filters": [
      { "field": "scenario", "operator": "eq", "value": "optimistic" }
    ],
```

```
      "time_range": { "type": "relative", "value": "last_36_months" }
    },
    "analytics_question": "What are the 3-year savings per member for optimistic scen
    "chart_config": {
      "icon": "tabler-coin",
      "subtitle": "$4,477 over 3 years"
    }
  },
  {
    "widget_id": "w_roi_comparison_chart",
    "widget_type": "grouped-bar-chart",
    "title": "ROI Comparison Across Scenarios",
    "description": "Visual comparison of ROI, savings, and payback period across all
    "position": { "row": 3, "col": 1, "row_span": 2, "col_span": 3 },
    "size": { "min_height": 2 },
    "data_requirements": {
      "query_type": "comparison",
      "dimensions": ["scenario", "metric_type"],
      "metrics": [
        {
          "name": "metric_value",
          "expression": "CASE WHEN metric_type = 'ROI' THEN roi_multiplier WHEN metri
          "data_type": "number"
        }
      ],
      "filters": [
        { "field": "scenario", "operator": "in", "value": ["conservative", "moderate"
      ]
    },
    "analytics_question": "What are the ROI, savings per member, and payback period f
    "chart_config": {
      "chart_type": "bar",
      "x_axis": "scenario",
      "y_axis": "metric_value",
      "group_by": "metric_type",
      "color": ["#c62828", "#6a1b9a", "#00897b"],
      "show_legend": true,
      "show_grid": true,
      "stacked": false
    }
  },
  {
    "widget_id": "w_clinical_assumptions",
```

```
      "widget_type": "data-table",
      "title": "Clinical Assumptions by Scenario",
      "description": "Key clinical assumptions driving each ROI scenario",
      "position": { "row": 5, "col": 1, "row_span": 2, "col_span": 3 },
      "data_requirements": {
        "query_type": "distribution",
        "dimensions": ["scenario", "assumption_category"],
        "metrics": [
          {
            "name": "assumption_value",
            "expression": "assumption_value",
            "data_type": "percentage"
          }
        ]
      },
      "analytics_question": "What are the clinical assumptions for each scenario?",
      "chart_config": {
        "columns": [
          { "field": "assumption_category", "header": "Clinical Metric" },
          { "field": "conservative_value", "header": "Conservative" },
          { "field": "moderate_value", "header": "Moderate" },
          { "field": "optimistic_value", "header": "Optimistic" }
        ],
        "sortable": true
      }
    }
  ],
  "metadata": {
    "generated_by": "llm",
    "llm_model": "claude-3-5-sonnet-20241022",
    "generation_timestamp": "2025-01-15T10:30:00Z",
    "key_features": [
      "Conservative scenario: 2.6x ROI, $2,828 savings",
      "Moderate scenario: 3.4x ROI, $3,624 savings",
      "Optimistic scenario: 4.1x ROI, $4,477 savings",
      "Clinical assumptions and outcomes for each scenario"
    ],
    "recommended_use_case": "Best for: Financial planning and risk assessment"
  },
  "status": "approved",
  "created_at": "2025-01-15T10:30:00Z",
  "updated_at": "2025-01-15T14:20:00Z"
}
```

A.3 Complete Dashboard Data JSON (Prospect-Specific)


```
{
  "prospect_id": "prospect-wellmark-001",
  "template_id": "scenario-comparison-template-001",
  "generated_at": "2025-01-15T16:00:00Z",
  "widgets_data": [
    {
      "widget_id": "w_scenario_1_roi",
      "data_points": [
        {
          "value": 2.6,
          "formatted_value": "2.6x",
          "metadata": {
            "total_savings": 14200000,
            "program_costs": 5500000,
            "scenario": "conservative"
          }
        }
      ],
      "query_metadata": {
        "query_used": "SELECT (SUM(savings_conservative) / SUM(program_costs) - 1) * 10
        "execution_time_ms": 125,
        "row_count": 1,
        "generated_at": "2025-01-15T16:00:05Z"
      }
    },
    {
      "widget_id": "w_scenario_2_roi",
      "data_points": [
        {
          "value": 3.4,
          "formatted_value": "3.4x",
          "metadata": {
            "total_savings": 18700000,
            "program_costs": 5500000,
            "scenario": "moderate"
          }
        }
      ],
      "query_metadata": {
        "query_used": "SELECT (SUM(savings_moderate) / SUM(program_costs) - 1) * 100 AS
        "execution_time_ms": 130,
```

```
    "row_count": 1,
    "generated_at": "2025-01-15T16:00:06Z"
  }
},
{
  "widget_id": "w_scenario_3_roi",
  "data_points": [
    {
      "value": 4.1,
      "formatted_value": "4.1x",
      "metadata": {
        "total_savings": 22500000,
        "program_costs": 5500000,
        "scenario": "optimistic"
      }
    }
  ],
  "query_metadata": {
    "query_used": "SELECT (SUM(savings_optimistic) / SUM(program_costs) - 1) * 100 
    "execution_time_ms": 128,
    "row_count": 1,
    "generated_at": "2025-01-15T16:00:07Z"
  }
},
{
  "widget_id": "w_scenario_1_savings",
  "data_points": [
    {
      "value": 2828,
      "formatted_value": "$2,828",
      "metadata": {
        "time_period": "36 months",
        "member_count": 5024
      }
    }
  ],
  "query_metadata": {
    "query_used": "SELECT SUM(total_savings) / COUNT(DISTINCT member_id) AS savings
    "execution_time_ms": 180,
    "row_count": 1,
    "generated_at": "2025-01-15T16:00:10Z"
  }
},
```

```
{
  "widget_id": "w_scenario_2_savings",
  "data_points": [
    {
      "value": 3624,
      "formatted_value": "$3,624",
      "metadata": {
        "time_period": "36 months",
        "member_count": 5160
      }
    }
  ],
  "query_metadata": {
    "query_used": "SELECT SUM(total_savings) / COUNT(DISTINCT member_id) AS savings
    "execution_time_ms": 185,
    "row_count": 1,
    "generated_at": "2025-01-15T16:00:11Z"
  }
},
{
  "widget_id": "w_scenario_3_savings",
  "data_points": [
    {
      "value": 4477,
      "formatted_value": "$4,477",
      "metadata": {
        "time_period": "36 months",
        "member_count": 5025
      }
    }
  ],
  "query_metadata": {
    "query_used": "SELECT SUM(total_savings) / COUNT(DISTINCT member_id) AS savings
    "execution_time_ms": 182,
    "row_count": 1,
    "generated_at": "2025-01-15T16:00:12Z"
  }
},
{
  "widget_id": "w_roi_comparison_chart",
  "data_points": [
    {
      "label": "Conservative",
```

```
  "value": 2.6,
  "formatted_value": "2.6x",
  "metadata": { "metric_type": "ROI", "scenario": "conservative" }
},
{
  "label": "Conservative",
  "value": 2.828,
  "formatted_value": "$2.8K",
  "metadata": { "metric_type": "Savings", "scenario": "conservative" }
},
{
  "label": "Conservative",
  "value": 18,
  "formatted_value": "18 mo",
  "metadata": { "metric_type": "Payback", "scenario": "conservative" }
},
{
  "label": "Moderate",
  "value": 3.4,
  "formatted_value": "3.4x",
  "metadata": { "metric_type": "ROI", "scenario": "moderate" }
},
{
  "label": "Moderate",
  "value": 3.624,
  "formatted_value": "$3.6K",
  "metadata": { "metric_type": "Savings", "scenario": "moderate" }
},
{
  "label": "Moderate",
  "value": 15,
  "formatted_value": "15 mo",
  "metadata": { "metric_type": "Payback", "scenario": "moderate" }
},
{
  "label": "Optimistic",
  "value": 4.1,
  "formatted_value": "4.1x",
  "metadata": { "metric_type": "ROI", "scenario": "optimistic" }
},
{
  "label": "Optimistic",
  "value": 4.477,
```

```
      "formatted_value": "$4.5K",
      "metadata": { "metric_type": "Savings", "scenario": "optimistic" }
    },
    {
      "label": "Optimistic",
      "value": 12,
      "formatted_value": "12 mo",
      "metadata": { "metric_type": "Payback", "scenario": "optimistic" }
    }
  ],
  "summary_stats": {
    "min": 2.6,
    "max": 4.477,
    "avg": 3.037,
    "count": 9
  },
  "query_metadata": {
    "query_used": "SELECT scenario, metric_type, CASE WHEN metric_type = 'ROI'...",
    "execution_time_ms": 245,
    "row_count": 9,
    "generated_at": "2025-01-15T16:00:15Z"
  }
},
{
  "widget_id": "w_clinical_assumptions",
  "data_points": [
    {
      "label": "HbA1c Reduction",
      "value": 0.5,
      "formatted_value": "0.5%",
      "metadata": { "scenario": "conservative", "assumption": "hba1c_reduction" }
    },
    {
      "label": "HbA1c Reduction",
      "value": 0.8,
      "formatted_value": "0.8%",
      "metadata": { "scenario": "moderate", "assumption": "hba1c_reduction" }
    },
    {
      "label": "HbA1c Reduction",
      "value": 1.2,
      "formatted_value": "1.2%",
      "metadata": { "scenario": "optimistic", "assumption": "hba1c_reduction" }
```

```
    },
    {
      "label": "Member Engagement",
      "value": 45,
      "formatted_value": "45%",
      "metadata": { "scenario": "conservative", "assumption": "engagement_rate" }
    },
    {
      "label": "Member Engagement",
      "value": 65,
      "formatted_value": "65%",
      "metadata": { "scenario": "moderate", "assumption": "engagement_rate" }
    },
    {
      "label": "Member Engagement",
      "value": 80,
      "formatted_value": "80%",
      "metadata": { "scenario": "optimistic", "assumption": "engagement_rate" }
    },
    {
      "label": "Complication Reduction",
      "value": 25,
      "formatted_value": "25%",
      "metadata": { "scenario": "conservative", "assumption": "complication_reducti
    },
    {
      "label": "Complication Reduction",
      "value": 43,
      "formatted_value": "43%",
      "metadata": { "scenario": "moderate", "assumption": "complication_reduction"
    },
    {
      "label": "Complication Reduction",
      "value": 68,
      "formatted_value": "68%",
      "metadata": { "scenario": "optimistic", "assumption": "complication_reduction
    }
  ],
  "query_metadata": {
    "query_used": "SELECT scenario, assumption_category, assumption_value...",
    "execution_time_ms": 165,
    "row_count": 9,
    "generated_at": "2025-01-15T16:00:18Z"
```

```
      }
    }
  ],
  "metadata": {
    "total_widgets": 8,
    "successful_widgets": 8,
    "failed_widgets": [],
    "total_execution_time_ms": 1520,
    "queries_executed": 8,
    "data_freshness": "2024-12-31"
  }
}
```


A.4 Rendered Dashboard Preview

Layout Grid (3 columns x 7 rows):

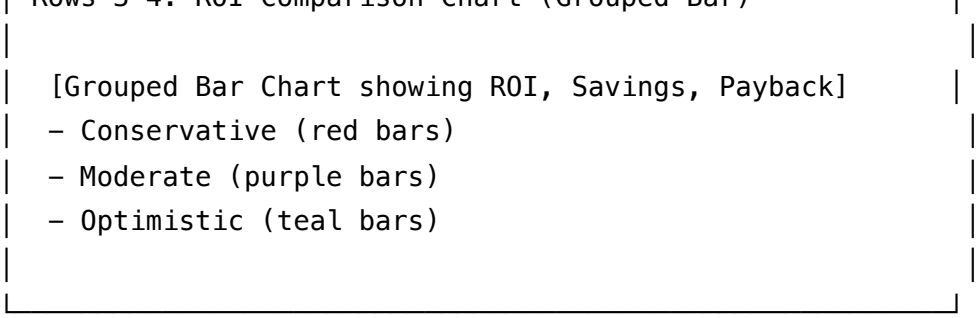
Row 1: Scenario ROI KPIs

Conservative ROI	Moderate ROI	Optimistic ROI
2.6x	3.4x	4.1x

Row 2: Savings per Member KPIs

Conservative	Moderate	Optimistic
$2,828 (3-year savings)	$3,624 (3-year savings)	$4,477 (3-year savings)

Rows 3-4: ROI Comparison Chart (Grouped Bar)



Rows 5-6: Clinical Assumptions Table

Clinical Metric	Conservative	Moderate	Optimistic
HbA1c Reduction	0.5%	0.8%	1.2%
Member Engagement	45%	65%	80%
Complication Reduc.	25%	43%	68%

Appendix B: Chart.js Widget Implementation

Examples

B.1 KPICard Component

```
// Used for widget_type: 'kpi-card'
import { Card, Typography, Box } from '@mui/material'
import { Chart } from 'chart.js'

interface KPICardProps {
  data: DataPoint
  config: ChartConfig
  color: string
}

export const KPICard: React.FC<KPICardProps> = ({ data, config, color }) => {
  return (
    <Card sx={{ p: 3, height: '100%', border: '1px solid #e0e0e0' }}>
      <Box sx={{ display: 'flex', justifyContent: 'space-between' }}>
        <Typography variant="body2" color="text.secondary">
          {config.title}
        </Typography>
        {config.icon && (
          <Box sx={{
            width: 40,
            height: 40,
            borderRadius: 2,
            bgcolor: color + '15',
            display: 'flex',
            alignItems: 'center',
            justifyContent: 'center'
          }}>
            <i className={config.icon} style={{ fontSize: 20, color }} />
          </Box>
        )}
      </Box>

      <Typography variant="h4" fontWeight="bold" color={color} sx={{ mt: 2 }}>
        {data.formatted_value}
      </Typography>

      {config.subtitle && (
        <Typography variant="body2" color="text.secondary" sx={{ mt: 1 }}>
```

```
          {config.subtitle}
        </Typography>
      )}
    </Card>
  )
}
```

B.2 MultiLineChart Component

```
// Used for widget_type: 'multi-line-chart'
import { useEffect, useRef } from 'react'
import { Card, Typography, Box } from '@mui/material'
import { Chart, registerables } from 'chart.js'

Chart.register(...registerables)

interface MultiLineChartProps {
  data: DataPoint[]
  config: ChartConfig
  colors: VisualStyle
}

export const MultiLineChart: React.FC<MultiLineChartProps> = ({ data, config, colors })
  const chartRef = useRef<HTMLCanvasElement>(null)
  const chartInstance = useRef<Chart | null>(null)

  useEffect(() => {
    if (!chartRef.current) return

    // Destroy existing chart
    if (chartInstance.current) {
      chartInstance.current.destroy()
    }

    const ctx = chartRef.current.getContext('2d')
    if (!ctx) return

    // Group data by series
    const seriesMap = new Map<string, DataPoint[]>()
    data.forEach(point => {
      const series = point.metadata?.series || 'default'
      if (!seriesMap.has(series)) {
        seriesMap.set(series, [])
      }
      seriesMap.get(series)!.push(point)
    })

    // Create datasets
    const datasets = Array.from(seriesMap.entries()).map(([series, points], idx) => ({
      label: series,
```

```
    data: points.map(p => ({ x: p.label, y: p.value })),
    borderColor: Array.isArray(config.color) ? config.color[idx] : colors.primary_col
    backgroundColor: (Array.isArray(config.color) ? config.color[idx] : colors.primar
    borderWidth: config.border_width || 3,
    tension: config.tension || 0.4,
    fill: config.fill || true,
    pointRadius: config.point_radius || 4
  }))

  chartInstance.current = new Chart(ctx, {
    type: 'line',
    data: {
      datasets
    },
    options: {
      responsive: true,
      maintainAspectRatio: false,
      plugins: {
        legend: {
          display: config.show_legend !== false,
          position: 'bottom'
        },
        tooltip: {
          mode: 'index',
          intersect: false
        }
      },
      scales: {
        y: {
          beginAtZero: true,
          grid: {
            display: config.show_grid !== false
          }
        },
        x: {
          grid: {
            display: false
          }
        }
      }
    }
  })
```

```
    return () => {
      if (chartInstance.current) {
        chartInstance.current.destroy()
      }
    }
  }, [data, config, colors])

  return (
    <Card sx={{ p: 3, height: '100%', border: '1px solid #e0e0e0' }}>
      <Typography variant="h6" gutterBottom>
        {config.title}
      </Typography>
      <Typography variant="body2" color="text.secondary" paragraph>
        {config.description}
      </Typography>
      <Box sx={{ height: 300 }}>
        <canvas ref={chartRef} />
      </Box>
    </Card>
  )
}
```

B.3 WaterfallChart Component

```
// Used for widget_type: 'waterfall-chart'
import { useEffect, useRef } from 'react'
import { Card, Typography, Box } from '@mui/material'
import { Chart, registerables } from 'chart.js'

Chart.register(...registerables)

interface WaterfallChartProps {
  data: DataPoint[]
  config: ChartConfig
  color: string
}

export const WaterfallChart: React.FC<WaterfallChartProps> = ({ data, config, color })
  const chartRef = useRef<HTMLCanvasElement>(null)
  const chartInstance = useRef<Chart | null>(null)

  useEffect(() => {
    if (!chartRef.current) return

    if (chartInstance.current) {
      chartInstance.current.destroy()
    }

    const ctx = chartRef.current.getContext('2d')
    if (!ctx) return

    const totalSavings = data.reduce((sum, d) => sum + d.value, 0)

    chartInstance.current = new Chart(ctx, {
      type: 'bar',
      data: {
        labels: data.map(d => d.label),
        datasets: [{
          label: config.y_axis || 'Annual Savings',
          data: data.map(d => d.value),
          backgroundColor: data.map((_, idx) => {
            const opacity = 0.8 - (idx * 0.15)
            return color + Math.round(opacity * 255).toString(16).padStart(2, '0')
          }),
          borderRadius: 6
```

```
      }]
    },
    options: {
      responsive: true,
      maintainAspectRatio: false,
      plugins: {
        legend: { display: false },
        tooltip: {
          callbacks: {
            label: (context) => {
              const value = context.parsed.y
              const percentage = ((value / totalSavings) * 100).toFixed(1)
              return [
                `Savings: ${(value / 1000000).toFixed(1)}M`,
                `Contribution: ${percentage}% of total`
              ]
            }
          }
        }
      },
      scales: {
        y: {
          beginAtZero: true,
          ticks: {
            callback: (value) => '$' + (Number(value) / 1000000).toFixed(1) + 'M'
          }
        }
      }
    }
  })

  return () => {
    if (chartInstance.current) {
      chartInstance.current.destroy()
    }
  }
}, [data, config, color])

return (
  <Card sx={{ p: 3, height: '100%', border: '1px solid #e0e0e0' }}>
    <Typography variant="h6" gutterBottom>
      {config.title}
    </Typography>
```

```
        <Typography variant="body2" color="text.secondary" paragraph>
          {config.description}
        </Typography>
        <Box sx={{ height: 300 }}>
          <canvas ref={chartRef} />
        </Box>
      </Card>
    )
}
```

Document Revision History

Version	Date	Author	Changes
1.0	2025-01-17	System	Initial architecture document
1.1	2025-01-17	System	Removed migration section, added Chart.js widget catalog, added Three-Scenario dashboard example, updated partial failure handling