# Dashboard Creator - Technical & UI Design Document

## Executive Summary

The Dashboard Creator is a drag-and-drop wizard application that allows users to build custom analytics dashboards by selecting predefined metrics and arranging them on a configurable grid canvas. The application features a multi-step wizard interface with real-time preview capabilities and persistent storage.

## Architecture Overview

### Frontend Architecture

* **Framework**: React 18 with TypeScript
* **State Management**: TanStack Query for server state, React hooks for local state
* **UI Components**: Shadcn/ui built on Radix UI primitives
* **Styling**: Tailwind CSS with custom CSS variables
* **Routing**: Wouter for lightweight client-side routing
* **Build Tool**: Vite for development and production builds

### Backend Architecture

* **Runtime**: Node.js with Express.js
* **Language**: TypeScript with ES modules
* **Data Layer**: In-memory storage (development) / PostgreSQL (production)
* **ORM**: Drizzle ORM for type-safe database operations
* **API Design**: RESTful endpoints with Zod validation

## Complete Tech Stack

### Core Technologies

#### Frontend Stack

* **React 18.3.1**: Modern React with hooks, concurrent features, and TypeScript support
* **TypeScript 5.6.3**: Static type checking and enhanced developer experience
* **Vite 6.0.3**: Lightning-fast build tool with Hot Module Replacement (HMR)
* **Wouter 3.3.5**: Minimalist routing library (~2.8KB) for client-side navigation

#### Backend Stack

* **Node.js**: JavaScript runtime environment
* **Express 4.21.2**: Fast, minimalist web framework for Node.js
* **TypeScript (tsx)**: Runtime TypeScript execution with hot reload
* **Drizzle ORM**: Type-safe database toolkit with excellent TypeScript integration

#### Database & Storage

* **PostgreSQL**: Production database with JSONB support for flexible widget storage
* **In-Memory Storage**: Development fallback using Map-based storage for rapid prototyping
* **Session Storage**: PostgreSQL-based session persistence using connect-pg-simple

### UI Framework & Components

#### Design System Foundation

* **Shadcn/ui**: High-quality component library built on Radix UI primitives
* **Radix UI**: Unstyled, accessible component primitives
  + @radix-ui/react-dialog: Modal dialogs for preview and configuration
  + @radix-ui/react-dropdown-menu: Action menus for widgets
  + @radix-ui/react-toast: Notification system
  + @radix-ui/react-tabs: Tabbed interfaces
  + @radix-ui/react-accordion: Collapsible content sections
  + @radix-ui/react-select: Custom select components
  + @radix-ui/react-checkbox: Styled checkboxes
  + @radix-ui/react-slider: Range input controls

#### Styling & Theming

* **Tailwind CSS 3.4.17**: Utility-first CSS framework
* **PostCSS**: CSS post-processor for modern CSS features
* **CSS Custom Properties**: Theme variables for consistent design
* **Tailwind Plugins**:
  + @tailwindcss/typography: Beautiful typographic defaults
  + tailwindcss-animate: Animation utilities

### Data Visualization & Charts

* **Recharts 2.13.3**: Composable charting library built on React and D3
  + Line charts for time-series data
  + Bar charts for categorical comparisons
  + Pie charts for proportional data
  + Area charts for filled time-series
  + Responsive design with tooltips and legends

### State Management & Data Fetching

* **TanStack Query 5.62.7**: Powerful data synchronization for React
  + Server state caching and synchronization
  + Background refetching
  + Optimistic updates
  + Error handling and retry logic
  + Query invalidation and cache management

### Form Handling & Validation

* **React Hook Form 7.54.2**: Performant forms with minimal re-renders
* **Hookform Resolvers**: Integration bridge for validation libraries
* **Zod 3.24.1**: TypeScript-first schema validation
  + Runtime type checking
  + Schema composition and transformation
  + Integration with Drizzle ORM for type safety

### Icons & Visual Assets

* **Lucide React 0.468.0**: Beautiful, customizable SVG icons
  + Consistent 24x24 grid system
  + Tree-shakeable icon imports
  + Accessible with proper ARIA labels
* **React Icons**: Additional icon sets when needed
  + Company logos via react-icons/si
  + Alternative icon styles

### Development Tools & Build System

#### Build & Development

* **ESBuild**: Ultra-fast JavaScript bundler and minifier
* **TypeScript Compiler**: Type checking and code generation
* **Vite Plugins**:
  + @vitejs/plugin-react: React support with Fast Refresh
  + @replit/vite-plugin-cartographer: Replit-specific development tools
  + @replit/vite-plugin-runtime-error-modal: Enhanced error reporting

#### Code Quality & Type Safety

* **Drizzle Kit**: Database migration and introspection tools
* **Drizzle Zod**: Schema validation integration
* **TypeScript Definitions**: Comprehensive type coverage
  + @types/express: Express.js type definitions
  + @types/node: Node.js type definitions
  + @types/react: React type definitions

### Wizard System Extensions & Functionality

The Dashboard Creator implements a sophisticated wizard system with several key extensions that provide comprehensive functionality:

#### Multi-Step Navigation System

* **Custom Wizard Hook**: State management for step progression and validation
* **Step Indicators**: Visual progress display with click-to-navigate functionality
* **Validation Gates**: Prevent progression without required data
* **Auto-Save**: Periodic saving of draft state to prevent data loss
* **Previous/Next Navigation**: Seamless step-by-step progression with validation
* **Step Validation**: Each step validates completion before allowing progression

#### Advanced Drag & Drop System (use-drag-drop.tsx)

The application implements a comprehensive drag-and-drop system that provides:

**Core Functionality:** - **Metric Dragging**: Drag metrics from library to canvas with smooth animations - **Widget Repositioning**: Move existing widgets within canvas boundaries - **Grid Snapping**: Automatic alignment to configurable grid (10px, 20px, 30px) - **Visual Feedback**: Drop zones, drag previews, and cursor state changes - **Collision Detection**: Prevent widget overlap (future enhancement)

**Technical Implementation:** - **Mouse Event Handling**: Precise mouse tracking with offset calculations - **Touch Support**: Mobile-friendly touch event handling - **Performance Optimization**: Debounced operations and efficient re-renders - **State Management**: Centralized drag state with React hooks - **Canvas Integration**: Seamless integration with grid layout system

#### Smart Grid Layout Engine

The grid system provides precise widget positioning and alignment:

**Grid Configuration:** - **Multiple Grid Sizes**: 10px, 20px, 30px options for different precision levels - **Visual Grid Overlay**: CSS-based grid pattern with toggle visibility - **Snap-to-Grid**: Optional automatic alignment during drag operations - **Dynamic Canvas**: Auto-expanding canvas based on widget positions - **Boundary Detection**: Prevent widgets from being dragged outside canvas

**Layout Features:** - **Collision Prevention**: Smart positioning to avoid widget overlap - **Auto-Resize**: Canvas expands to accommodate new widgets - **Grid Guides**: Visual snap lines during drag operations - **Responsive Design**: Grid adapts to different screen sizes - **Memory Persistence**: Grid settings saved with dashboard configuration

#### Advanced Widget Configuration System

Each widget type has specialized configuration options:

**Configuration Interface:** - **Modal Configuration**: Per-widget settings in popup dialogs using Radix UI - **Real-Time Preview**: Immediate visual feedback for configuration changes - **Type-Specific Settings**: Different options for KPI, Chart, and Table widgets - **Form Validation**: Zod schema validation for all configuration inputs - **Auto-Apply**: Changes apply immediately without manual save

**Widget-Specific Features:** - **KPI Widgets**: Number formatting, trend indicators, color schemes, threshold alerts - **Chart Widgets**: Chart type selection, axis configuration, color palettes, data filtering - **Table Widgets**: Column management, sorting options, pagination settings, row styling - **Data Source Mapping**: Connect widgets to different data sources and API endpoints - **Visual Customization**: Titles, descriptions, colors, and display options

#### Comprehensive Preview & Publish System

The preview system provides complete dashboard visualization:

**Preview Features:** - **Modal Preview**: Full-screen dashboard preview in dedicated modal dialog - **Read-Only Mode**: All interactions disabled to simulate end-user experience - **Live Data Simulation**: Sample data rendering with realistic chart animations - **Responsive Preview**: Preview adapts to different screen sizes - **Full-Screen Option**: External window preview for complete testing

**Publishing Workflow:** - **Draft Management**: Auto-save drafts with timestamp tracking - **Version Control**: Track changes with creation and update timestamps - **Publishing States**: Draft vs Published status with appropriate workflows - **Validation**: Ensure dashboard completeness before publishing - **Sharing Configuration**: Future URL-based sharing and embedding system

### Database Schema Extensions

#### JSONB Storage Strategy

* **Flexible Widget Storage**: Store complex widget configurations as JSON
* **Query Performance**: Efficient JSONB operations in PostgreSQL
* **Schema Evolution**: Add new widget types without database migrations
* **Validation Layer**: Zod schemas ensure data integrity

#### Session Management

* **PostgreSQL Sessions**: Persistent session storage using connect-pg-simple
* **Memory Store Fallback**: In-memory sessions for development
* **Session Security**: Configurable session timeouts and security options

### Performance Optimizations

#### Frontend Performance

* **Code Splitting**: Lazy loading of widget components
* **Tree Shaking**: Eliminate unused code from bundles
* **Image Optimization**: SVG icons for scalability and performance
* **Debounced Operations**: Auto-save with 2-second debounce
* **Memoization**: React.memo and useMemo for expensive operations

#### Backend Performance

* **Express Middleware**: Efficient request processing pipeline
* **CORS Configuration**: Optimized cross-origin resource sharing
* **Compression**: Gzip compression for API responses
* **Static Asset Serving**: Efficient static file delivery

### Security & Validation

#### Input Validation

* **Zod Schemas**: Runtime validation for all API endpoints
* **Type Safety**: End-to-end TypeScript type checking
* **Sanitization**: Input cleaning to prevent XSS attacks
* **Rate Limiting**: Future API throttling implementation

#### Data Protection

* **Parameterized Queries**: SQL injection prevention via ORM
* **Session Security**: Secure session configuration
* **Environment Variables**: Secure configuration management

## Wizard Extensions Deep Dive

### Custom Hook Implementation

#### use-dashboard.tsx - Dashboard State Management

* **Purpose**: Centralizes all dashboard-related state and operations
* **Features**: Auto-save functionality, optimistic updates, error handling
* **Integration**: Works seamlessly with TanStack Query for server synchronization

#### use-drag-drop.tsx - Advanced Drag & Drop Engine

* **Mouse Tracking**: Precise cursor position calculation with offset handling
* **Touch Support**: Full mobile device compatibility with touch events
* **Grid Integration**: Automatic snapping to configurable grid boundaries
* **Performance**: Debounced updates to prevent excessive re-renders
* **Visual Feedback**: Dynamic cursor changes and drop zone highlighting

#### use-mobile.tsx - Responsive Design Hook

* **Breakpoint Detection**: Automatic mobile/desktop layout switching
* **Touch Optimization**: Enhanced touch interactions for mobile devices
* **Responsive Grids**: Adaptive grid sizing based on screen dimensions

### Component Architecture Extensions

#### Multi-Step Wizard System

The wizard implements a sophisticated step-by-step flow:

1. **Step 1 - Basic Info**: Dashboard metadata collection
2. **Step 2 - Add Metrics**: Metric selection with real-time library search
3. **Step 3 - Layout & Design**: Canvas customization and widget arrangement
4. **Step 4 - Review & Publish**: Final preview and publishing workflow

**Step Validation Logic:** - Each step validates completion before allowing progression - Visual indicators show step completion status - Previous/Next navigation respects validation rules - Click-to-navigate allows jumping to completed steps

#### Widget Type System

The application supports three main widget types with extensible architecture:

**KPI Widgets:** - Single-value display with trend indicators - Configurable number formatting (currency, percentage, etc.) - Color-coded status indicators - Sample data includes revenue, user counts, and performance metrics

**Chart Widgets:** - Multiple chart types: Line, Bar, Pie, Area - Recharts integration with responsive design - Configurable axes, colors, and data formatting - Interactive tooltips and legends

**Table Widgets:** - Sortable columns with customizable headers - Configurable row styling and pagination - Data filtering and search capabilities - Responsive design for mobile devices

### Data Flow Architecture

#### Real-Time Updates

* **Auto-Save**: Changes saved automatically every 2 seconds
* **Optimistic Updates**: UI updates immediately, syncs to server
* **Conflict Resolution**: Handles concurrent editing scenarios
* **Offline Support**: Local storage fallback for network interruptions

#### Cache Management

* **TanStack Query**: Intelligent server state caching
* **Cache Invalidation**: Strategic cache updates on mutations
* **Background Sync**: Automatic data refresh when window gains focus
* **Stale-While-Revalidate**: Show cached data while fetching updates

### Development Workflow

#### Hot Reload System

* **Vite HMR**: Instant frontend updates during development
* **Express Restart**: Automatic server restart on backend changes
* **Full-Stack Sync**: Coordinated frontend and backend development

#### Error Handling

* **LSP Integration**: Real-time error detection and reporting
* **Runtime Error Modal**: Enhanced error display in development
* **Graceful Degradation**: Fallback states for missing data

#### Testing Infrastructure (Future)

* **React Testing Library**: Component testing framework
* **Jest**: Unit testing framework
* **Cypress**: End-to-end testing for user workflows
* **API Testing**: Backend route testing with supertest

## Database Schema

### Tables Structure

#### 1. Users Table

CREATE TABLE users (  
 id VARCHAR PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 username TEXT UNIQUE NOT NULL,  
 password TEXT NOT NULL  
);

#### 2. Dashboards Table

CREATE TABLE dashboards (  
 id VARCHAR PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 name TEXT NOT NULL,  
 description TEXT,  
 user\_id VARCHAR REFERENCES users(id),  
 widgets JSONB NOT NULL DEFAULT '[]',  
 grid\_settings JSONB NOT NULL DEFAULT '{}',  
 is\_published BOOLEAN NOT NULL DEFAULT FALSE,  
 created\_at TIMESTAMP NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMP NOT NULL DEFAULT NOW()  
);

#### 3. Metrics Table

CREATE TABLE metrics (  
 id VARCHAR PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 name TEXT NOT NULL,  
 description TEXT,  
 category TEXT NOT NULL,  
 type TEXT NOT NULL, -- 'kpi', 'chart', 'table'  
 icon TEXT NOT NULL,  
 color TEXT NOT NULL,  
 default\_config JSONB NOT NULL DEFAULT '{}',  
 sample\_data JSONB NOT NULL DEFAULT '{}'  
);

### Data Models

#### Widget Schema

interface Widget {  
 id: string;  
 metricId: string;  
 x: number;  
 y: number;  
 width: number;  
 height: number;  
 config?: Record<string, any>;  
}

#### Grid Settings Schema

interface GridSettings {  
 showGrid: boolean;  
 gridSize: number;  
 snapToGrid: boolean;  
}

## UI Design Specification

### Design System

#### Color Palette

* **Primary**: hsl(207, 90%, 54%) - Blue for primary actions
* **Secondary**: hsl(142, 70%, 45%) - Green for success states
* **Neutral**: Slate color scale for backgrounds and text
* **Destructive**: hsl(0, 84.2%, 60.2%) - Red for delete actions

#### Typography

* **Font Family**: System font stack with fallbacks
* **Sizes**: Tailwind CSS scale (text-xs to text-4xl)
* **Weights**: 400 (normal), 500 (medium), 600 (semibold), 700 (bold)

#### Spacing

* **Grid Unit**: 20px base grid for canvas positioning
* **Component Spacing**: 4px increments following Tailwind scale
* **Layout Gaps**: 16px-24px between major sections

### Layout Structure

#### Header Bar

* **Height**: 80px
* **Content**: Logo, app title, save/publish actions
* **Background**: White with bottom border
* **Elevation**: Subtle shadow

#### Main Content Area

* **Layout**: Sidebar + Canvas (80/20 split on desktop)
* **Height**: calc(100vh - 80px)
* **Responsive**: Stacked on mobile, tabbed interface

#### Sidebar (320px width)

1. **Wizard Steps Panel**
   * Fixed height: ~200px
   * Shows current progress
   * Clickable step navigation
2. **Metrics Library Panel**
   * Scrollable content area
   * Search functionality
   * Categorized metric groups
   * Drag-enabled metric cards

#### Canvas Area

1. **Canvas Header**
   * Dashboard name and description
   * Action buttons (Preview, Clear All, Grid Toggle)
2. **Grid Canvas**
   * Flexible container with background grid pattern
   * Drop zone for new widgets
   * Positioned widgets with drag handles
3. **Bottom Action Bar**
   * Status indicators (widget count, save status)
   * Navigation buttons (Previous/Next step)

## Wizard Flow Design

### Step 1: Basic Info

**Purpose**: Collect dashboard metadata - Dashboard name input field - Description textarea (optional) - Template selection (future enhancement) - **Next Action**: Proceed to metrics selection

### Step 2: Add Metrics

**Purpose**: Select and add metrics to dashboard - Metrics library with search and filtering - Drag and drop from sidebar to canvas - Real-time widget creation - **Next Action**: Proceed to layout customization

### Step 3: Layout & Design

**Purpose**: Fine-tune widget positioning and appearance - Grid configuration options - Widget resize and reposition - Widget-specific configuration modals - **Next Action**: Proceed to review

### Step 4: Review & Publish

**Purpose**: Final review and publication - Dashboard preview mode - Publish/save options - Sharing configuration - **Next Action**: Deploy dashboard

## Component Architecture

### Core Components

#### 1. DashboardWizard (Main Container)

* **Purpose**: Orchestrates the entire wizard flow
* **State**: Dashboard data, widgets, grid settings, current step
* **Key Functions**: Auto-save, step navigation, publish workflow

#### 2. WizardSteps (Navigation)

* **Purpose**: Shows progress and enables step navigation
* **Features**: Visual progress indicators, click-to-navigate
* **States**: Completed, current, upcoming

#### 3. MetricsLibrary (Content Panel)

* **Purpose**: Display available metrics for selection
* **Features**: Search, categorization, drag initiation
* **Data Source**: Metrics API endpoint

#### 4. DashboardCanvas (Main Workspace)

* **Purpose**: Visual dashboard builder interface
* **Features**: Grid system, drag & drop, widget management
* **Sub-components**: Individual widget renderers

#### 5. Widget Components

* **KpiWidget**: Single-value metrics with trend indicators
* **ChartWidget**: Line, bar, and pie chart visualizations
* **TableWidget**: Tabular data with sorting capabilities
* **Common Features**: Drag handles, configuration access, delete option

### Widget System

#### Widget Types and Properties

1. **KPI Widgets**
   * **Size**: 200x120px (default)
   * **Data**: Single value, change percentage, trend direction
   * **Formats**: Currency, percentage, raw numbers
   * **Visual Elements**: Color-coded icon, large value display
2. **Chart Widgets**
   * **Size**: 400x240px (default)
   * **Chart Types**: Line, bar, pie, area charts
   * **Data**: Time series or categorical data
   * **Interactivity**: Tooltips, hover states
   * **Library**: Recharts for rendering
3. **Table Widgets**
   * **Size**: 480x200px (default)
   * **Features**: Sortable columns, pagination (future)
   * **Data**: Structured row/column format
   * **Styling**: Minimal borders, alternating row colors

#### Widget Configuration System

* **Modal Interface**: Popup configuration for each widget
* **Common Settings**: Title, data source, refresh interval
* **Type-specific Settings**: Chart type, table columns, KPI format
* **Real-time Preview**: Changes apply immediately

## Interaction Design

### Drag and Drop System

#### Drag Sources

1. **Metrics from Library**: Creates new widget on canvas
2. **Existing Widgets**: Repositions widget on canvas

#### Drop Targets

* **Canvas Area**: Primary drop zone for all draggable items
* **Grid Snapping**: Automatic alignment to grid when enabled
* **Collision Detection**: Prevents widget overlap (future enhancement)

#### Visual Feedback

* **Drag State**: Semi-transparent dragged item
* **Drop Zones**: Highlighted areas when dragging
* **Grid Guides**: Visible snap lines during positioning
* **Cursor Changes**: Grab/grabbing cursor states

### Grid System

#### Configuration Options

* **Grid Size**: 10px, 20px, 30px options
* **Visibility**: Toggle grid background pattern
* **Snapping**: Enable/disable snap-to-grid behavior
* **Responsive**: Grid adapts to canvas size changes

#### Positioning Logic

* **Coordinate System**: Top-left origin (0,0)
* **Unit System**: Pixel-based positioning
* **Constraints**: Minimum widget sizes, boundary detection
* **Persistence**: Position data stored in widget configuration

## API Design

### RESTful Endpoints

#### Metrics Management

GET /api/metrics // Get all metrics  
GET /api/metrics/category/:category // Get metrics by category  
GET /api/metrics/:id // Get specific metric  
POST /api/metrics // Create new metric (admin)

#### Dashboard Operations

GET /api/dashboards // Get user dashboards  
GET /api/dashboards/:id // Get specific dashboard  
POST /api/dashboards // Create new dashboard  
PATCH /api/dashboards/:id // Update dashboard  
DELETE /api/dashboards/:id // Delete dashboard

### Data Transfer Objects

#### Dashboard Creation Request

interface CreateDashboardRequest {  
 name: string;  
 description?: string;  
 widgets: Widget[];  
 gridSettings: GridSettings;  
 isPublished: boolean;  
}

#### Widget Data Structure

interface Widget {  
 id: string;  
 metricId: string;  
 x: number;  
 y: number;  
 width: number;  
 height: number;  
 config?: {  
 title?: string;  
 dataSource?: string;  
 refreshInterval?: string;  
 [key: string]: any;  
 };  
}

## Performance Considerations

### Frontend Optimization

* **React Query Caching**: Aggressive caching of metrics data
* **Debounced Auto-save**: 2-second delay to prevent excessive API calls
* **Virtual Scrolling**: For large metric libraries (future enhancement)
* **Lazy Loading**: Code splitting for widget components

### Backend Optimization

* **Response Compression**: Gzip compression for API responses
* **Database Indexing**: Indexes on user\_id, category, and timestamps
* **Query Optimization**: Efficient JSONB queries for widget data
* **Rate Limiting**: API throttling to prevent abuse

### Caching Strategy

* **Browser Cache**: Static assets with long cache headers
* **API Cache**: Short-term caching for metrics data
* **Session Storage**: Temporary storage for unsaved changes
* **Local Storage**: User preferences and grid settings

## Security Considerations

### Data Protection

* **Input Validation**: Zod schemas for all API inputs
* **SQL Injection Prevention**: Parameterized queries via ORM
* **XSS Protection**: React’s built-in escaping
* **CSRF Protection**: Same-origin policy enforcement

### Access Control

* **User Authentication**: Session-based authentication (future)
* **Dashboard Privacy**: User-specific dashboard access
* **Metrics Access**: Role-based metric visibility
* **API Security**: Rate limiting and input sanitization

## Current Issues & Solutions

### Issue 1: Preview Button Functionality

**Problem**: Preview button only shows a toast message **Solution**: Implement modal preview mode with read-only dashboard view

### Issue 2: Next Step Button

**Problem**: Bottom action buttons don’t connect to wizard navigation **Solution**: Wire up Previous/Next buttons to wizard step management

### Issue 3: React Key Warning

**Problem**: Key prop being spread in widget components **Solution**: Extract key prop before spreading widget props

## Future Enhancements

### Phase 2 Features

* **User Authentication**: Login/register system
* **Dashboard Templates**: Pre-built dashboard layouts
* **Custom Metrics**: User-defined metric creation
* **Real Data Sources**: API integrations for live data
* **Collaboration**: Shared dashboard editing
* **Export Options**: PDF/PNG dashboard exports

### Phase 3 Features

* **Advanced Widgets**: Calendar, map, gauge widgets
* **Dashboard Themes**: Custom color schemes and layouts
* **Mobile App**: React Native companion app
* **Analytics**: Usage tracking and optimization
* **Enterprise Features**: Team management, permissions

## Testing Strategy

### Unit Testing

* **Component Tests**: React Testing Library for UI components
* **Hook Tests**: Custom hook testing with renderHook
* **API Tests**: Jest for backend route testing
* **Utility Tests**: Pure function testing

### Integration Testing

* **Drag & Drop**: End-to-end drag operation testing
* **API Integration**: Frontend-backend communication tests
* **Database Tests**: Storage layer integration tests
* **User Flows**: Complete wizard navigation tests

### Performance Testing

* **Load Testing**: Large dashboard rendering performance
* **Memory Testing**: Widget creation/deletion memory leaks
* **API Performance**: Response time benchmarks
* **Bundle Size**: JavaScript bundle optimization

## Deployment Architecture

### Development Environment

* **Local Development**: Vite dev server with HMR
* **API Server**: Express with auto-restart
* **Database**: In-memory storage for rapid development
* **Hot Reload**: Full-stack development with instant feedback

### Production Environment

* **Build Process**: Vite production build with optimization
* **Server**: Single Express server serving static assets and API
* **Database**: PostgreSQL with connection pooling
* **Monitoring**: Error tracking and performance monitoring

### Environment Configuration

* **Development**: Local environment variables
* **Staging**: Test deployment with sample data
* **Production**: Secure environment with real database
* **CI/CD**: Automated testing and deployment pipeline

## Implementation Priority

### High Priority Issues

1. Fix Preview button to show functional preview mode
2. Connect Next/Previous step buttons to wizard navigation
3. Resolve React key warning in widget components
4. Implement proper step validation and progression logic

### Medium Priority Enhancements

1. Add widget resize functionality
2. Implement widget configuration modal actions
3. Add real-time save status indicators
4. Create dashboard export functionality

### Low Priority Features

1. Add keyboard shortcuts for common actions
2. Implement undo/redo functionality
3. Add widget templates and presets
4. Create advanced grid layout options

This design document serves as the blueprint for the Dashboard Creator application, providing both current implementation details and future development roadmap.