# Interactive Visualization Techniques and Libraries for NextJS

This document provides research findings on interactive visualization techniques and best practices for creating flywheel diagrams, flow charts, org charts, and roadmaps in a NextJS application. It includes recommendations for JavaScript libraries that support interactivity and can be customized to match Verida's branding.

## Table of Contents



## Verida Branding Considerations

When implementing interactive visualizations, it's important to maintain consistent branding. Based on the Verida branding information, here are the key elements to incorporate:

### Color Palette

| Color Name | Hex Code | Usage |
|---|---|---|
| Verida Orange (Primary) | #FF5000 | Primary elements, highlights, key nodes |
| Verida Blue | #0077C8 | Secondary elements, supporting nodes |
| Verida Dark Blue | #003366 | Text, backgrounds, borders |
| White | #FFFFFF | Backgrounds, text on dark elements |

### Design Language

- **Clean, Minimalist Aesthetic**: Simple layouts with ample white space
- **Bold Color Usage**: Strategic use of Verida orange as an accent color
- **Modern Interface Elements**: Rounded corners, subtle shadows, clear hierarchy
- **Responsive Design**: Adaptable layouts for various screen sizes

# Best Practices for Interactive Visualizations

## 1. Goal-Oriented Design

- Have a clear purpose for each visualization before implementation
- Focus on the specific insights you want to communicate
- Consider the audience and their needs

## 2. Clarity and Simplicity

- Avoid visual clutter and unnecessary elements
- Use direct labeling instead of legends when possible
- Limit the number of elements displayed at once

## 3. Interactivity

- Implement intuitive interactions (zoom, pan, click, hover)
- Provide clear visual feedback for interactive elements
- Include tooltips for additional information
- Ensure smooth transitions between states

## 4. Accessibility

- Use sufficient color contrast (especially with Verida orange)
- Provide alternative text for screen readers
- Ensure keyboard navigation is possible
- Test with accessibility tools

## 5. Performance

- Optimize rendering for large datasets
- Implement virtualization for complex visualizations
- Consider lazy loading for complex components

## 6. Responsive Design

- Ensure visualizations work across device sizes
- Adapt layout and interaction models for touch devices
- Test on various screen sizes and orientations

# Flywheel Diagrams

Flywheel diagrams represent cyclical, self-reinforcing processes where momentum builds over time. They're ideal for visualizing business models, growth strategies, or continuous improvement cycles.

## Recommended Libraries

### 1. React Flow

**Website**: https://reactflow.dev/ (https://reactflow.dev/)

**Pros**:
- Highly customizable nodes and edges
- Built-in support for interactive features (drag, zoom, pan)
- Excellent React integration with hooks and components
- Active development and community support

- MIT licensed and free to use
- Supports custom styling to match Verida branding

**Cons**:
- Learning curve for advanced customizations
- May require additional work for perfectly circular layouts
- Performance can degrade with very large diagrams

**Code Example**:

```
import React from 'react';
import ReactFlow, {
  Background,
  Controls,
  MiniMap,
  useNodesState,
  useEdgesState
} from 'reactflow';
import 'reactflow/dist/style.css';

// Custom node component for flywheel stages
const CustomNode = ({ data }) => {
  return (
    <div style={{
      padding: '10px',
      borderRadius: '8px',
      backgroundColor: '#FFFFFF',
      border: `2px solid ${data.borderColor || '#FF5000'}`,
      color: '#003366',
      width: '150px',
      textAlign: 'center'
    }}>
      <div style={{ fontWeight: 'bold' }}>{data.label}</div>
      {data.description && <div>{data.description}</div>}
    </div>
  );
};

// Node types registration
const nodeTypes = {
  custom: CustomNode,
};

const FlywheelDiagram = () => {
  // Define nodes in a circular pattern
  const initialNodes = [
    {
      id: '1',
      type: 'custom',
      position: { x: 250, y: 0 },
      data: { label: 'Stage 1', description: 'Description', borderColor: '#FF5000' },
    },
    {
      id: '2',
      type: 'custom',
      position: { x: 500, y: 250 },
      data: { label: 'Stage 2', description: 'Description', borderColor: '#0077C8' },
    },
    {
      id: '3',
      type: 'custom',
      position: { x: 250, y: 500 },
      data: { label: 'Stage 3', description: 'Description', borderColor: '#FF5000' },
    },
    {
      id: '4',
      type: 'custom',
      position: { x: 0, y: 250 },
```

```
      data: { label: 'Stage 4', description: 'Description', borderColor: '#0077C8' },
    },
  ];

  // Define edges connecting the nodes in a cycle
  const initialEdges = [
    { id: 'e1-2', source: '1', target: '2', animated: true, style: { stroke:
'#FF5000' } },
    { id: 'e2-3', source: '2', target: '3', animated: true, style: { stroke:
'#0077C8' } },
    { id: 'e3-4', source: '3', target: '4', animated: true, style: { stroke:
'#FF5000' } },
    { id: 'e4-1', source: '4', target: '1', animated: true, style: { stroke:
'#0077C8' } },
  ];

  const [nodes, setNodes, onNodesChange] = useNodesState(initialNodes);
  const [edges, setEdges, onEdgesChange] = useEdgesState(initialEdges);

  return (
    <div style={{ width: '100%', height: '600px' }}>
      <ReactFlow
        nodes={nodes}
        edges={edges}
        onNodesChange={onNodesChange}
        onEdgesChange={onEdgesChange}
        nodeTypes={nodeTypes}
        fitView
      >
        <Background />
        <Controls />
        <MiniMap />
      </ReactFlow>
    </div>
  );
};

export default FlywheelDiagram;
```

### 2. JointJS (and JointJS+)

**Website**: https://www.jointjs.com/react-diagrams (https://www.jointjs.com/react-diagrams)

**Pros**:
- Professional-grade diagramming library
- Extensive customization options
- Supports SVG and HTML elements
- Excellent documentation and examples
- Supports complex interactions and animations
- React integration available

**Cons**:
- Commercial license required for JointJS+ (advanced features)
- Steeper learning curve
- Heavier than some alternatives

**Code Example**:

```
import React, { useEffect, useRef } from 'react';
import * as joint from 'jointjs';

const JointJSFlywheel = () => {
  const containerRef = useRef(null);

  useEffect(() => {
    // Create graph and paper
    const graph = new joint.dia.Graph();
    const paper = new joint.dia.Paper({
      el: containerRef.current,
      model: graph,
      width: 800,
      height: 600,
      gridSize: 10,
      drawGrid: true,
      background: {
        color: '#F8F9FA',
      },
    });

    // Create nodes for the flywheel
    const stage1 = new joint.shapes.standard.Rectangle({
      position: { x: 300, y: 50 },
      size: { width: 120, height: 60 },
      attrs: {
        body: {
          fill: '#FFFFFF',
          stroke: '#FF5000',
          strokeWidth: 2,
          rx: 8,
          ry: 8,
        },
        label: {
          text: 'Stage 1',
          fill: '#003366',
          fontSize: 14,
        }
      }
    });

    const stage2 = new joint.shapes.standard.Rectangle({
      position: { x: 500, y: 250 },
      size: { width: 120, height: 60 },
      attrs: {
        body: {
          fill: '#FFFFFF',
          stroke: '#0077C8',
          strokeWidth: 2,
          rx: 8,
          ry: 8,
        },
        label: {
          text: 'Stage 2',
          fill: '#003366',
          fontSize: 14,
        }
      }
```

```
});

const stage3 = new joint.shapes.standard.Rectangle({
  position: { x: 300, y: 450 },
  size: { width: 120, height: 60 },
  attrs: {
    body: {
      fill: '#FFFFFF',
      stroke: '#FF5000',
      strokeWidth: 2,
      rx: 8,
      ry: 8,
    },
    label: {
      text: 'Stage 3',
      fill: '#003366',
      fontSize: 14,
    }
  }
});

const stage4 = new joint.shapes.standard.Rectangle({
  position: { x: 100, y: 250 },
  size: { width: 120, height: 60 },
  attrs: {
    body: {
      fill: '#FFFFFF',
      stroke: '#0077C8',
      strokeWidth: 2,
      rx: 8,
      ry: 8,
    },
    label: {
      text: 'Stage 4',
      fill: '#003366',
      fontSize: 14,
    }
  }
});

// Add nodes to the graph
graph.addCells([stage1, stage2, stage3, stage4]);

// Create links between nodes
const link1 = new joint.shapes.standard.Link({
  source: { id: stage1.id },
  target: { id: stage2.id },
  attrs: {
    line: {
      stroke: '#FF5000',
      strokeWidth: 2,
      targetMarker: {
        type: 'path',
        d: 'M 10 -5 0 0 10 5 Z',
        fill: '#FF5000'
      }
    }
  }
});
```

```
  const link2 = new joint.shapes.standard.Link({
    source: { id: stage2.id },
    target: { id: stage3.id },
    attrs: {
      line: {
        stroke: '#0077C8',
        strokeWidth: 2,
        targetMarker: {
          type: 'path',
          d: 'M 10 -5 0 0 10 5 Z',
          fill: '#0077C8'
        }
      }
    }
  });

  const link3 = new joint.shapes.standard.Link({
    source: { id: stage3.id },
    target: { id: stage4.id },
    attrs: {
      line: {
        stroke: '#FF5000',
        strokeWidth: 2,
        targetMarker: {
          type: 'path',
          d: 'M 10 -5 0 0 10 5 Z',
          fill: '#FF5000'
        }
      }
    }
  });

  const link4 = new joint.shapes.standard.Link({
    source: { id: stage4.id },
    target: { id: stage1.id },
    attrs: {
      line: {
        stroke: '#0077C8',
        strokeWidth: 2,
        targetMarker: {
          type: 'path',
          d: 'M 10 -5 0 0 10 5 Z',
          fill: '#0077C8'
        }
      }
    }
  });

  // Add links to the graph
  graph.addCells([link1, link2, link3, link4]);

  // Clean up on unmount
  return () => {
    paper.remove();
  };
}, []);

return <div ref={containerRef} style={{ width: '100%', height: '600px' }}></div>;
```

```
};

export default JointJSFlywheel;
```

### 3. Cytoscape.js

**Website**: https://js.cytoscape.org/ (https://js.cytoscape.org/)

**Pros**:
- Specialized for graph theory and network visualization
- Excellent for complex, interconnected diagrams
- Strong performance with large datasets
- Supports custom styling and layouts
- Free and open-source

**Cons**:
- Less React-specific than alternatives
- May require additional wrapper components
- More focused on networks than business diagrams

## Flow Charts

Flow charts visualize processes, workflows, or algorithms as a series of steps connected by arrows.

### Recommended Libraries

#### 1. React Flow

**Website**: https://reactflow.dev/ (https://reactflow.dev/)

**Pros**:
- Purpose-built for flow diagrams
- Excellent React integration
- Highly customizable nodes and edges
- Built-in interactivity (drag, zoom, pan)
- Active development and community
- MIT licensed

**Cons**:
- May require custom styling for complex diagrams
- Advanced features have a learning curve

**Code Example**:

```
import React, { useState, useCallback } from 'react';
import ReactFlow, {
  addEdge,
  Background,
  Controls,
  MiniMap,
  useNodesState,
  useEdgesState,
} from 'reactflow';
import 'reactflow/dist/style.css';

// Define custom node types if needed
const nodeTypes = {};

const FlowChart = () => {
  // Initial nodes
  const initialNodes = [
    {
      id: '1',
      type: 'input',
      data: { label: 'Start' },
      position: { x: 250, y: 0 },
      style: {
        background: '#FFFFFF',
        color: '#003366',
        border: '2px solid #FF5000',
        borderRadius: '8px',
        padding: '10px',
      },
    },
    {
      id: '2',
      data: { label: 'Process A' },
      position: { x: 250, y: 100 },
      style: {
        background: '#FFFFFF',
        color: '#003366',
        border: '2px solid #0077C8',
        borderRadius: '8px',
        padding: '10px',
      },
    },
    {
      id: '3',
      data: { label: 'Decision' },
      position: { x: 250, y: 200 },
      style: {
        background: '#FFFFFF',
        color: '#003366',
        border: '2px solid #FF5000',
        borderRadius: '8px',
        padding: '10px',
      },
    },
    {
      id: '4',
      data: { label: 'Process B' },
      position: { x: 100, y: 300 },
```

```
      style: {
        background: '#FFFFFF',
        color: '#003366',
        border: '2px solid #0077C8',
        borderRadius: '8px',
        padding: '10px',
      },
    },
    {
      id: '5',
      data: { label: 'Process C' },
      position: { x: 400, y: 300 },
      style: {
        background: '#FFFFFF',
        color: '#003366',
        border: '2px solid #0077C8',
        borderRadius: '8px',
        padding: '10px',
      },
    },
    {
      id: '6',
      type: 'output',
      data: { label: 'End' },
      position: { x: 250, y: 400 },
      style: {
        background: '#FFFFFF',
        color: '#003366',
        border: '2px solid #FF5000',
        borderRadius: '8px',
        padding: '10px',
      },
    },
  ];

  // Initial edges
  const initialEdges = [
    { id: 'e1-2', source: '1', target: '2', animated: false, style: { stroke:
'#FF5000' } },
    { id: 'e2-3', source: '2', target: '3', animated: false, style: { stroke:
'#0077C8' } },
    {
      id: 'e3-4',
      source: '3',
      target: '4',
      animated: false,
      label: 'Yes',
      labelStyle: { fill: '#003366', fontWeight: 'bold' },
      style: { stroke: '#FF5000' }
    },
    {
      id: 'e3-5',
      source: '3',
      target: '5',
      animated: false,
      label: 'No',
      labelStyle: { fill: '#003366', fontWeight: 'bold' },
      style: { stroke: '#FF5000' }
    },
```

```
    { id: 'e4-6', source: '4', target: '6', animated: false, style: { stroke:
'#0077C8' } },
    { id: 'e5-6', source: '5', target: '6', animated: false, style: { stroke:
'#0077C8' } },
  ];

  const [nodes, setNodes, onNodesChange] = useNodesState(initialNodes);
  const [edges, setEdges, onEdgesChange] = useEdgesState(initialEdges);

  const onConnect = useCallback(
    (params) => setEdges((eds) => addEdge({ ...params, animated: true }, eds)),
    [setEdges]
  );

  return (
    <div style={{ width: '100%', height: '600px' }}>
      <ReactFlow
        nodes={nodes}
        edges={edges}
        onNodesChange={onNodesChange}
        onEdgesChange={onEdgesChange}
        onConnect={onConnect}
        nodeTypes={nodeTypes}
        fitView
      >
        <Background />
        <Controls />
        <MiniMap />
      </ReactFlow>
    </div>
  );
};

export default FlowChart;
```

### 2. Flowy

**Website**: https://github.com/alyssaxuu/flowy (https://github.com/alyssaxuu/flowy)

**Pros**:
- Lightweight and minimal
- Easy to set up
- Good for simple flowcharts
- Drag-and-drop functionality

**Cons**:
- Less feature-rich than alternatives
- Not React-specific (requires DOM manipulation)
- Less active development

## Organizational Charts

Organizational charts display hierarchical relationships between people or departments within an organization.

## Recommended Libraries

### 1. React Organizational Chart

**Website**: https://www.npmjs.com/package/react-organizational-chart (https://www.npmjs.com/package/react-organizational-chart)

**Pros**:
- Specifically designed for org charts
- Simple API with React components
- Supports custom styling
- Lightweight (494 KB)
- MIT licensed

**Cons**:
- Limited interactivity out of the box
- Basic feature set
- Less active development (last published 2 years ago)

**Code Example**:

```
import React from 'react';
import { Tree, TreeNode } from 'react-organizational-chart';
import styled from 'styled-components';

const StyledNode = styled.div`
  padding: 10px;
  border-radius: 8px;
  display: inline-block;
  border: 2px solid #FF5000;
  background-color: white;
  color: #003366;
`;

const StyledChildNode = styled.div`
  padding: 10px;
  border-radius: 8px;
  display: inline-block;
  border: 2px solid #0077C8;
  background-color: white;
  color: #003366;
`;

const OrgChart = () => {
  return (
    <Tree
      lineWidth={'2px'}
      lineColor={'#FF5000'}
      lineBorderRadius={'10px'}
      label={<StyledNode>CEO</StyledNode>}
    >
      <TreeNode label={<StyledChildNode>CTO</StyledChildNode>}>
        <TreeNode label={<StyledChildNode>Engineering Manager</StyledChildNode>}>
          <TreeNode label={<StyledChildNode>Frontend Developer</StyledChildNode>} />
          <TreeNode label={<StyledChildNode>Backend Developer</StyledChildNode>} />
        </TreeNode>
        <TreeNode label={<StyledChildNode>QA Manager</StyledChildNode>}>
          <TreeNode label={<StyledChildNode>QA Engineer</StyledChildNode>} />
        </TreeNode>
      </TreeNode>
      <TreeNode label={<StyledChildNode>CFO</StyledChildNode>}>
        <TreeNode label={<StyledChildNode>Accounting Manager</StyledChildNode>} />
        <TreeNode label={<StyledChildNode>Financial Analyst</StyledChildNode>} />
      </TreeNode>
      <TreeNode label={<StyledChildNode>COO</StyledChildNode>}>
        <TreeNode label={<StyledChildNode>Operations Manager</StyledChildNode>} />
        <TreeNode label={<StyledChildNode>HR Manager</StyledChildNode>} />
      </TreeNode>
    </Tree>
  );
};

export default OrgChart;
```

### 2. PrimeReact OrganizationChart

**Website**: https://www.primefaces.org/primereact/showcase/#/organizationchart (https://www.primefaces.org/primereact/showcase/#/organizationchart)

**Pros**:
- Part of a comprehensive UI component library
- Good documentation and examples
- Supports templating for custom node content
- Interactive features (expand/collapse, selection)
- Active development and support

**Cons**:
- Requires importing the entire PrimeReact library
- Commercial license for premium features
- Less customizable than specialized libraries

**Code Example**:

```
import React, { useState } from 'react';
import { OrganizationChart } from 'primereact/organizationchart';
import 'primereact/resources/themes/lara-light-indigo/theme.css';
import 'primereact/resources/primereact.min.css';
import 'primeicons/primeicons.css';

const PrimeOrgChart = () => {
  const [selection, setSelection] = useState([]);

  const data = {
    key: '0',
    type: 'person',
    styleClass: 'p-person',
    data: {
      name: 'CEO',
      avatar: 'avatar.png'
    },
    children: [
      {
        key: '0_0',
        type: 'person',
        styleClass: 'p-person',
        data: {
          name: 'CTO',
          avatar: 'avatar.png'
        },
        children: [
          {
            key: '0_0_0',
            type: 'person',
            styleClass: 'p-person',
            data: {
              name: 'Engineering Manager',
              avatar: 'avatar.png'
            },
            children: [
              {
                key: '0_0_0_0',
                type: 'person',
                styleClass: 'p-person',
                data: {
                  name: 'Frontend Developer',
                  avatar: 'avatar.png'
                }
              },
              {
                key: '0_0_0_1',
                type: 'person',
                styleClass: 'p-person',
                data: {
                  name: 'Backend Developer',
                  avatar: 'avatar.png'
                }
              }
            ]
          }
        ]
      },
```

```
      {
        key: '0_1',
        type: 'person',
        styleClass: 'p-person',
        data: {
          name: 'CFO',
          avatar: 'avatar.png'
        },
        children: [
          {
            key: '0_1_0',
            type: 'person',
            styleClass: 'p-person',
            data: {
              name: 'Financial Analyst',
              avatar: 'avatar.png'
            }
          }
        ]
      }
    ]
  };

  const nodeTemplate = (node) => {
    return (
      <div className="p-organizationchart-node-content" style={{
        padding: '10px',
        borderRadius: '8px',
        border: node.data.name.includes('CEO') ? '2px solid #FF5000' : '2px solid
#0077C8',
        backgroundColor: 'white',
        color: '#003366'
      }}>
        <div>{node.data.name}</div>
      </div>
    );
  };

  return (
    <div className="card">
      <OrganizationChart
        value={data}
        nodeTemplate={nodeTemplate}
        selection={selection}
        selectionMode="multiple"
        onSelectionChange={e => setSelection(e.data)}
      />
    </div>
  );
};

export default PrimeOrgChart;
```

### 3. yFiles React Organization Chart

**Website**: https://www.yworks.com/pages/yfiles-react-organization-chart-component (https://www.yworks.com/pages/yfiles-react-organization-chart-component)

**Pros**:
- Professional-grade diagramming library
- Highly interactive and customizable
- Advanced features (search, tooltips, context menus)
- Excellent performance with large org charts
- React-specific implementation

**Cons**:
- Commercial license required
- More complex setup
- Larger file size

# Roadmaps and Timelines

Roadmaps and timelines visualize events, milestones, or tasks over time, often used for project planning and strategic communication.

## Recommended Libraries

### 1. vis-timeline

**Website**: https://visjs.github.io/vis-timeline/docs/timeline/ (https://visjs.github.io/vis-timeline/docs/timeline/)

**Pros**:
- Specialized for interactive timelines
- Highly customizable
- Supports items with start/end dates
- Interactive features (zoom, drag, edit)
- Open-source

**Cons**:
- Not React-specific (requires wrapper)
- Learning curve for advanced features

**Code Example**:

```
import React, { useEffect, useRef } from 'react';
import { Timeline, DataSet } from 'vis-timeline/standalone';
import 'vis-timeline/styles/vis-timeline-graph2d.css';

const VisTimeline = () => {
  const containerRef = useRef(null);

  useEffect(() => {
    // Create a DataSet with items
    const items = new DataSet([
      { id: 1, content: 'Phase 1', start: '2023-01-01', end: '2023-03-31', style: 'back
ground-color: rgba(255, 80, 0, 0.2); color: #003366; border-color: #FF5000;' },
      { id: 2, content: 'Phase 2', start: '2023-04-01', end: '2023-06-30', style: 'back
ground-color: rgba(0, 119, 200, 0.2); color: #003366; border-color: #0077C8;' },
      { id: 3, content: 'Phase 3', start: '2023-07-01', end: '2023-09-30', style: 'back
ground-color: rgba(255, 80, 0, 0.2); color: #003366; border-color: #FF5000;' },
      { id: 4, content: 'Phase 4', start: '2023-10-01', end: '2023-12-31', style: 'back
ground-color: rgba(0, 119, 200, 0.2); color: #003366; border-color: #0077C8;' },
      { id: 5, content: 'Milestone 1', start: '2023-03-15', type: 'point', style: 'col-
or: #003366; border-color: #FF5000;' },
      { id: 6, content: 'Milestone 2', start: '2023-06-15', type: 'point', style: 'col-
or: #003366; border-color: #0077C8;' },
      { id: 7, content: 'Milestone 3', start: '2023-09-15', type: 'point', style: 'col-
or: #003366; border-color: #FF5000;' },
      { id: 8, content: 'Milestone 4', start: '2023-12-15', type: 'point', style: 'col-
or: #003366; border-color: #0077C8;' },
    ]);

    // Configuration for the Timeline
    const options = {
      height: '400px',
      min: new Date(2023, 0, 1),
      max: new Date(2023, 11, 31),
      zoomMin: 1000 * 60 * 60 * 24 * 7, // One week in milliseconds
      zoomMax: 1000 * 60 * 60 * 24 * 365, // One year in milliseconds
      editable: true,
      tooltip: {
        followMouse: true,
        overflowMethod: 'cap'
      }
    };

    // Create a Timeline
    const timeline = new Timeline(containerRef.current, items, options);

    // Clean up on unmount
    return () => {
      timeline.destroy();
    };
  }, []);

  return <div ref={containerRef}></div>;
};

export default VisTimeline;
```

### 2. Timeline.js

**Website**: https://timeline.knightlab.com/ (https://timeline.knightlab.com/)

**Pros**:
- Simple setup with minimal coding
- Supports multimedia content
- Good for storytelling and educational timelines
- Free and open-source

**Cons**:
- Less interactive than alternatives
- Limited customization options
- Not React-specific

### 3. React Chrono

**Website**: https://github.com/prabhuignoto/react-chrono (https://github.com/prabhuignoto/react-chrono)

**Pros**:
- React-specific timeline component
- Multiple layout modes (vertical, horizontal)
- Supports media (images, videos)
- Customizable themes
- Active development

**Cons**:
- Less suitable for complex project roadmaps
- Fewer interactive features than vis-timeline

**Code Example**:

```
import React from 'react';
import { Chrono } from 'react-chrono';

const ReactChronoTimeline = () => {
  const items = [
    {
      title: "January 2023",
      cardTitle: "Project Kickoff",
      cardSubtitle: "Initial planning and team formation",
      cardDetailedText:
"Detailed description of the kickoff phase and initial planning activities."
    },
    {
      title: "March 2023",
      cardTitle: "Design Phase",
      cardSubtitle: "UI/UX design and prototyping",
      cardDetailedText:
"Detailed description of the design phase including wireframing and user testing."
    },
    {
      title: "June 2023",
      cardTitle: "Development Sprint 1",
      cardSubtitle: "Core functionality implementation",
      cardDetailedText: "Details about the first development sprint and key features
implemented."
    },
    {
      title: "September 2023",
      cardTitle: "Beta Release",
      cardSubtitle: "Limited user testing",
      cardDetailedText: "Information about the beta release process and feedback col-
lection."
    },
    {
      title: "December 2023",
      cardTitle: "Public Launch",
      cardSubtitle: "Full product release",
      cardDetailedText: "Details about the public launch event and marketing activit-
ies."
    }
  ];

  return (
    <div style={{ width: '100%', height: '500px' }}>
      <Chrono
        items={items}
        mode="VERTICAL"
        theme={{
          primary: '#FF5000',
          secondary: '#0077C8',
          cardBgColor: '#FFFFFF',
          cardForeColor: '#003366',
          titleColor: '#003366'
        }}
        cardHeight={200}
        slideShow
        slideItemDuration={2000}
        enableOutline
```

```
      />
    </div>
  );
};

export default ReactChronoTimeline;
```

## Conclusion

Based on the research conducted, here are the top recommendations for each visualization type:

### Flywheel Diagrams

1. **React Flow** - Best overall choice for its React integration, customization options, and active development.
2. **JointJS/JointJS+** - Good alternative for complex, professional-grade diagrams (commercial license).

### Flow Charts

1. **React Flow** - Excellent choice for its purpose-built flow diagram capabilities and React integration.
2. **Flowy** - Simpler alternative for basic flowcharts.

### Organizational Charts

1. **React Organizational Chart** - Simple, lightweight option for basic org charts.
2. **PrimeReact OrganizationChart** - Good balance of features and ease of use.
3. **yFiles React Organization Chart** - Best for complex, enterprise-grade org charts (commercial license).

### Roadmaps and Timelines

1. **vis-timeline** - Most powerful and customizable option for interactive timelines.
2. **React Chrono** - React-specific alternative with good styling options.

For a NextJS application that needs to implement all these visualization types while maintaining consistent branding and user experience, **React Flow** stands out as the most versatile option that could potentially handle multiple visualization types with a consistent API and styling approach. For specialized needs like complex org charts or highly interactive timelines, supplementing with purpose-built libraries like **React Organizational Chart** or **vis-timeline** would be recommended.

All recommended libraries can be styled to match Verida's branding colors and design language, with special attention to using the Verida Orange (#FF5000) for primary elements and highlights, and ensuring sufficient contrast for accessibility.