Gantt Chart and Precedence Diagram Implementation Guide

Library Recommendations for Short Timeline Development

Date: December 16, 2025 **Project:** Nye Hædda Barneskole - PM Simulator **Team Size:** 4 developers **Deadline:** December 17, 2025 (End of Day) **Topic:** Choosing the right libraries for Gantt Chart and Precedence Diagram implementation

---

📋 Original Question

From Team:

> Teacher told us that we could use "https://ui.shadcn.com/" library to make the gantt diagram. Would you recommend this or are there more easier ways to implement this? Would you recommend to use this also for the precedence diagram? If so, how? Let's discuss this firstly - remember we have a short timespan to finish our project (by tomorrow EOD, we are 4 developers only).

🤔 Shadcn UI for Gantt/Precedence Diagrams?

Short Answer

❌ **No, Shadcn is NOT the right tool for these diagrams.**

Why Not?

Shadcn UI is:

- ✅ Beautiful UI components (buttons, cards, forms, dialogs, tables)
- ✅ Great for application structure and layout
- ✅ Perfect for the UI **around** your diagrams

Shadcn UI is NOT:

- ❌ A charting or visualization library
- ❌ Designed for timeline-based displays
- ❌ Designed for network/graph diagrams
- ❌ Has NO built-in Gantt or precedence diagram components

Recommended Use of Shadcn UI

✅ **DO use Shadcn for:**

- Tab components (switching between Overview/Gantt/Precedence views)
- Card components (wrapping the diagrams)
- Button components (zoom, export, filter controls)
- Dialog components (modals, tooltips)

❌ **DON'T use Shadcn for:**

- The actual Gantt chart rendering
- The actual Precedence diagram rendering

📊 GANTT CHART - Library Options

Option 1: gantt-task-react (RECOMMENDED) ⭐

Effort: 2-3 hours (including learning) **Difficulty:** Easy **Quality:** Professional

Pros

- ✅ Pre-built React Gantt component
- ✅ Looks professional out of the box
- ✅ Less code to write
- ✅ Handles timeline rendering, task bars, dependencies
- ✅ Good documentation
- ✅ TypeScript support

Cons

- ❌ New dependency to learn (1 hour docs reading)
- ❌ May not match your exact design system
- ❌ Less control over styling

Installation

```
npm install gantt-task-react
```

Basic Implementation

```
import { Gantt, Task, ViewMode } from 'gantt-task-react'
import "gantt-task-react/dist/index.css"

export function GanttChart({ wbsItems, commitments, timeline }) {
  const tasks: Task[] = wbsItems
    .filter(item => timeline.earliest_start[item.id])
    .map(item => ({
      id: item.id,
      name: item.name,
      start: new Date(timeline.earliest_start[item.id]),
      end: new Date(timeline.earliest_finish[item.id]),
      progress: commitments.find(c => c.wbs_item_id === item.id) ? 100 : 0,
      type: 'task',
      styles: {
        backgroundColor: timeline.critical_path.includes(item.id)
          ? '#ef4444' // red for critical path
          : item.negotiable
          ? '#22c55e' // green for negotiable
          : '#9ca3af', // gray for locked
        progressColor: '#1d4ed8'
      }
    }))

  return (
    <div className="bg-white dark:bg-gray-800 p-6 rounded-lg">
      <h3 className="text-xl font-bold mb-4">Gantt Chart</h3>
      <Gantt
        tasks={tasks}
        viewMode={ViewMode.Month}
        locale="nb-NO"
      />
    </div>
  )
}
```

Option 2: Frappe Gantt (ALTERNATIVE)

Effort: 3 hours **Difficulty:** Easy-Medium **Quality:** Beautiful

Pros

- ✅ Lightweight (only 40KB)
- ✅ Beautiful default design
- ✅ Interactive (click, drag)
- ✅ Simple API

Cons

- ❌ Not React-native (requires wrapper component)
- ❌ Less customizable than gantt-task-react
- ❌ Smaller community

Installation

```
npm install frappe-gantt
```

Option 3: Custom Timeline with Tailwind CSS

Effort: 3-4 hours **Difficulty:** Medium **Quality:** Good (if done well)

Pros

- ✅ Full control over design
- ✅ No new dependencies
- ✅ Uses only HTML/CSS/Tailwind (already familiar)
- ✅ Can be "good enough" for MVP

Cons

- ❌ More coding work
- ❌ No advanced features (drag/drop, zoom)
- ❌ Manual timeline calculations
- ❌ More potential for bugs

Visual Example

```
┌─────────────────────────────────────────────────────────┐
│ Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   │ ← Timeline
├─────────────────────────────────────────────────────────┤
│ 1.3.1 [████████░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░]        │ ← Task bar
│ 1.3.2      [████████████░░░░░░░░░░░░░░░░░░░░░]        │
│ 1.4.1                [████████████████░░░░░░░░]        │
└─────────────────────────────────────────────────────────┘
```

🔗 PRECEDENCE DIAGRAM - Library Options

Option 1: ReactFlow (STRONGLY RECOMMENDED) ⭐⭐⭐

Effort: 4-5 hours (including learning) **Difficulty:** Medium **Quality:** Excellent

Why ReactFlow is PERFECT for Precedence Diagrams

Precedence diagrams are **node-based graphs** with:

- Nodes = WBS items (boxes showing task info)
- Edges = Dependencies (arrows connecting tasks)

ReactFlow is SPECIFICALLY BUILT for this use case.

Pros

- ✅ **BUILT FOR THIS** - node-based diagrams are its specialty
- ✅ Automatic edge routing (arrows don't overlap)
- ✅ Built-in zoom and pan
- ✅ Drag nodes to reposition
- ✅ Excellent documentation and examples
- ✅ Large community (active development)
- ✅ TypeScript support
- ✅ Highly customizable node rendering
- ✅ Can highlight critical path easily

Cons

- ❌ New library to learn (2 hours for basics)
- ❌ Some setup required for custom nodes
- ❌ Need to calculate initial node positions (or use auto-layout)

Installation

```
npm install reactflow
```

Basic Implementation

```
import ReactFlow, { Node, Edge, Controls, Background } from 'reactflow'
import 'reactflow/dist/style.css'

export function PrecedenceDiagram({ wbsItems, timeline }) {
  // Build nodes
  const nodes: Node[] = wbsItems.map((item, idx) => {
    const isCritical = timeline.critical_path.includes(item.id)
    const es = Math.ceil(
      (new Date(timeline.earliest_start[item.id]) - new Date('2025-01-15'))
      / 86400000
    )
    const ef = Math.ceil(
      (new Date(timeline.earliest_finish[item.id]) - new Date('2025-01-15'))
      / 86400000
    )

    return {
      id: item.id,
      data: {
        label: (
          <div className="text-center">
            <div className="font-bold text-sm">{item.id}</div>
            <div className="text-xs">{item.name}</div>
            <div className="text-xs mt-1">ES:{es} EF:{ef}</div>
            {isCritical && (
              <div className="text-xs font-semibold text-red-600">KRITISK</div>
            )}
          </div>
        )
      },
      position: {
        x: (idx % 5) * 200,
        y: Math.floor(idx / 5) * 150
      },
      style: {
        background: isCritical ? '#fee2e2' : '#f3f4f6',
        border: `2px solid ${isCritical ? '#ef4444' : '#9ca3af'}`,
        borderRadius: '8px',
        padding: '10px',
        width: '160px'
      }
    }
  })

  // Build edges (dependency arrows)
  const edges: Edge[] = []
  wbsItems.forEach(item => {
    item.dependencies?.forEach(depId => {
      const isCriticalEdge =
        timeline.critical_path.includes(depId) &&
        timeline.critical_path.includes(item.id)

      edges.push({
        id: `${depId}-${item.id}`,
        source: depId,
        target: item.id,
        type: 'smoothstep',
        animated: isCriticalEdge,
        style: {
          stroke: isCriticalEdge ? '#ef4444' : '#9ca3af',
          strokeWidth: isCriticalEdge ? 3 : 2
        }
      })
    })
  })

  return (
    <div className="bg-white dark:bg-gray-800 p-6 rounded-lg h-[600px]">
      <h3 className="text-xl font-bold mb-4">Precedence Diagram (AON)</h3>
      <ReactFlow
        nodes={nodes}
        edges={edges}
        fitView
      >
        <Background />
        <Controls />
      </ReactFlow>
    </div>
  )
}
```

Features You Get for Free

- ✅ Zoom in/out with mouse wheel
- ✅ Pan by dragging background
- ✅ Drag nodes to reposition
- ✅ Mini-map (optional)
- ✅ Node selection
- ✅ Edge animation for critical path

Option 2: vis-network (ALTERNATIVE)

Effort: 4 hours **Difficulty:** Medium **Quality:** Good

Pros

- ✅ Automatic layout (physics simulation positions nodes)
- ✅ Interactive
- ✅ Good for complex networks

Cons

- ❌ Not React-native (needs wrapper)
- ❌ Harder to style to match your design system
- ❌ Less control than ReactFlow
- ❌ Physics can make layout unpredictable

Option 3: Custom SVG (NOT RECOMMENDED)

Effort: 6-8 hours **Difficulty:** Hard **Quality:** Variable

Why NOT Recommended for Your Timeline

- ❌ MOST work required
- ❌ You handle node positioning manually
- ❌ You handle arrow path calculations manually
- ❌ You handle zoom/pan from scratch
- ❌ You handle node dragging from scratch
- ❌ High chance of bugs
- ❌ No time for this with tomorrow deadline

Only Consider If:

- You have very specific design requirements
- You can't use external libraries (not your case)

- You have 2+ extra days

🎯 FINAL RECOMMENDATION FOR YOUR TEAM

The Winning Strategy

Given your constraints:

- ✅ 4 developers
- ✅ Tight deadline (tomorrow EOD)
- ✅ Need professional quality
- ✅ Need to focus on core features too

Use These Libraries:

Feature	Library	Developer	Time
Gantt Chart	`gantt-task-react`	Developer 2	3 hours
Precedence Diagram	`ReactFlow`	Developer 3	5 hours
UI Shell	Shadcn UI	Both	Already have

Total Visualization Time: 8 hours (instead of 12-16 custom)

Why This Works

Time Savings:

- ✅ Saves 4-8 hours vs building custom
- ✅ More time for testing and bug fixes
- ✅ More time for core features (validation, export, tests)

Quality:

- ✅ Professional, polished look

- ✅ Proven, battle-tested libraries
- ✅ Fewer bugs than custom implementation
- ✅ Better UX (zoom, pan, interactions built-in)

Learning Curve:

- ✅ Both libraries have excellent docs
- ✅ Many examples available online
- ✅ Active communities for help

📋 IMPLEMENTATION PLAN (4 Developers)

Developer 1: Core Flow (10 hours)

Focus: Session completion, validation, export, renegotiation

Tasks:

1. Backend: Validation endpoint with critical path
2. Backend: Export endpoint
3. Backend: Uncommit endpoint
4. Frontend: Validation modals (success + error)
5. Frontend: Export handler
6. Frontend: Uncommit modal

Deliverables:

- Users can submit plan and see validation results
- Users can export session as JSON
- Users can uncommit and renegotiate

Developer 2: Gantt Chart (3 hours)

Focus: Timeline visualization using gantt-task-react

Tasks:

1. Install `gantt-task-react` (5 min)
2. Read documentation (30 min)
3. Create `GanttChart.tsx` component (1 hour)
4. Transform WBS data to task format (30 min)
5. Style to match design system (30 min)
6. Add to dashboard tabs (15 min)
7. Test with real data (15 min)

Deliverables:

- Gantt chart showing all 15 WBS items
- Color-coded: Red (critical path), Green (negotiable), Gray (locked)
- Timeline from Jan 2025 to May 2026
- Deadline marker visible

Developer 3: Precedence Diagram (5 hours)

Focus: Network diagram using ReactFlow

Tasks:

1. Install `reactflow` (5 min)
2. Read documentation and examples (1 hour)
3. Create `PrecedenceDiagram.tsx` component (1.5 hours)
4. Build nodes with ES/EF/LS/LF data (1 hour)
5. Build edges from dependencies (30 min)
6. Style critical path (red) vs normal (gray) (30 min)
7. Add to dashboard tabs (15 min)
8. Test interactions (zoom, pan, drag) (15 min)

Deliverables:

- AON network diagram with all WBS items
- Nodes show: ID, name, ES, EF, LS, LF
- Critical path highlighted in red
- Dependency arrows connect tasks
- Interactive (zoom, pan works)

Developer 4: Chat Fix + Tests (9 hours)

Focus: Bug fixes and quality assurance

Tasks:

1. Chat history loading from DB (2 hours)
2. Backend tests - validation, export, uncommit (4 hours)
3. Frontend tests - dashboard, modals (3 hours)

Deliverables:

- Chat messages persist across page refresh
- Test suite covering core functionality
- CI/CD ready for deployment

💻 QUICK START GUIDE

Step 1: Install Libraries

```
# For Gantt Chart
npm install gantt-task-react

# For Precedence Diagram
npm install reactflow

# Both are React-compatible, TypeScript-ready
```

Step 2: Import Styles

```
// In your Gantt component
import "gantt-task-react/dist/index.css"

// In your Precedence component
import 'reactflow/dist/style.css'
```

Step 3: Integrate with Dashboard

```
// In dashboard/page.tsx

const [activeTab, setActiveTab] = useState<'overview' | 'gantt' | 'precedence'>
('overview')

// Tabs
<div className="flex gap-2 mb-4">
  <button onClick={() => setActiveTab('overview')}>Oversikt</button>
  <button onClick={() => setActiveTab('gantt')}>Gantt Chart</button>
  <button onClick={() => setActiveTab('precedence')}>Precedensdiagram</button>
</div>

// Content
{activeTab === 'overview' && <DashboardOverview />}
{activeTab === 'gantt' && <GanttChart wbsItems={wbs} commitments={commits}
timeline={timeline} />}
{activeTab === 'precedence' && <PrecedenceDiagram wbsItems={wbs} timeline=
{timeline} />}
```

🔧 TROUBLESHOOTING

Common Issues with gantt-task-react

Issue: Tasks don't show up

- **Fix:** Ensure start/end are valid Date objects, not strings

Issue: Styling doesn't match

- **Fix:** Override with custom CSS or use `styles` prop on each task

Issue: Norwegian locale not working

- **Fix:** Check locale prop is set to "nb-NO"

Common Issues with ReactFlow

Issue: Nodes overlap

- **Fix:** Adjust initial positions or use layout library like `dagre`

Issue: Edges don't connect properly

- **Fix:** Ensure source/target IDs match node IDs exactly

Issue: Can't see diagram

- **Fix:** Set explicit height on container (e.g., `h-[600px]`)

📚 DOCUMENTATION LINKS

gantt-task-react

- **GitHub:** https://github.com/MaTeMaTuK/gantt-task-react
- **NPM:** https://www.npmjs.com/package/gantt-task-react
- **Examples:** https://github.com/MaTeMaTuK/gantt-task-react/tree/main/example

ReactFlow

- **Website:** https://reactflow.dev/
- **GitHub:** https://github.com/xyflow/xyflow
- **Examples:** https://reactflow.dev/examples
- **Docs:** https://reactflow.dev/learn

Shadcn UI (for surrounding UI)

- **Website:** https://ui.shadcn.com/
- **Components:** https://ui.shadcn.com/docs/components
- **Installation:** https://ui.shadcn.com/docs/installation

⏱️ TIME SAVINGS COMPARISON

Approach	Gantt Time	Precedence Time	Total Time	Quality
Custom SVG/CSS	6-8 hours	6-8 hours	**12-16 hours**	Good
Libraries (Recommended)	2-3 hours	4-5 hours	**6-8 hours**	Excellent
Time Saved	3-5 hours	1-3 hours	**4-8 hours**	Better!

What you can do with saved time:

- ✅ More thorough testing
- ✅ Better error handling
- ✅ Code review and polish
- ✅ Documentation
- ✅ Sleep before deadline! 😊

✅ SUCCESS CRITERIA

Gantt Chart Must Show:

- ✅ All 15 WBS items as task bars
- ✅ Timeline from Jan 2025 to May 2026
- ✅ Deadline marker (May 15, 2026)
- ✅ Color coding: Red (critical), Green (negotiable), Gray (locked)
- ✅ Task durations based on commitments
- ✅ Dependencies visible (optional with gantt-task-react)

Precedence Diagram Must Show:

- ✅ All 15 WBS items as nodes
- ✅ Each node shows: WBS ID, Name, ES, EF
- ✅ Dependency arrows between tasks
- ✅ Critical path highlighted in red
- ✅ Interactive (zoom, pan)

- ✅ Clear layout (nodes don't overlap badly)

🎯 CONCLUSION

Bottom Line:

1. **DON'T** try to use Shadcn UI for the diagrams themselves
2. **DO** use Shadcn UI for tabs, cards, and layout
3. **DO** use `gantt-task-react` for Gantt Chart (saves 3-5 hours)
4. **DO** use `ReactFlow` for Precedence Diagram (saves 1-3 hours)
5. **TOTAL SAVINGS:** 4-8 hours of development time

With your tight deadline (tomorrow EOD) and 4 developers, using proven libraries is the SMART choice.

📞 NEXT STEPS

1. ✅ Get team agreement on using these libraries
2. ✅ Assign Developer 2 to Gantt (3 hours)
3. ✅ Assign Developer 3 to Precedence (5 hours)
4. ✅ Install libraries: `npm install gantt-task-react reactflow`
5. ✅ Start with documentation reading (30 min each)
6. ✅ Build components following code examples above
7. ✅ Test with real data
8. ✅ Ship by tomorrow EOD! 🚀

Good luck with your implementation!

Remember: Using the right tool for the job isn't cheating—it's smart engineering. These libraries exist so you don't have to reinvent the wheel. Focus your energy on making your core features excellent.

Document Status: Ready for Implementation **Last Updated:** December 16, 2025 **Authors:** Development Team + Claude Code **Project:** Nye Hædda Barneskole PM

Simulation