Frontend Developer Hiring Assignment

Timeline/Gantt View - Interactive UI Component Development

Overview

Welcome to the **Design System Component Library** frontend developer hiring challenge. This assignment evaluates your ability to build **production-grade, complex interactive components** that align with our design system architecture.

You will implement a sophisticated **Timeline/Gantt View** component from scratch using modern web technologies. Your component will be showcased via **Storybook** stories demonstrating all features and interactions.

Timeline

Estimated Time: 8-12 hours
Submission Deadline: As specified in your Internshala application

Submission Method

Storybook Required: Your component must be documented and demonstrated through Storybook stories showing all states, interactions, and variants.

Objective

Build a fully functional **Timeline/Gantt View** - a horizontal time-based project visualization with task dependencies and resource allocation.

Your implementation should demonstrate:

- **Production-quality code architecture**
- **Enterprise-grade UI/UX patterns**
- **Performance optimization techniques**
- **Accessibility-first approach**
- **Scalable component design**

Technology Stack

Required Technologies

Technology	Purpose	Version
TypeScript	Type-safe development	^5.0.0
React	Component framework	^18.0.0
Tailwind CSS	Utility-first styling	^3.0.0
Vite or **Next.js**	Build tooling	Latest stable

Explicitly Forbidden

Component Libraries: Radix UI, Shadcn, Headless UI, MUI, Ant Design, Chakra UI, Mantine

CSS-in-JS: styled-components, emotion, vanilla-extract, stitches

UI Generators: Lovable, Locofy, TeleportHQ, Uizard, Builder.io

Timeline Libraries: vis-timeline, react-gantt-chart, bryntum, dhtmlxGantt

Pre-built Timeline Components: Any library that provides ready-made timeline/gantt components

Allowed Utilities

date-fns or **dayjs** (date manipulation only)

clsx or **classnames** (conditional class management)

zustand or **jotai** (lightweight state management)

framer-motion (animations - bonus only)

@dnd-kit/core (low-level drag primitives only, not pre-built components)

Storybook (required for component documentation)

> ***Important:*** *If you use `@dnd-kit`, you must implement your own drag logic and visual feedback. Simply wrapping pre-built hooks without custom implementation will be considered non-compliant.*

Storybook Requirements

Your Storybook stories must include:

- **Default** - Basic timeline with sample tasks
- **Empty State** - Timeline with no tasks
- **With Dependencies** - Timeline showing task dependencies
- **Multiple View Modes** - Day/Week/Month view demonstrations
- **Interactive Demo** - Fully functional drag-and-drop and resize
- **Mobile View** - Responsive layout demonstration
- **Accessibility** - Keyboard navigation demonstration

Required Project Structure

```
timeline-component/
│
├── README.md                        # Documentation
├── package.json                     # Dependencies
├── tsconfig.json                    # TypeScript config
├── tailwind.config.js               # Tailwind customization
├── .storybook/                      # Storybook configuration
│   ├── main.ts
│   └── preview.ts
│
└── src/
    ├── components/
    │   ├── Timeline/
    │   │   ├── TimelineView.tsx       # Main component
    │   │   ├── TimelineView.stories.tsx # Storybook stories
    │   │   ├── TimelineView.types.ts
    │   │   ├── TimelineGrid.tsx
    │   │   ├── TimelineRow.tsx
    │   │   ├── TaskBar.tsx
    │   │   ├── DependencyLine.tsx
    │   │   └── TaskDetailSidebar.tsx
    │   │
    │   └── primitives/                # Reusable UI elements
    │       ├── Button.tsx
    │       ├── Modal.tsx
    │       └── Slider.tsx
    │
    ├── hooks/
    │   ├── useTimeline.ts
    │   ├── useDragAndDrop.ts
    │   ├── useZoom.ts
    │   └── useScrollSync.ts
    │
    ├── utils/                         # Pure utility functions
    │   ├── date.utils.ts
    │   ├── position.utils.ts
    │   ├── dependency.utils.ts
    │   ├── validation.utils.ts
    │   └── formatting.utils.ts
    │
    ├── types/                         # TypeScript definitions
    │   └── timeline.types.ts
    │
    ├── constants/                     # Configuration constants
    │   └── timeline.constants.ts
    │
    └── styles/                        # Global styles
        ├── globals.css                # Base styles + Tailwind imports
        └── animations.css             # Custom animations
```

Design Requirements

Visual Design Principles

Your implementation should follow **modern SaaS product design patterns**:

1. **Clean & Minimal** - Remove visual noise, focus on content
2. **Consistent Spacing** - Use Tailwind's spacing scale (4px base unit)
3. **Clear Hierarchy** - Typography and color establish importance
4. **Subtle Interactions** - Micro-animations provide feedback
5. **Purposeful Color** - Use color to communicate state and action

Tailwind Configuration

Extend Tailwind with design tokens that align with our system:

```
// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: {
          50: '#f0f9ff',
          100: '#e0f2fe',
          500: '#0ea5e9',
          600: '#0284c7',
          700: '#0369a1',
        },
        neutral: {
          50: '#fafafa',
          100: '#f4f4f5',
          200: '#e4e4e7',
          300: '#d4d4d8',
          700: '#3f3f46',
          900: '#18181b',
        },
      },
      spacing: {
        18: '4.5rem',
        88: '22rem',
      },
      borderRadius: {
        'xl': '0.75rem',
      },
    },
  },
}
```

Responsive Breakpoints

Breakpoint	Width	Target Device	Layout Behavior
`sm`	640px+	Large mobile	Stack columns, expand cards
`md`	768px+	Tablet	2-column layouts, side panels
`lg`	1024px+	Desktop	Full multi-column, split views
`xl`	1280px+	Large desktop	Max width containers, sidebars

Timeline/Gantt View Detailed Requirements

Core Features

1. Data Structure

```
interface TimelineTask {
  id: string;
  title: string;
  startDate: Date;
  endDate: Date;
  progress: number; // 0-100
  assignee?: string;
  rowId: string; // which row/resource this belongs to
  dependencies?: string[]; // IDs of tasks this depends on
  color?: string;
  isMilestone?: boolean;
}

interface TimelineRow {
  id: string;
  label: string;
  avatar?: string;
  tasks: string[]; // task IDs assigned to this row
}

interface TimelineViewProps {
  rows: TimelineRow[];
  tasks: Record<string, TimelineTask>;
  startDate: Date;
  endDate: Date;
  viewMode: 'day' | 'week' | 'month';
  onTaskUpdate: (taskId: string, updates: Partial<TimelineTask>) => void;
  onTaskMove: (taskId: string, newRowId: string, newStartDate: Date) => void;
}
```

2. Timeline Grid Structure

Left Panel (Fixed):

- Row labels with avatars (200px wide)
- Sticky during horizontal scroll
- Expand/collapse row groups (bonus)

Right Panel (Scrollable):

- Time axis header showing dates
- Grid lines for each time unit
- Task bars positioned absolutely
- Current date indicator (vertical line)

3. Time Scale Requirements

View Mode	Time Unit	Column Width	Label Format
Day	1 day	40px	"Mon 24"
Week	1 week	80px	"Week 43" or "Oct 24"
Month	1 month	120px	"Oct 2024"

4. Task Bar Rendering

```
// Calculate position and width based on dates
const startPx = calculatePosition(task.startDate, viewStartDate, pixelsPerDay);
const widthPx = calculateDuration(task.startDate, task.endDate, pixelsPerDay);

<div
```

```
  className="absolute rounded shadow-sm cursor-move hover:shadow-lg transition-shadow"
  style={{
    left: `${startPx}px`,
    width: `${widthPx}px`,
    top: '8px',
    height: task.isMilestone ? '24px' : '32px',
    backgroundColor: task.color || '#0ea5e9',
  }}
>
  <div className="flex items-center justify-between h-full px-2">
    <span className="text-xs font-medium text-white truncate">
      {task.title}
    </span>
    {!task.isMilestone && (
      <span className="text-xs text-white opacity-75">
        {task.progress}%
      </span>
    )}
  </div>

  {/* Progress bar overlay */}
  {!task.isMilestone && task.progress > 0 && (
    <div
      className="absolute bottom-0 left-0 h-1 bg-white opacity-40 rounded-b"
      style={{ width: `${task.progress}%` }}
    />
  )}

  {/* Resize handles */}
  <div className="absolute left-0 top-0 bottom-0 w-1 cursor-ew-resize hover:bg-white
opacity-0 hover:opacity-50" />
  <div className="absolute right-0 top-0 bottom-0 w-1 cursor-ew-resize hover:bg-white
opacity-0 hover:opacity-50" />
</div>
```

5. Interactive Behaviors

Action	Expected Behavior
Drag task bar	Move to new row and/or date range
Resize left edge	Adjust start date
Resize right edge	Adjust end date
Click task bar	Open task detail modal/sidebar
Hover task bar	Show tooltip with dates and assignee
Scroll timeline	Smooth horizontal pan, load more dates dynamically
Zoom in/out	Change view mode (day ↔ week ↔ month)

6. Dependency Lines

When a task has dependencies, draw connecting lines:

- Start from predecessor's end point
- Arrow points to dependent task's start point
- Lines should route around tasks (use SVG or Canvas)
- Highlight dependency chain on hover

```
// Simplified SVG dependency line
<svg className="absolute inset-0 pointer-events-none" style={{ zIndex: 1 }}>
  <defs>
    <marker id="arrowhead" markerWidth="10" markerHeight="10" refX="9" refY="3"
orient="auto">
      <polygon points="0 0, 10 3, 0 6" fill="#94a3b8" />
    </marker>
  </defs>
  <path
    d={`M ${x1} ${y1} L ${x2} ${y2}`}
    stroke="#94a3b8"
    strokeWidth="2"
    fill="none"
    markerEnd="url(#arrowhead)"
  />
</svg>
```

7. Current Date Indicator

- Vertical red line spanning full timeline height
- Label showing "Today" at top
- Automatically scroll to today on initial load
- Update position dynamically if view is kept open

8. Task Detail Sidebar

When clicking a task, open slide-out panel showing:

- Task name (editable)
- Date range picker (start/end)
- Progress slider (0-100%)
- Assignee selector
- Dependencies selector (search other tasks)
- Notes/description textarea
- Activity log (bonus)
- Delete task button

9. Advanced Features

- **Milestones:** Diamond-shaped markers at specific dates
- **Task Groups:** Collapsible summary bars for grouped tasks (bonus)
- **Baseline Comparison:** Show original planned dates vs current (bonus)
- **Critical Path:** Highlight tasks that affect project end date (bonus)
- **Zooming:** Pinch or mouse wheel to zoom time scale

10. Responsive Behavior

- **Desktop:** Full side-by-side panel + timeline
- **Tablet:** Narrower left panel (150px), smaller column widths

- **Mobile:** Vertical list view with swipeable task cards, date headers

---

▯ Accessibility Requirements

All implementations **must** meet WCAG 2.1 AA standards:

Keyboard Navigation

Key	Action
`Tab`	Move focus between interactive elements
`Shift + Tab`	Move focus backwards
`Enter` / `Space`	Activate focused element or select task
`Escape`	Close modal or cancel action
`Arrow Keys`	Navigate between tasks or rows
`Home` / `End`	Jump to first/last task
`+` / `-`	Zoom in/out

ARIA Implementation

Required ARIA attributes:

```
// Example Task Bar
<div
  role="button"
  tabIndex={0}
  aria-label={`${task.title}. From ${formatDate(task.startDate)} to
${formatDate(task.endDate)}. Progress: ${task.progress}%. Press Enter to edit.`}
  aria-describedby={`task-${task.id}-details`}
  onKeyDown={handleKeyDown}
>
  {/* task bar content */}
</div>

// Example Timeline Row
<div
  role="region"
  aria-label={`${row.label} timeline. ${row.tasks.length} tasks.`}
>
  {/* row content */}
</div>

// Example Sidebar
<aside
  role="complementary"
  aria-label="Task details"
  aria-hidden={!isOpen}
>
```

```
  {/* sidebar content */}
</aside>
```

Visual Accessibility

- All interactive elements must have `:focus-visible` styles
- Color contrast ratio minimum 4.5:1 for text
- Focus indicators must be clearly visible (not `outline: none` without replacement)
- Text must be resizable up to 200% without loss of functionality

---

Performance Requirements

Your implementation will be tested for performance under stress conditions.

Performance Benchmarks

Metric	Target	Measurement
Initial Render	< 300ms	Time to interactive
Drag Response	< 16ms	Frame time during drag
Scroll Performance	60 FPS	Smooth scrolling
Large Dataset	Handle 100+ tasks	No visible lag
Bundle Size	< 200kb (gzipped)	Production build

Optimization Techniques

Required:

1. Use `React.memo()` for expensive components
2. Implement virtualization for rows (if >20 rows)
3. Use Canvas or SVG efficiently for dependency lines
4. Lazy load sidebar and detail views
5. Use `useCallback` and `useMemo` appropriately
6. Throttle scroll and resize events

Example Position Calculation:

```
// Memoized position calculation
const calculateTaskPosition = useMemo(() => {
  const msPerDay = 1000 * 60 * 60 * 24;
  const viewStartMs = viewStartDate.getTime();
  const taskStartMs = task.startDate.getTime();
  const taskEndMs = task.endDate.getTime();

  const daysSinceStart = (taskStartMs - viewStartMs) / msPerDay;
  const durationDays = (taskEndMs - taskStartMs) / msPerDay;

  return {
    left: daysSinceStart * pixelsPerDay,
    width: durationDays * pixelsPerDay,
  };
}, [task.startDate, task.endDate, viewStartDate, pixelsPerDay]);
```

▯ Code Quality Standards

TypeScript Standards

1. Strict Mode Enabled

```
// tsconfig.json
{
  "compilerOptions": {
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true
  }
}
```

2. Comprehensive Type Definitions

- No `any` types (use `unknown` if needed)
- Interface over type aliases for object shapes
- Proper generic constraints
- Discriminated unions for complex states

3. Example Type Safety

```
// Good ▯
interface TaskPosition {
  left: number;
  width: number;
  top: number;
  height: number;
}

interface DependencyConnection {
  from: { x: number; y: number };
  to: { x: number; y: number };
  taskId: string;
}

// Bad ▯
interface TaskPosition {
  left: any;
  width: any;
}
```

Code Organization

1. Component Structure

```
// TaskBar.tsx

import React from 'react';
import { TimelineTask } from '@/types/timeline.types';
import { formatDate } from '@/utils/date.utils';

interface TaskBarProps {
  task: TimelineTask;
  position: { left: number; width: number };
  onDragStart: (taskId: string) => void;
  onDragEnd: () => void;
  onClick: (task: TimelineTask) => void;
}

export const TaskBar: React.FC<TaskBarProps> = ({
  task,
  position,
  onDragStart,
  onDragEnd,
  onClick,
}) => {
  // Component logic

  return (
    // JSX
  );
};
```

2. Custom Hooks Pattern

```
// useTimeline.ts

import { useState, useCallback, useMemo } from 'react';

interface TimelineState {
  viewMode: 'day' | 'week' | 'month';
  startDate: Date;
  endDate: Date;
  pixelsPerDay: number;
}

export const useTimeline = (initialDate: Date = new Date()) => {
  const [state, setState] = useState<TimelineState>({
    viewMode: 'week',
    startDate: new Date(initialDate.getFullYear(), initialDate.getMonth(), 1),
    endDate: new Date(initialDate.getFullYear(), initialDate.getMonth() + 3, 0),
    pixelsPerDay: 40,
  });

  const zoomIn = useCallback(() => {
    setState(prev => {
      if (prev.viewMode === 'month') return { ...prev, viewMode: 'week', pixelsPerDay:
```

```
80 };
      if (prev.viewMode === 'week') return { ...prev, viewMode: 'day', pixelsPerDay:
40 };
      return prev;
    });
  }, []);

  const zoomOut = useCallback(() => {
    setState(prev => {
      if (prev.viewMode === 'day') return { ...prev, viewMode: 'week', pixelsPerDay:
80 };
      if (prev.viewMode === 'week') return { ...prev, viewMode: 'month', pixelsPerDay:
120 };
      return prev;
    });
  }, []);

  const scrollToToday = useCallback(() => {
    const today = new Date();
    setState(prev => ({
      ...prev,
      startDate: new Date(today.getFullYear(), today.getMonth() - 1, 1),
      endDate: new Date(today.getFullYear(), today.getMonth() + 2, 0),
    }));
  }, []);

  return {
    ...state,
    zoomIn,
    zoomOut,
    scrollToToday,
  };
};
```

3. Utility Function Pattern

```
// position.utils.ts

/**
 * Calculate pixel position from date
 */
export const calculatePosition = (
  date: Date,
  startDate: Date,
  pixelsPerDay: number
): number => {
  const msPerDay = 1000 * 60 * 60 * 24;
  const daysSinceStart = (date.getTime() - startDate.getTime()) / msPerDay;
  return Math.round(daysSinceStart * pixelsPerDay);
};

/**
```

```
 * Calculate duration in pixels
 */
export const calculateDuration = (
  startDate: Date,
  endDate: Date,
  pixelsPerDay: number
): number => {
  const msPerDay = 1000 * 60 * 60 * 24;
  const durationDays = (endDate.getTime() - startDate.getTime()) / msPerDay;
  return Math.round(durationDays * pixelsPerDay);
};

/**
 * Calculate date from pixel position
 */
export const calculateDateFromPosition = (
  position: number,
  startDate: Date,
  pixelsPerDay: number
): Date => {
  const days = Math.round(position / pixelsPerDay);
  const result = new Date(startDate);
  result.setDate(result.getDate() + days);
  return result;
};

/**
 * Generate time scale labels
 */
export const generateTimeScale = (
  startDate: Date,
  endDate: Date,
  viewMode: 'day' | 'week' | 'month'
): Array<{ date: Date; label: string }> => {
  const scale: Array<{ date: Date; label: string }> = [];
  const current = new Date(startDate);

  while (current <= endDate) {
    if (viewMode === 'day') {
      scale.push({
        date: new Date(current),
        label: current.toLocaleDateString('en-US', { weekday: 'short', day: 'numeric'
}),
      });
      current.setDate(current.getDate() + 1);
    } else if (viewMode === 'week') {
      scale.push({
        date: new Date(current),
        label: `Week ${getWeekNumber(current)}`,
      });
      current.setDate(current.getDate() + 7);
    } else {
      scale.push({
        date: new Date(current),
        label: current.toLocaleDateString('en-US', { month: 'short', year: 'numeric'
}),
      });
      current.setMonth(current.getMonth() + 1);
    }
  }

  return scale;
};

/**
 * Get week number
 */
const getWeekNumber = (date: Date): number => {
  const d = new Date(Date.UTC(date.getFullYear(), date.getMonth(), date.getDate()));
  const dayNum = d.getUTCDay() || 7;
  d.setUTCDate(d.getUTCDate() + 4 - dayNum);
  const yearStart = new Date(Date.UTC(d.getUTCFullYear(), 0, 1));
  return Math.ceil((((d.getTime() - yearStart.getTime()) / 86400000) + 1) / 7);
};
```

4. Dependency Calculation

```
// dependency.utils.ts

interface DependencyLine {
  x1: number;
  y1: number;
  x2: number;
  y2: number;
  fromTaskId: string;
  toTaskId: string;
}

/**
 * Calculate dependency line coordinates
 */
export const calculateDependencyLine = (
  fromTask: TimelineTask,
  toTask: TimelineTask,
  fromPosition: { left: number; width: number; top: number },
  toPosition: { left: number; width: number; top: number }
): DependencyLine => {
  // Start from end of predecessor task
  const x1 = fromPosition.left + fromPosition.width;
  const y1 = fromPosition.top + 16; // middle of task bar

  // End at start of dependent task
  const x2 = toPosition.left;
  const y2 = toPosition.top + 16;
  return {
    x1,
    y1,
    x2,
    y2,
    fromTaskId: fromTask.id,
    toTaskId: toTask.id,
  };
};

/**
 * Get all dependencies for a task
 */
export const getTaskDependencies = (
  taskId: string,
  tasks: Record<string, TimelineTask>
): string[] => {
  const task = tasks[taskId];
  return task?.dependencies || [];
};

/**
 * Get all tasks that depend on this task
 */
export const getDependentTasks = (
  taskId: string,
  tasks: Record<string, TimelineTask>
): string[] => {
  return Object.values(tasks)
    .filter(task => task.dependencies?.includes(taskId))
    .map(task => task.id);
};
```

Testing (Bonus)

If you include tests, follow these patterns:

```
// TaskBar.test.tsx

import { render, screen, fireEvent } from '@testing-library/react';
import { TaskBar } from './TaskBar';

describe('TaskBar', () => {
  const mockTask = {
    id: 'task-1',
    title: 'Test Task',
    startDate: new Date(2024, 0, 1),
    endDate: new Date(2024, 0, 5),
    progress: 60,
    rowId: 'row-1',
  };
  const mockPosition = { left: 100, width: 200 };

  it('renders task title correctly', () => {
    render(<TaskBar task={mockTask} position={mockPosition} onDragStart={() => {}}
onDragEnd={() => {}} onClick={() => {}} />);
    expect(screen.getByText('Test Task')).toBeInTheDocument();
  });

  it('displays progress percentage', () => {
    render(<TaskBar task={mockTask} position={mockPosition} onDragStart={() => {}}
onDragEnd={() => {}} onClick={() => {}} />);
    expect(screen.getByText('60%')).toBeInTheDocument();
  });

  it('calls onClick when clicked', () => {
    const handleClick = jest.fn();
    render(<TaskBar task={mockTask} position={mockPosition} onDragStart={() => {}}
onDragEnd={() => {}} onClick={handleClick} />);

    fireEvent.click(screen.getByText('Test Task'));
    expect(handleClick).toHaveBeenCalledWith(mockTask);
  });
});
```

▯ Submission Requirements

1. Repository Setup

Your GitHub repository must include:

```
▯ README.md with complete documentation
▯ package.json with all dependencies (including Storybook)
▯ .gitignore (exclude node_modules, storybook-static)
▯ Source code in /src following required structure
▯ Storybook configuration in .storybook/
▯ Component stories (.stories.tsx files)
▯ At least 5 meaningful commits
▯ Deployed Storybook (Chromatic/Vercel/Netlify)
▯ NO node_modules folder
```

2. Storybook Documentation

Your Storybook must include:

Required Stories:

1. **Default** - Basic timeline with sample tasks
2. **Empty** - Empty timeline state
3. **With Dependencies** - Timeline showing task dependencies
4. **View Modes** - Day/Week/Month view demonstrations
5. **Interactive Playground** - Fully functional with controls

3. README.md Format

~~~~markdown
# Timeline/Gantt View Component

## □ Live Storybook
[Your Deployed Storybook URL]

## □ Installation
\`\`\`bash
npm install
npm run storybook
\`\`\`

## □ Architecture
[Brief explanation]

## □ Features
- [x] Timeline grid with time scale
- [x] Task drag-and-drop
- [x] Task resizing
- [x] Dependencies
- [x] View mode switching

## □ Storybook Stories
List your stories here

## □ Technologies
- React + TypeScript
- Tailwind CSS
- Storybook

## □ Contact
[Your email]
~~~~

4. Git Commit Guidelines

Follow conventional commit format:

```
feat: add timeline grid rendering
feat: implement task drag and drop
feat: add dependency line visualization
fix: resolve date calculation bug for month boundaries
style: improve mobile responsiveness
refactor: extract position calculation utilities
docs: update storybook stories
perf: implement memoization for position calculations
```

5. Storybook Deployment

Deploy your Storybook to:

- **Chromatic** (recommended)
- **Vercel**
- **Netlify**
- **GitHub Pages**

Include the deployed link prominently in your README.

Evaluation Rubric

Your submission will be scored across these dimensions:

1. Functionality (30 points)

Criteria	Points	Description
Core features work correctly	15	All required interactions function without errors
Edge cases handled	8	Validates inputs, handles empty states, prevents crashes
Data persistence works	7	State updates correctly, can add/edit/delete/move tasks

2. Code Quality (25 points)

Criteria	Points	Description
TypeScript usage	8	Proper types, no `any`, strict mode enabled
Component architecture	8	Clean separation, reusable components, single responsibility
Code organization	5	Logical folder structure, proper imports
Comments & docs	4	Code is self-documenting with strategic comments

3. UI/UX Design (20 points)

Criteria	Points	Description
Visual polish	8	Professional appearance, consistent styling
Interaction feedback	6	Hover states, drag feedback, smooth transitions
Responsive design	6	Works seamlessly on mobile, tablet, desktop

4. Accessibility (10 points)

Criteria	Points	Description
Keyboard navigation	4	All features accessible via keyboard
ARIA implementation	3	Proper labels, roles, live regions
Focus management	3	Logical focus order, visible focus indicators

5. Performance (10 points)

Criteria	Points	Description
Optimized rendering	5	No unnecessary re-renders, uses memoization

Handles large datasets	3	Smooth performance with 100+ tasks
Bundle size	2	Production build under 200kb gzipped

6. Documentation (5 points)

Criteria	Points	Description
Storybook stories completeness	3	All required stories implemented
README quality	2	Clear setup and architecture explanation

Bonus Points (up to +15)

- Interactive story controls (+3)
- Dark mode implementation (+3)
- Additional stories beyond requirements (+3)
- Accessibility story with keyboard demo (+3)
- Performance optimization documentation (+3)

Total Possible: 100 points (115 with bonus)

Passing Score: 70 points

Disqualification Criteria

Your submission will be **immediately rejected** if any of these violations are found:

1. **Use of forbidden libraries:**
    - Component libraries (Radix, Shadcn, MUI, Ant Design, etc.)
    - Pre-built timeline/gantt components
    - CSS-in-JS solutions (styled-components, emotion)
2. **AI-generated UI:**
    - Code generated by Lovable, Bolt, v0, Locofy, etc.
    - Entire components copy-pasted from ChatGPT/Claude/Copilot without understanding
    - (Note: Using AI for debugging or learning is acceptable, but the final code must be your own)
3. **Plagiarism:**
    - Code copied from tutorials, Stack Overflow, or GitHub without attribution
    - Using paid templates or starter kits
4. **Non-functional submission:**
    - Cannot be run locally
    - Missing core required features
    - Critical bugs that crash the application
    - No deployed Storybook link provided
    - Storybook doesn't build or has critical errors

5. **Incomplete submission:**
    - No README
    - No source code
    - Repository is private and access not granted

Tips for Success

Before You Start

1. **Study reference implementations:**
    - Monday.com
    - TeamGantt
    - Microsoft Project
    - Asana Timeline
    - Don't copy code, but understand interaction patterns
2. **Set up Storybook first:**
    - Initialize Storybook before building components
    - Build component and story together, not sequentially
    - Use Storybook for visual testing during development
3. **Set up your environment:**
    - Use VS Code with Tailwind IntelliSense extension
    - Install ESLint and Prettier for consistent formatting
    - Set up TypeScript strict mode from the start

During Development

1. **Start with basic component structure in Storybook:**
    - Create default story showing basic rendering
    - Build static layout with mock data
    - Test responsive behavior in Storybook viewports
2. **Implement features incrementally:**
    - Day 1-2: Component structure, basic story, grid layout
    - Day 3-4: Task rendering, empty state story, positioning logic
    - Day 5-6: Interactions (drag, resize), interactive story, mobile story
    - Day 7-8: Polish, accessibility story, deployment
3. **Test edge cases via stories:**
    - Create stories for empty states, large datasets, mobile views
    - Use Storybook controls to test different prop combinations
    - Verify accessibility with keyboard navigation in stories

Common Pitfalls to Avoid

Building full app: You only need the component, not a complete application
Skipping Storybook: Stories are not optional—they're the submission format
Over-engineering: Don't add Redux if React Context is sufficient
Styling inconsistency: Stick to Tailwind classes, avoid inline styles

Accessibility as afterthought: Build it in from the start
Ignoring edge cases: Empty states, loading states, error states
Performance issues: Test with larger datasets early

Learning Resources

If you need to brush up on specific skills:

Storybook

- Storybook for React Tutorial
- Writing Stories
- Storybook Controls

Date Manipulation

- date-fns documentation
- Day.js documentation

SVG/Canvas

- MDN - SVG Tutorial
- MDN - Canvas API

Drag and Drop

- MDN - HTML Drag and Drop API
- @dnd-kit documentation (if using)

Accessibility

- WCAG 2.1 Guidelines
- A11y Project Checklist

TypeScript Patterns

- React TypeScript Cheatsheet

Tailwind Best Practices

- Tailwind CSS Documentation
- Refactoring UI

FAQ

Q: Do I need to build a full application or just the component?
A: Just the component! Build it within Storybook—no full app scaffolding needed.

Q: How many Storybook stories are required?
A: Minimum 7 stories covering: Default, Empty State, Large Dataset, Mobile View, Interactive Demo, Week View, and one Accessibility story.

Q: Can I use Storybook addons?
A: Yes! Use addons like Controls, Actions, Viewport, and a11y to enhance your stories.

Q: Can I use a UI component library for just the sidebar or date picker?
A: No. Build all UI components from scratch. This tests your fundamentals.

Q: Can I use an icon library like Heroicons or Lucide?
A: Yes, icon libraries are acceptable. You can also use SVG icons directly.

Q: Should I use SVG or Canvas for dependency lines?
A: Either is acceptable. SVG is generally easier to work with for this use case, but Canvas may perform better with many dependencies.

Q: Is it okay to use ChatGPT/Copilot for help?
A: Yes, but you must understand every line of code you submit. We will ask you to explain your implementation in detail during the interview.

Q: How important is the visual design vs functionality?
A: Both matter. Aim for 60% functionality, 40% design. The component must work perfectly AND look professional.

Q: Where should I deploy?
A: Deploy your Storybook to Chromatic (recommended), Vercel, Netlify, or GitHub Pages. Include the live link in your README.

Final Checklist Before Submission

Use this checklist to verify your submission:

Storybook Stories

- ☐ Default Story: Basic timeline with sample tasks
- ☐ Empty State Story: No tasks added yet
- ☐ Large Dataset Story: 30+ tasks across multiple rows
- ☐ Mobile View Story: Responsive behavior demonstration
- ☐ Interactive Demo Story: Drag, resize, edit functionality
- ☐ Week View Story: Shows week-level time scale
- ☐ Accessibility Story: Keyboard navigation demonstration
- ☐ Storybook builds without errors
- ☐ All stories render correctly

Functionality

- ☐ Timeline displays time scale correctly
- ☐ Task bars render at correct positions
- ☐ Can drag tasks to new dates/rows
- ☐ Can resize tasks by edges
- ☐ View mode switching works (day/week/month)
- ☐ Current date indicator displays
- ☐ Task detail sidebar opens on click
- ☐ Can edit task details
- ☐ Dependency lines render (bonus)
- ☐ No console errors in browser
- ☐ Responsive on mobile, tablet, desktop

Code Quality

- [ ] TypeScript strict mode enabled with no errors
- [ ] All components properly typed
- [ ] No `any` types used
- [ ] ESLint passes with no errors
- [ ] Code formatted consistently
- [ ] Follows required folder structure

Accessibility

- [ ] All interactive elements keyboard accessible
- [ ] ARIA labels on custom controls
- [ ] Focus indicators visible
- [ ] Tested with keyboard-only navigation

Documentation

- [ ] README includes deployed Storybook link
- [ ] Setup instructions clear and tested
- [ ] Architecture decisions explained
- [ ] Component usage examples provided

Repository

- [ ] Repository is public: `timeline-component-[yourname]`
- [ ] .gitignore excludes node_modules and build files
- [ ] At least 5 meaningful commits
- [ ] No sensitive data in code

Deployment

- [ ] Storybook is deployed and accessible
- [ ] Deployed link works correctly
- [ ] All stories render in production build

Constraints Compliance

- [ ] No forbidden UI libraries used
- [ ] Only Tailwind CSS for styling (no CSS-in-JS)
- [ ] No AI-generated UI tools used
- [ ] No pre-built timeline libraries used

Submission Process

Step 1: Complete Your Implementation

Ensure all required Storybook stories work and code is clean.

Step 2: Deploy Your Storybook

Deploy to Chromatic (recommended), Vercel, Netlify, or GitHub Pages and verify all stories render correctly.

Step 3: Create GitHub Repository

- Repository name: `timeline-component-[yourname]`
- Make repository **public**

Step 4: Verify Locally

```
# Fresh clone test
git clone [your-repo-url]
cd timeline-component-[yourname]
npm install
npm run storybook
# Verify all stories work
```

Step 5: Submit via Internshala

Submit the following through the Internshala portal:

- Link to your GitHub repository
- Link to deployed Storybook
- Brief description of your implementation (2-3 paragraphs)

▯ Good Luck!

We're excited to see what you build! This assignment is your chance to showcase not just your coding skills, but your component design thinking, attention to detail, and documentation quality.

Remember: **Quality over quantity.** A polished, well-documented component with complete Storybook stories will score higher than a buggy attempt at every possible feature.

We're rooting for you! ▯

Appendix A: Sample Data Structure

Timeline View Sample Data

```
const sampleRows: TimelineRow[] = [
  { id: 'row-1', label: 'Frontend Team', avatar: '/avatars/frontend.png', tasks:
['task-1', 'task-2'] },
  { id: 'row-2', label: 'Backend Team', avatar: '/avatars/backend.png', tasks: ['task-
3'] },
  { id: 'row-3', label: 'Design Team', avatar: '/avatars/design.png', tasks: ['task-
4'] },
];

const sampleTasks: Record<string, TimelineTask> = {
  'task-1': {
    id: 'task-1',
    title: 'UI Component Development',
    startDate: new Date(2024, 0, 1),
    endDate: new Date(2024, 0, 15),
    progress: 60,
    assignee: 'Frontend Team',
    rowId: 'row-1',
    dependencies: [],
    color: '#3b82f6',
    isMilestone: false,
  },
  'task-2': {
    id: 'task-2',
    title: 'Integration Testing',
    startDate: new Date(2024, 0, 16),
    endDate: new Date(2024, 0, 25),
    progress: 0,
    assignee: 'Frontend Team',
    rowId: 'row-1',
    dependencies: ['task-1', 'task-3'],
    color: '#3b82f6',
    isMilestone: false,
  },
  'task-3': {
    id: 'task-3',
    title: 'API Development',
    startDate: new Date(2024, 0, 1),
    endDate: new Date(2024, 0, 14),
    progress: 80,
    assignee: 'Backend Team',
    rowId: 'row-2',
    dependencies: [],
    color: '#10b981',
    isMilestone: false,
  },
  'task-4': {
    id: 'task-4',
    title: 'Design System Update',
    startDate: new Date(2024, 0, 5),
    endDate: new Date(2024, 0, 12),
    progress: 100,
    assignee: 'Design Team',
    rowId: 'row-3',
    dependencies: [],
    color: '#f59e0b',
    isMilestone: false,
  },
};
```

Appendix B: Tailwind Configuration Template

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        primary: {
          50: '#f0f9ff',
          100: '#e0f2fe',
          200: '#bae6fd',
          300: '#7dd3fc',
          400: '#38bdf8',
          500: '#0ea5e9',
          600: '#0284c7',
          700: '#0369a1',
          800: '#075985',
          900: '#0c4a6e',
        },
        neutral: {
          50: '#fafafa',
          100: '#f4f4f5',
          200: '#e4e4e7',
          300: '#d4d4d8',
          400: '#a1a1aa',
          500: '#71717a',
          600: '#52525b',
          700: '#3f3f46',
          800: '#27272a',
          900: '#18181b',
        },
        success: {
          50: '#f0fdf4',
          500: '#10b981',
          700: '#047857',
        },
        warning: {
          50: '#fffbeb',
          500: '#f59e0b',
          700: '#b45309',
        },
        error: {
          50: '#fef2f2',
          500: '#ef4444',
          700: '#b91c1c',
        },
      },
      fontFamily: {
        sans: ['Inter', 'system-ui', 'sans-serif'],
        mono: ['Fira Code', 'monospace'],
      },
      spacing: {
        18: '4.5rem',
        88: '22rem',
        112: '28rem',
        128: '32rem',
      },
      borderRadius: {
        '4xl': '2rem',
      },
      boxShadow: {
        'card': '0 1px 3px 0 rgb(0 0 0 / 0.1), 0 1px 2px -1px rgb(0 0 0 / 0.1)',
        'card-hover': '0 10px 15px -3px rgb(0 0 0 / 0.1), 0 4px 6px -4px rgb(0 0 0 /
0.1)',
        'modal': '0 20px 25px -5px rgb(0 0 0 / 0.1), 0 8px 10px -6px rgb(0 0 0 /
0.1)',
      },
      animation: {
        'fade-in': 'fadeIn 0.2s ease-in-out',
        'slide-up': 'slideUp 0.3s ease-out',
        'slide-down': 'slideDown 0.3s ease-out',
      },
      keyframes: {
        fadeIn: {
          '0%': { opacity: '0' },
          '100%': { opacity: '1' },
        },
        slideUp: {
          '0%': { transform: 'translateY(10px)', opacity: '0' },
          '100%': { transform: 'translateY(0)', opacity: '1' },
        },
        slideDown: {
          '0%': { transform: 'translateY(-10px)', opacity: '0' },
          '100%': { transform: 'translateY(0)', opacity: '1' },
        },
      },
    },
  },
  plugins: [],
}
```

End of Timeline/Gantt View Assignment Document

This assignment is part of Design System Component Library's hiring process. All submitted code remains your intellectual property.