Frontend Developer Hiring Assignment

Calendar 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 **Calendar 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 **Calendar View** - a fully interactive date/event management interface.

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
Calendar/Date Libraries: react-big-calendar, FullCalendar, react-calendar
Pre-built Calendar Components: Any library that provides ready-made calendar 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)
Storybook (required for component documentation)

Storybook Requirements

Your Storybook stories must include:

- **Default** - Current month with sample events
- **Empty State** - Calendar with no events
- **Week View** - Week view demonstration
- **With Many Events** - Month with 20+ events
- **Interactive Demo** - Fully functional event management
- **Mobile View** - Responsive layout demonstration
- **Accessibility** - Keyboard navigation demonstration

Required Project Structure

```
calendar-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/
    │   ├── Calendar/
    │   │   ├── CalendarView.tsx       # Main component
    │   │   ├── CalendarView.stories.tsx # Storybook stories
    │   │   ├── CalendarView.types.ts
    │   │   ├── MonthView.tsx
    │   │   ├── WeekView.tsx
    │   │   ├── CalendarCell.tsx
    │   │   └── EventModal.tsx
    │   │
    │   └── primitives/                # Reusable UI elements
    │       ├── Button.tsx
    │       ├── Modal.tsx
    │       └── Select.tsx
    │
    ├── hooks/
    │   ├── useCalendar.ts
    │   └── useEventManager.ts
    │
    ├── utils/
    │   ├── date.utils.ts
    │   └── event.utils.ts
    │
    └── styles/
        └── globals.css
```

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:

```js
// 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

Calendar View Detailed Requirements

Core Features

1. Calendar Grid Rendering

```typescript
interface CalendarEvent {
  id: string;
  title: string;
  description?: string;
  startDate: Date;
  endDate: Date;
  color?: string;
  category?: string;
}

interface CalendarViewProps {
  events: CalendarEvent[];
  onEventAdd: (event: CalendarEvent) => void;
  onEventUpdate: (id: string, updates: Partial<CalendarEvent>) => void;
  onEventDelete: (id: string) => void;
  initialView?: 'month' | 'week';
  initialDate?: Date;
}
```

2. Month View Requirements

- Display 42 cells (6 weeks × 7 days) to show complete weeks
- Gray out dates from adjacent months
- Highlight current day with distinct styling
- Show event count badge when >3 events exist in a day
- Clicking a day cell opens "Add Event" modal

- Clicking an event opens "Edit Event" modal

3. Week View Requirements

- Show 7-day horizontal layout with time slots (00:00 - 23:00)
- 30-minute or 1-hour interval grid lines
- Events positioned based on start time with height proportional to duration
- Handle overlapping events with side-by-side positioning
- Support drag-to-create new events directly on time slots

4. Interactive Behaviors

Action	Expected Behavior
Click empty cell	Open "Create Event" modal with pre-filled date
Click event	Open "Edit Event" modal with full details
Drag event	Move event to new date/time (show ghost preview)
Hover event	Display tooltip with time and description
Keyboard navigation	Arrow keys move focus between days, Enter opens selected
Multi-select	Shift+Click or Drag to select date range

5. Event Management Modal

Must include:

- Title input (required, max 100 chars)
- Description textarea (optional, max 500 chars)
- Start date/time picker
- End date/time picker (must be after start)
- Color picker (5-8 preset colors)
- Category dropdown (optional)
- Delete button (for existing events)
- Cancel and Save buttons

6. Navigation Controls

- Previous/Next month buttons
- "Today" quick-jump button
- Month/Year picker dropdown
- View toggle (Month ↔ Week)

7. Visual Requirements

```
// Example Month View Cell
<div className="border border-neutral-200 h-32 p-2 hover:bg-neutral-50 transition-colors cursor-pointer">
  <div className="flex justify-between items-start mb-1">
    <span className="text-sm font-medium text-neutral-900">{dayNumber}</span>
    {isToday && <span className="w-6 h-6 bg-primary-500 rounded-full text-white text-xs flex items-center justify-center">{dayNumber}</span>}
  </div>

  <div className="space-y-1 overflow-hidden">
```

```
    {events.slice(0, 3).map(event => (
      <div
        key={event.id}
        className="text-xs px-2 py-1 rounded truncate"
        style={{ backgroundColor: event.color }}
      >
        {event.title}
      </div>
    ))}

    {events.length > 3 && (
      <button className="text-xs text-primary-600 hover:underline">
        +{events.length - 3} more
      </button>
    )}
  </div>
</div>
```

8. Responsive Behavior

- **Desktop:** Full 7-column grid, sidebar for event list
- **Tablet:** Vertical scrolling, sticky header row
- **Mobile:** List view with expandable days, swipe between dates

---

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
`Escape`	Close modal or cancel action
`Arrow Keys`	Navigate grid cells or list items
`Home` / `End`	Jump to first/last item

ARIA Implementation

Required ARIA attributes:

```
// Example Calendar Cell
<div
  role="button"
  tabIndex={0}
  aria-label={`${monthName} ${day}, ${year}. ${eventCount} events.`}
  aria-pressed={isSelected}
  onKeyDown={handleKeyDown}
```

```
>
  {/* cell content */}
</div>

// Example Modal
<div
  role="dialog"
  aria-modal="true"
  aria-labelledby="modal-title"
  aria-describedby="modal-description"
>
  <h2 id="modal-title">Edit Event</h2>
  <div id="modal-description">Update event details below</div>
  {/* modal content */}
</div>
```

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
Search/Filter	< 100ms	Results update latency
Large Dataset	Handle 500+ events	No visible lag
Bundle Size	< 200kb (gzipped)	Production build

Optimization Techniques

Required:

1. Use `React.memo()` for expensive components
2. Implement virtualization for long lists (>50 items)
3. Debounce search and filter inputs
4. Lazy load modals and detail views
5. Use `useCallback` and `useMemo` appropriately

Example Optimization:

```
// Memoized calendar cell component
const CalendarCell = React.memo<CalendarCellProps>(({ date, events, isToday, onClick
}) => {
  const handleClick = useCallback(() => {
    onClick(date);
  }, [date, onClick]);

  const eventCount = useMemo(() => events.length, [events]);

  return (
    <div onClick={handleClick}>
      {/* cell content */}
    </div>
  );
});
```

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 EventFormData {
  title: string;
  startDate: Date;
  endDate: Date;
  description?: string;
}

type FormErrors = Partial<Record<keyof EventFormData, string>>;

// Bad
```

```
interface EventFormData {
  title: any;
  startDate: any;
  endDate: any;
  description: any;
}
```

Code Organization

1. Component Structure

```
// CalendarCell.tsx

import React from 'react';
import { CalendarEvent } from '@/types/calendar.types';
import { formatDate } from '@/utils/date.utils';

interface CalendarCellProps {
  date: Date;
  events: CalendarEvent[];
  isToday: boolean;
  isSelected: boolean;
  onClick: (date: Date) => void;
  onEventClick: (event: CalendarEvent) => void;
}

export const CalendarCell: React.FC<CalendarCellProps> = ({
  date,
  events,
  isToday,
  isSelected,
  onClick,
  onEventClick,
}) => {
  // Component logic

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

2. Custom Hooks Pattern

```
// useCalendar.ts

import { useState, useCallback } from 'react';

interface CalendarState {
  currentDate: Date;
  view: 'month' | 'week';
  selectedDate: Date | null;
}
```

```
export const useCalendar = (initialDate: Date = new Date()) => {
  const [state, setState] = useState<CalendarState>({
    currentDate: initialDate,
    view: 'month',
    selectedDate: null,
  });

  const goToNextMonth = useCallback(() => {
    setState(prev => ({
      ...prev,
      currentDate: new Date(prev.currentDate.getFullYear(),
prev.currentDate.getMonth() + 1, 1),
    }));
  }, []);

  const goToPreviousMonth = useCallback(() => {
    setState(prev => ({
      ...prev,
      currentDate: new Date(prev.currentDate.getFullYear(),
prev.currentDate.getMonth() - 1, 1),
    }));
  }, []);

  const goToToday = useCallback(() => {
    setState(prev => ({
      ...prev,
      currentDate: new Date(),
    }));
  }, []);

  return {
    ...state,
    goToNextMonth,
    goToPreviousMonth,
    goToToday,
  };
};
```

3. Utility Function Pattern

```
// date.utils.ts

/**
 * Calculates the number of days between two dates
 * @param start - Start date
 * @param end - End date
 * @returns Number of days (can be negative if end is before start)
 */
export const daysBetween = (start: Date, end: Date): number => {
  const msPerDay = 1000 * 60 * 60 * 24;
  const startMs = start.getTime();
  const endMs = end.getTime();
  return Math.floor((endMs - startMs) / msPerDay);
};

/**
 * Checks if two dates fall on the same day (ignores time)
 */
export const isSameDay = (date1: Date, date2: Date): boolean => {
  return (
    date1.getFullYear() === date2.getFullYear() &&
    date1.getMonth() === date2.getMonth() &&
    date1.getDate() === date2.getDate()
  );
};

/**
 * Gets all days in a month
 */
export const getDaysInMonth = (date: Date): Date[] => {
  const year = date.getFullYear();
  const month = date.getMonth();
  const daysCount = new Date(year, month + 1, 0).getDate();

  return Array.from({ length: daysCount }, (_, i) => new Date(year, month, i + 1));
};

/**
 * Gets the calendar grid (42 cells for month view)
 */
export const getCalendarGrid = (date: Date): Date[] => {
  const year = date.getFullYear();
  const month = date.getMonth();
  const firstDay = new Date(year, month, 1);
  const lastDay = new Date(year, month + 1, 0);

  const startDate = new Date(firstDay);
  startDate.setDate(startDate.getDate() - startDay.getDay());

  const grid: Date[] = [];
  for (let i = 0; i < 42; i++) {
    grid.push(new Date(startDate));
    startDate.setDate(startDate.getDate() + 1);
  }

  return grid;
};
```

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** - Current month with sample events
2. **Empty** - Empty calendar state
3. **Week View** - Week view with time slots
4. **Large Dataset** - Calendar with 20+ events
5. **Interactive Playground** - Fully functional with controls

3. README.md Format

````markdown
# Calendar View Component

## □ Live Storybook
[Your Deployed Storybook URL]

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

## □ Architecture
[Brief explanation]

## □ Features
- [x] Month/Week views
- [x] Event management
- [x] Responsive design
- [x] Keyboard accessibility

## □ 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 month view grid rendering
feat: implement event creation modal
fix: resolve date calculation bug
docs: update storybook stories
```

5. Storybook Deployment

Deploy your Storybook to:

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

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 events

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, loading indicators, 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	Virtualization or pagination for 100+ events
Bundle size	2	Production build under 200kb gzipped

6. Documentation (5 points)

Criteria	Points	Description
README completeness	3	All required sections filled out
Setup instructions	2	Clear steps to run locally

Bonus Points (up to +15)

- Unit tests with >70% coverage (+5)
- Advanced animations with Framer Motion (+3)
- Dark mode implementation (+3)
- LocalStorage persistence (+2)
- Additional features beyond requirements (+2)

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 calendar components (react-big-calendar, FullCalendar, etc.)
    - 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 link provided
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:**
    - Google Calendar
    - Apple Calendar
    - Outlook Calendar
    - Don't copy code, but understand interaction patterns
2. **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 static layout:**
    - Build the visual structure first without interactions
    - Perfect the responsive design before adding interactivity
    - Use placeholder data to test different scenarios
2. **Implement features incrementally:**
    - Day 1-2: Basic rendering and layout (month/week grid)
    - Day 3-4: Core interactions (click, navigation)
    - Day 5-6: Modals, forms, and event management
    - Day 7-8: Polish, accessibility, and deployment
3. **Test continuously:**

    - Test on actual mobile devices, not just browser dev tools
    - Use keyboard-only navigation to verify accessibility
    - Test with 50+ events to ensure performance

Common Pitfalls to Avoid

▯ **Over-engineering:** Don't add Redux if React Context is sufficient
▯ **Under-engineering:** Don't skip proper component abstraction
▯ **Styling inconsistency:** Stick to Tailwind classes, avoid inline styles
▯ **Accessibility as afterthought:** Build it in from the start
▯ **Poor mobile experience:** Design mobile-first, not desktop-only
▯ **Ignoring edge cases:** Empty states, loading states, error states

---

▯ Learning Resources

If you need to brush up on specific skills:

Date Manipulation

- date-fns documentation
- Day.js documentation

Accessibility

- WCAG 2.1 Guidelines
- A11y Project Checklist

TypeScript Patterns

- React TypeScript Cheatsheet

Tailwind Best Practices

- Tailwind CSS Documentation
- Refactoring UI

▯ FAQ

Q: Can I use a UI component library for just the modal 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: 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 app must work perfectly AND look professional.

Q: Can I deploy my assignment to Vercel/Netlify?
A: Yes, deployment is required! Include the live link in your submission.

Q: What if I can't complete all features in the estimated time?
A: Focus on core features first. A polished subset is better than a buggy full

implementation. Document what's missing in README.

Q: Can I add features beyond the requirements?
A: Yes, but only after completing all core requirements. Document bonus features clearly.

Final Checklist Before Submission

Use this checklist to verify your submission:

Functionality

- [ ] Month view displays 42-cell grid correctly
- [ ] Week view displays time slots correctly
- [ ] Can create new events
- [ ] Can edit existing events
- [ ] Can delete events
- [ ] Navigation controls work (prev/next/today)
- [ ] View toggle works (month ↔ week)
- [ ] No console errors in browser
- [ ] Forms validate correctly
- [ ] 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 all required sections
- [ ] Setup instructions tested on fresh clone
- [ ] Architecture decisions explained
- [ ] Known limitations documented
- [ ] Deployed link included and working

Repository

- [ ] Repository is public or access granted
- [ ] .gitignore excludes node_modules and build files
- [ ] At least 3 meaningful commits
- [ ] No sensitive data in code

Deployment

- [ ] Application is deployed and accessible
- [ ] Deployed link works correctly
- [ ] All features work 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 calendar libraries used

Submission Process

Step 1: Complete Your Implementation

Ensure all features work and code is clean.

Step 2: Deploy Your Application

Deploy to Vercel, Netlify, or GitHub Pages and verify it works.

Step 3: Create GitHub Repository

- Repository name: `calendar-view-[yourname]`
- Make repository **public**

Step 4: Verify Locally

```bash
# Fresh clone test
git clone [your-repo-url]
cd calendar-view-[yourname]
npm install
npm run dev
# Verify everything works
```

Step 5: Submit via Internshala

Submit the following through the Internshala portal:

- Link to your GitHub repository
- Link to deployed application
- Brief description of your implementation (3-4 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 product thinking, design sensibility, and attention to detail.

Remember: **Quality over quantity.** A polished, well-documented implementation of core features will score higher than a buggy attempt at every possible feature.

We're rooting for you! □

Appendix A: Sample Data Structure

Calendar View Sample Data

```
const sampleEvents: CalendarEvent[] = [
  {
    id: 'evt-1',
    title: 'Team Standup',
    description: 'Daily sync with the team',
    startDate: new Date(2024, 0, 15, 9, 0),
    endDate: new Date(2024, 0, 15, 9, 30),
    color: '#3b82f6',
    category: 'Meeting',
  },
  {
    id: 'evt-2',
    title: 'Design Review',
    description: 'Review new component designs',
    startDate: new Date(2024, 0, 15, 14, 0),
    endDate: new Date(2024, 0, 15, 15, 30),
    color: '#10b981',
    category: 'Design',
  },
  {
    id: 'evt-3',
    title: 'Client Presentation',
    startDate: new Date(2024, 0, 16, 10, 0),
    endDate: new Date(2024, 0, 16, 11, 30),
    color: '#f59e0b',
    category: 'Meeting',
  },
  {
    id: 'evt-4',
    title: 'Development Sprint',
    description: 'Sprint planning and task assignment',
    startDate: new Date(2024, 0, 17, 9, 0),
    endDate: new Date(2024, 0, 17, 17, 0),
    color: '#8b5cf6',
    category: 'Work',
  },
];
```

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 Calendar View Assignment Document

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