Design Systems in React

Building Component Libraries That Scale

React Paris Meetup #013

2025

Outline

What is a Design System?

Beyond a Component Library

A design system is:

- Design tokens (colors, spacing, typography)
- Reusable components
- Documentation and guidelines
- Design-development bridge
- Living, evolving standards

Why Build One?

- Consistency across products
- Faster development
- Better accessibility

What We'll Build

The Journey

1. Token-based design foundations
2. Component API design principles
3. Compound components pattern
4. Accessibility by default
5. Documentation with Storybook

The Foundation Layer

What Are Tokens?

```
{
  "color": {
    "primary": {
      "50": "#f0f9ff",
      "500": "#3b82f6",
      "900": "#1e3a8a"
    },
    "semantic": {
      "success": "{color.green.500}",
      "error": "{color.red.500}"
    }
  },
```

Style Dictionary: Token Pipeline

Configuration

```
// style-dictionary.config.js
module.exports = {
  source: ['tokens/**/*.json'],
  platforms: {
    css: {
      transformGroup: 'css',
      buildPath: 'dist/css/',
      files: [{
        destination: 'variables.css',
        format: 'css/variables'
      }]
    },
```

Generated Output

CSS Custom Properties

```
:root {
  --color-primary-50: #f0f9ff;
  --color-primary-500: #3b82f6;
  --color-primary-900: #1e3a8a;
  --color-semantic-success: var(--color-green-500);
  --color-semantic-error: var(--color-red-500);
  --spacing-xs: 4px;
  --spacing-sm: 8px;
  --spacing-md: 16px;
}
```

JavaScript Module

```
export const colorPrimary500 = '#3b82f6';
```

Principles of Good APIs

The Golden Rules

1. Intuitive defaults - works out of the box
2. Progressive disclosure - simple to complex
3. Consistent naming - learn once, apply everywhere
4. Type safety - IDE autocomplete is documentation
5. Composition over configuration

Button: A Case Study

The Naive Approach (Avoid)

```
<Button
  type="primary"
  size="large"
  icon="check"
  iconPosition="left"
  loading={true}
  disabled={false}
  fullWidth={false}
  rounded={true}
  onClick={handleClick}
/>
```

Too many boolean props!

Better Button API

Composition-First

```
// Clean, composable API
<Button variant="primary" size="lg">
  <CheckIcon />
  Submit
</Button>

<Button variant="primary" size="lg" loading>
  Submit
</Button>

// Polymorphic - render as link
<Button asChild>
```

Variant Props with CVA

Class Variance Authority

```
import { cva } from 'class-variance-authority';

const button = cva(
  'inline-flex items-center justify-center rounded-md font-medium',
  {
    variants: {
      variant: {
        primary: 'bg-blue-600 text-white hover:bg-blue-700',
        secondary: 'bg-gray-100 text-gray-900 hover:bg-gray-200',
        ghost: 'hover:bg-gray-100'
      },
      size: {
```

The Pattern

What Are Compound Components?

```
// Instead of prop drilling
<Select
  options={options}
  renderOption={...}
  renderGroup={...}
  onSelect={...}
/>

// Compose freely
<Select onValueChange={setValue}>
  <Select.Trigger>
    <Select.Value placeholder="Choose..." />
```

Implementing Compound Components

Context-Based Pattern

```
const SelectContext = createContext(null);

function Select({ children, value, onValueChange }) {
  return (
    <SelectContext.Provider value={{ value, onValueChange }}>
      <div className="relative">{children}</div>
    </SelectContext.Provider>
  );
}

function SelectItem({ value, children }) {
  const { onValueChange } = useContext(SelectContext);
```

Benefits

Why Compound Components?

Traditional Props	Compound Components
Fixed structure	Flexible composition
Prop explosion	Clean API
Hard to customize	Easy to extend
All-or-nothing	Use what you need

The ARIA Foundation

Built-in Accessibility

```
function Dialog({ open, onClose, children }) {
  return (
    <div
      role="dialog"
      aria-modal="true"
      aria-labelledby="dialog-title"
      aria-describedby="dialog-desc"
    >
      <button
        onClick={onClose}
        aria-label="Close dialog"
      >
```

Focus Management

Focus Trapping

```
function Dialog({ open, onClose }) {
  const firstFocusableRef = useRef(null);
  const lastFocusableRef = useRef(null);

  useEffect(() => {
    if (open) {
      // Focus first element when opened
      firstFocusableRef.current?.focus();

      // Return focus when closed
      return () => {
        document.activeElement?.blur();
```

Keyboard Navigation

Common Patterns

Component	Keys
Button	Enter, Space
Menu	Arrow keys, Home, End
Tabs	Arrow left/right
Dialog	Escape to close
Combobox	Arrows, Enter, Escape

Using React Aria

Headless Accessibility

```
import { useButton } from 'react-aria';

function Button(props) {
  const ref = useRef(null);
  const { buttonProps } = useButton(props, ref);

  return (
    <button {...buttonProps} ref={ref}>
      {props.children}
    </button>
  );
}
```

Storybook Setup

Stories as Documentation

```
// Button.stories.tsx
import type { Meta, StoryObj } from '@storybook/react';
import { Button } from './Button';

const meta: Meta<typeof Button> = {
  title: 'Components/Button',
  component: Button,
  tags: ['autodocs'],
  argTypes: {
    variant: {
      control: 'select',
      options: ['primary', 'secondary', 'ghost']
```

Interactive Documentation

Controls and Actions

```
export const Interactive: Story = {
  args: {
    variant: 'primary',
    size: 'md',
    disabled: false,
    loading: false
  },
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    const button = canvas.getByRole('button');

    await userEvent.click(button);
```

MDX Documentation

Combining Prose and Code

```
# Button

Buttons trigger actions when clicked.

## Usage

import { Button } from '@myorg/design-system';

<Button variant="primary">Submit</Button>

## Variants

<C
```

Build vs Buy Decision

The Trade-offs

Build Your Own	Use Existing Library
Full control	Quick to start
Matches brand exactly	Battle-tested
Maintenance burden	Upgrade complexity
Team expertise needed	Dependency on maintainers

The Middle Path: Headless

- Use headless primitives (Radix, React Aria)
- Add your own styling
- Best of both worlds

Migration Path

From MUI to Custom

```
Phase 1: Foundation (2-4 weeks)
  - Set up tokens
  - Create primitives (Button, Input)
  - Parallel usage allowed

Phase 2: Core Components (4-8 weeks)
  - Dialog, Select, Menu
  - Replace MUI components gradually
  - Feature flags for rollout

Phase 3: Cleanup (2-4 weeks)
  - Remove MUI dependency
  - Update documentation
```

Key Takeaways

Building for Scale

1. Start with tokens - single source of truth
2. Design APIs for composition
3. Compound components for flexibility
4. Accessibility is not optional
5. Document everything in Storybook

Resources

Learn More

- Design Systems Handbook
- Storybook
- Radix UI
- React Aria
- Class Variance Authority

Let's build better component libraries!
React Paris Meetup #013