TNG TECHNOLOGY CONSULTING

Bridging the Gap Between UX Design and Development

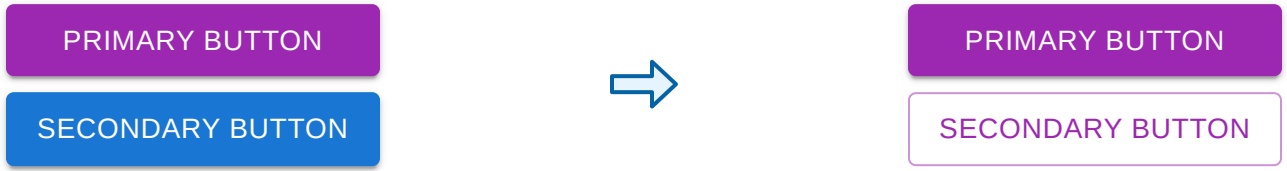
Atomic Design & Storybook.js

Franz Thoma

13.3.2026

BobKonf 2026, Berlin

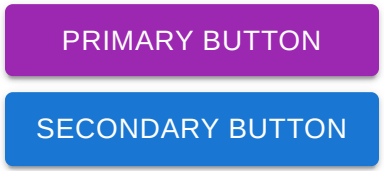
A Small Change in Design



This change took multiple weeks to complete.

A Small Change in Design

Can You Spot the Problem?




```
<Button variant="contained" color="primary">
  Primary Button
</Button>

<Button variant="contained" color="secondary">
  Secondary Button
</Button>
```


```
<Button variant="contained" color="primary">
  Primary Button
</Button>

<Button variant="outlined" color="primary">
  Secondary Button
</Button>
```

A Small Change in Design

What we did

```
<Button variant="contained" color="primary">
  Primary Button
</Button>

<Button variant="outlined" color="primary">
  Secondary Button
</Button>
```

What we should have done

```
<AcmeButton variant="primary">
  Primary button
</AcmeButton>

<AcmeButton variant="secondary">
  Secondary button
</AcmeButton>
```

How Did that Happen?

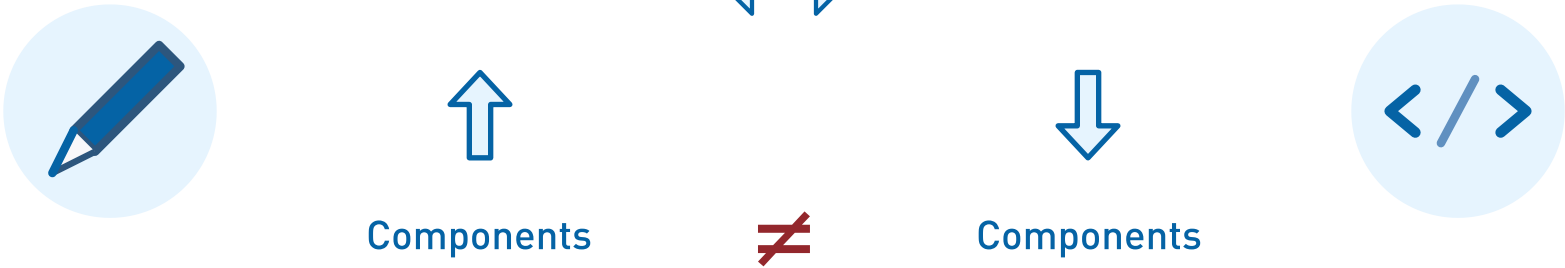
The Handover Process

UX-Designers work in Figma

Handover: Full screens in Figma, review of finished implementation

Developers reproduce designs in code, and use screens to verify

Problem Statement



Handover based only on screens leads to information loss.

⇒ How to find a common language?

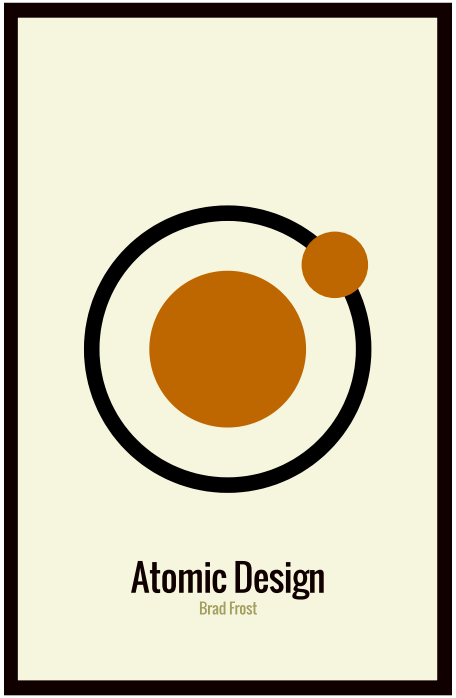
TNG TECHNOLOGY CONSULTING

1

Part 1: Atomic Design

Finding a common language

Atomic Design (atomicdesign.bradfrost.com)



Atomic Design by Brad Frost

We're tasked with making interfaces for more users in more contexts using more browsers on more devices with more screen sizes and more capabilities than ever before. That's a daunting task indeed. Thankfully, design systems are here to help.

Atomic Design details all that goes into creating and maintaining robust design systems, allowing you to roll out higher quality, more consistent UIs faster than ever before. This book introduces a methodology for thinking of our UIs as thoughtful hierarchies, discusses the qualities of effective pattern libraries, and showcases techniques to transform your team's design and development workflow.

order the e-book read now

Atomic Design: Building Blocks

Design Tokens

Colors, typography, spacing

Atoms

Smallest indivisible UI elements

Molecules

Groups of atoms working together

Organisms

Complex compound components

Templates & Pages

Complete layouts

Design Tokens


```
const theme = createTheme({
  palette: {
    primary: { main: '#1976d2' },
    secondary: { main: '#dc004e' },
  },
  typography: {
    fontFamily: 'Inter, sans-serif',
    h1: { fontSize: '2.5rem' },
  },
  spacing: 8, // 8px base unit
});
```

Design Tokens Colors, typography, spacing

Atoms Smallest indivisible UI elements

Molecules Groups of atoms working together

Organisms Complex compound components

Templates & Pages Complete layouts

Atoms

Name

PRIMARY BUTTON

- **Design Tokens** Colors, typography, spacing
- **Atoms** Smallest indivisible UI elements
- **Molecules** Groups of atoms working together
- **Organisms** Complex compound components
- **Templates & Pages** Complete layouts

Molecules

Email

john@example.com

Please enter your email address

SAVE

Design Tokens Colors, typography, spacing

Atoms Smallest indivisible UI elements

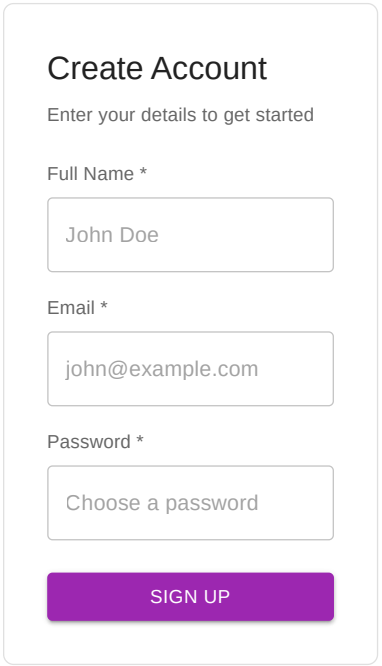
Molecules Groups of atoms working together

Organisms Complex compound components

Templates & Pages Complete layouts

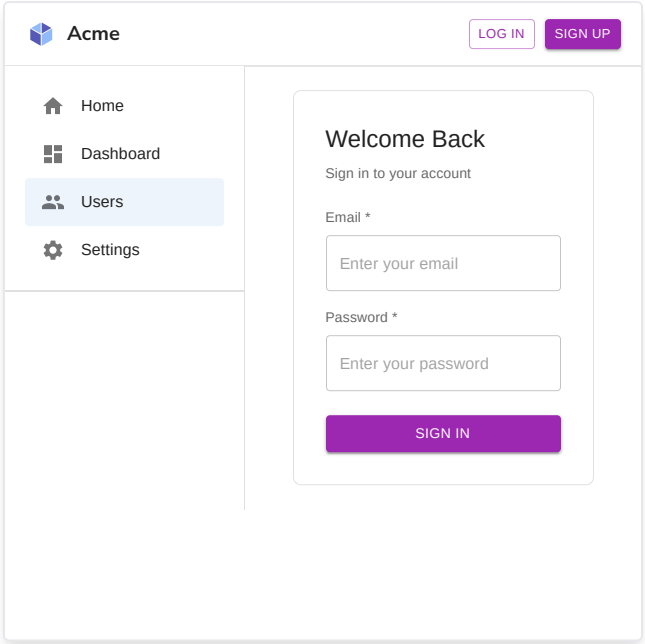
Organisms

TNG TECHNOLOGY CONSULTING



- **Design Tokens** Colors, typography, spacing
- **Atoms** Smallest indivisible UI elements
- **Molecules** Groups of atoms working together
- **Organisms** Complex compound components
- **Templates & Pages** Complete layouts

Templates & Pages



- **Design Tokens** Colors, typography, spacing
- **Atoms** Smallest indivisible UI elements
- **Molecules** Groups of atoms working together
- **Organisms** Complex compound components
- **Templates & Pages** Complete layouts

How This Solves the Button Problem

Before: Full-screen handover

"What was that button variant again…?"

- Designers saw: Button variants as building blocks
- Developers saw: Buttons with different styling

```
<Button variant="contained" color="primary">
  Primary Button
</Button>

<Button variant="outlined" color="primary">
  Secondary Button
</Button>
```

After: Common design language

Button as an **Atom** — clearly defined!

- Designers and developers have the same hierarchy
- Changes are applied on the same building block level

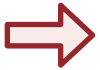
```
<AcmeButton variant="primary">
  Primary button
</AcmeButton>

<AcmeButton variant="secondary">
  Secondary button
</AcmeButton>
```

Handover Process revisited

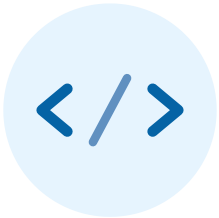
Screens

Screens

Components



Components



TNG TECHNOLOGY CONSULTING

2

Part 2: Storybook.js

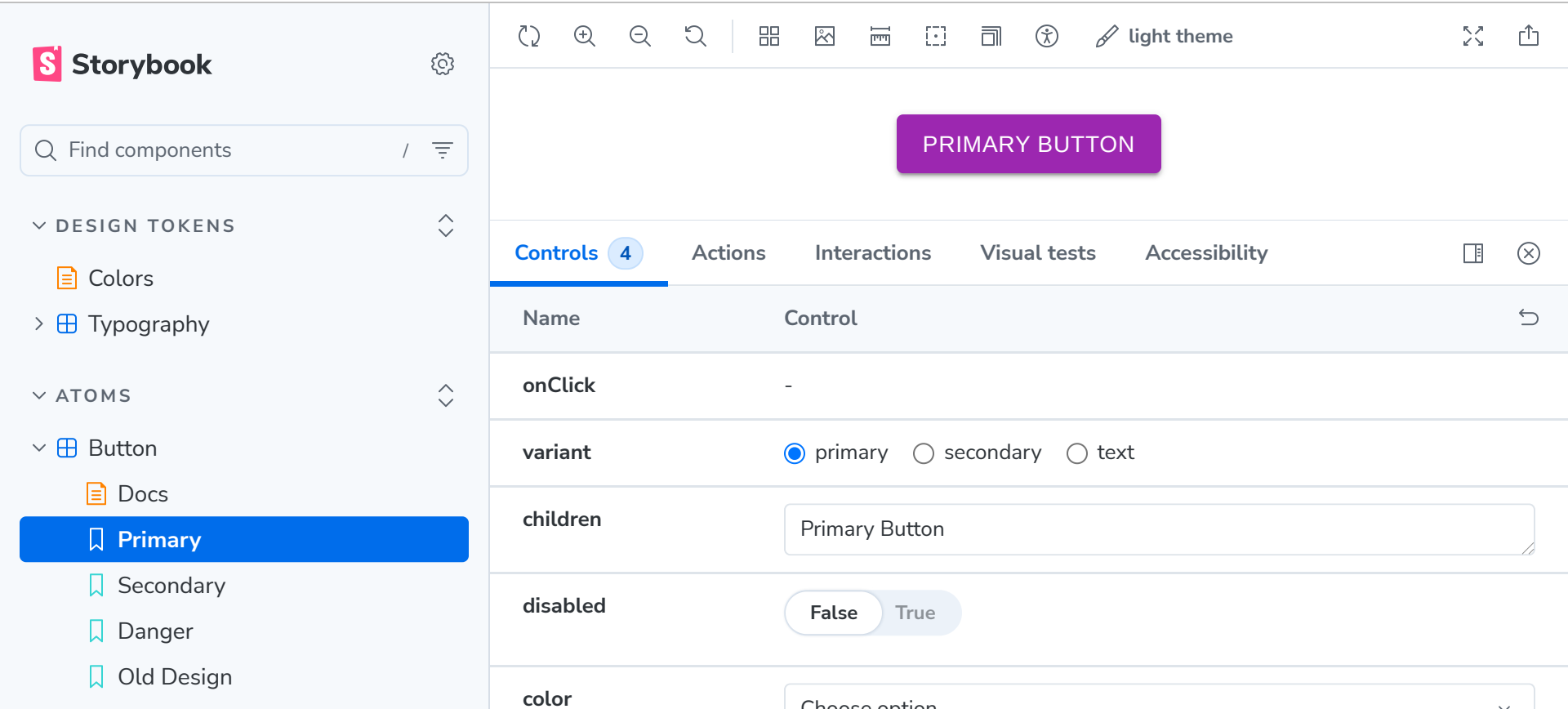
Developer-friendly Component Library

Introducing Storybook.js

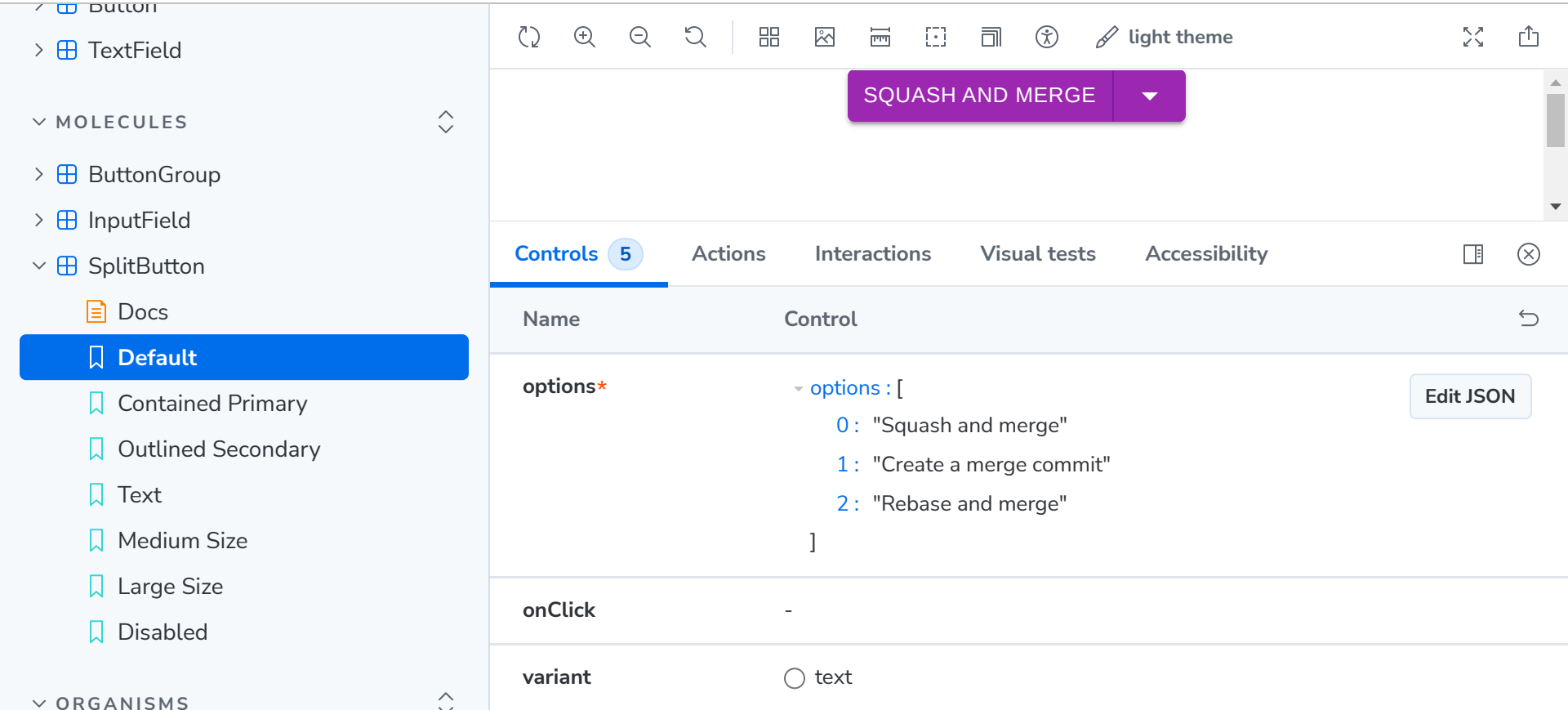
An isolated development environment for UI components

- Visualize every component state
- Interactive playground with controls
- Built for any framework
- Perfect for design systems & collaboration

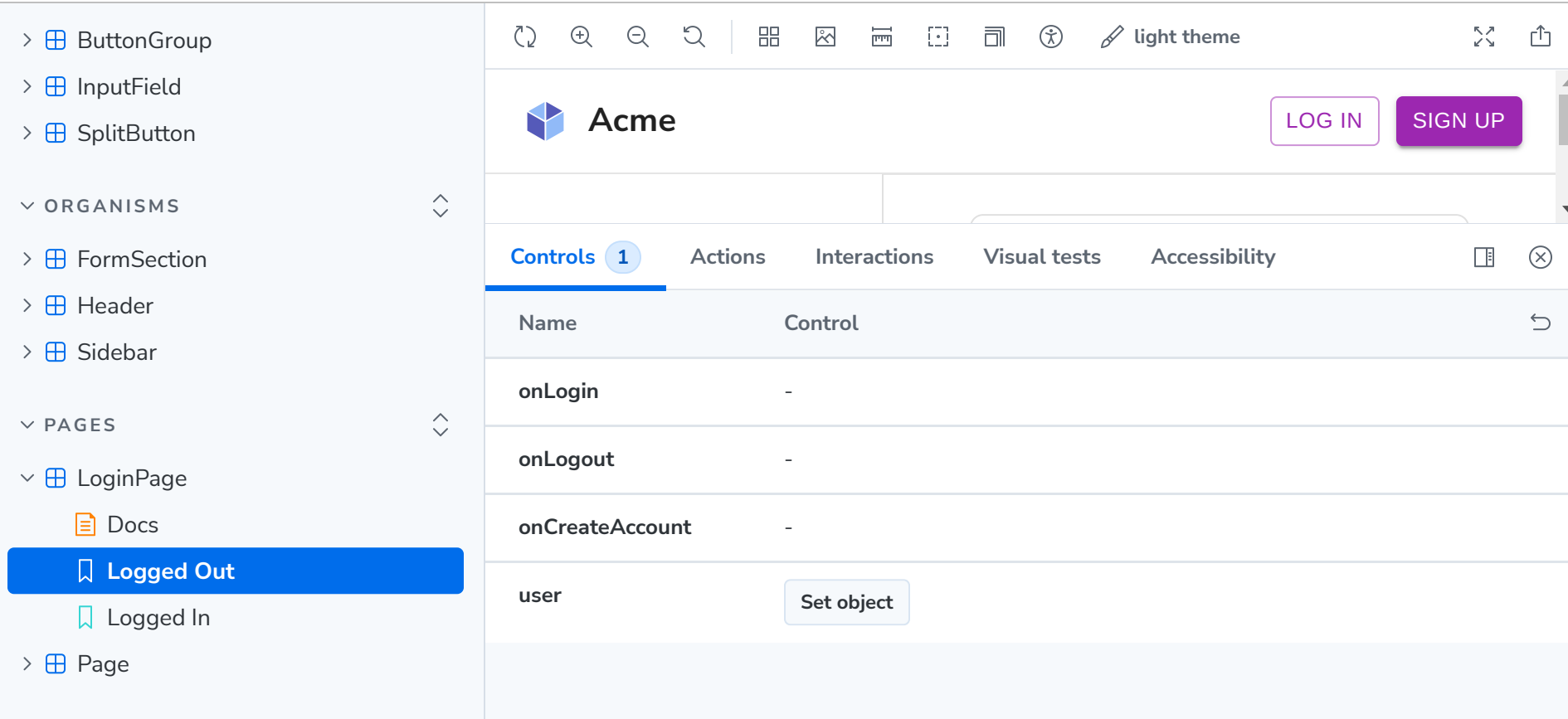
Storybook.js



Storybook.js



Storybook.js



Stories: Document Every State

Capture all component variants in one place

```
export const Primary: Story = {
  render: (props) => <Button {...props} />,
  args: {
    variant: "primary",
    children: "Primary Button",
  },
};
```

```
export const Secondary: Story = {
  render: (props) => <Button {...props} />,
  args: {
    variant: "secondary",
    children: "Secondary Button"
  },
};
```

Controls: Tweak & Explore

Explore and customize props interactively

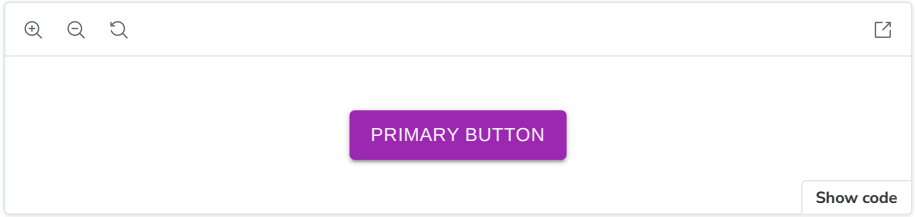
```
export const Primary: Story = {
  render: (props) => <AcmeButton {...props} />,
  args: {
    variant: "primary",
    children: "Primary Button",
    color: undefined,
    disabled: false,
  },
};
```

Button

A flexible button component supporting three variants: primary, secondary, and text.

- primary: Call-to-action button with contained styling (filled). Use at most one primary button per view for the most important action.
- secondary: Outlined button for secondary actions. Can be used multiple times alongside a primary button.
- text: Text-only button with no border or background for low-emphasis actions.

The `color` prop can be used to override the color (e.g. `color={"error"}` for a button warning of a destructive action).



Name	Description	Default	Control
onClick	function	-	-

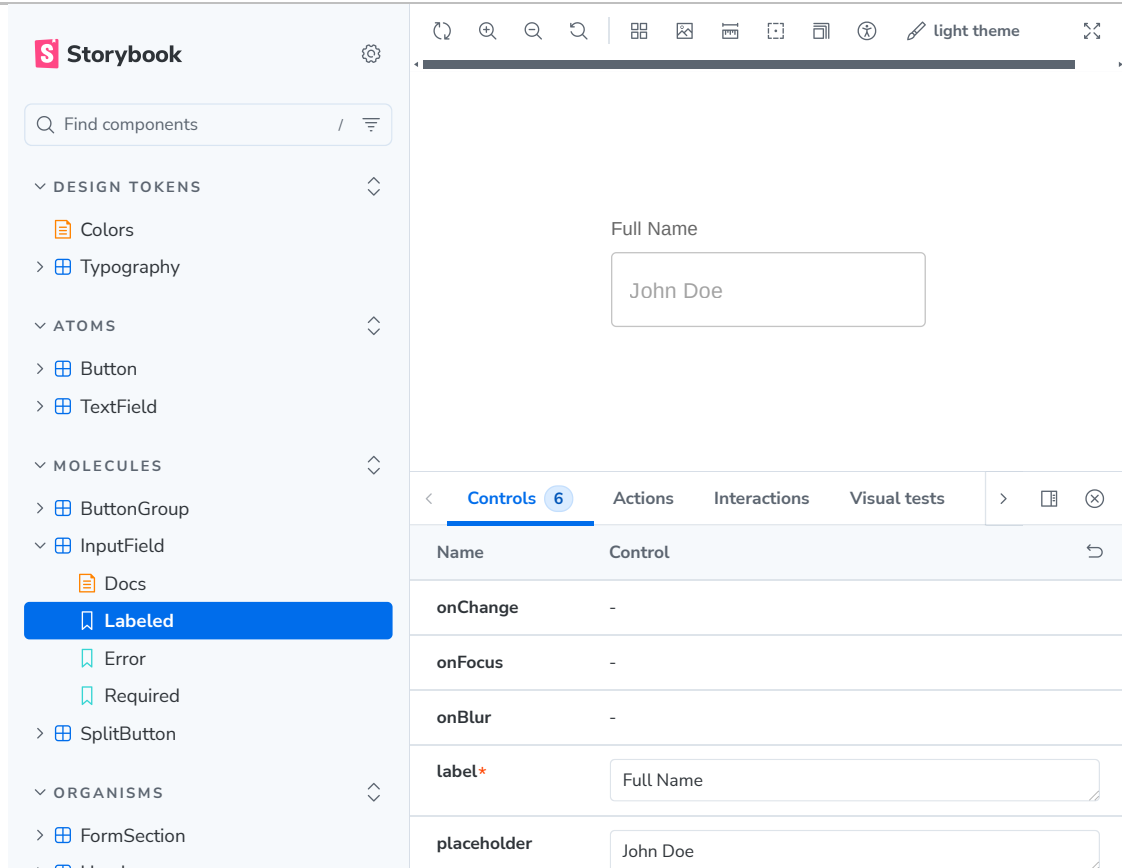
Controls: Tweak & Explore

Define your own controls

```
const meta = {
  title: 'Atoms/Button',
  component: AcmeButton,
  argTypes: {
    variant: {
      control: 'inline-radio',
      options: ['primary', 'secondary', 'text'],
    },
    color: {
      control: 'select',
      options: ['primary', 'secondary', 'error', 'info', 'success', 'warning'],
    },
    disabled: {
      control: 'boolean',
    },
  },
} satisfies Meta<typeof AcmeButton>;
```

Actions: Debug Event Handling


```
const meta = {
  title: 'Molecules/InputField',
  component: InputField,
  args: {
    onChange: fn(),
    onFocus: fn(),
    onBlur: fn(),
  },
} satisfies Meta<typeof InputField>;
```



Markdown Documentation

Rich markdown support in Storybook stories

```
import { Meta, ColorPalette, ColorItem } from '@storybook/addon-do
import { createTheme } from '@mui/material/styles';

<Meta title="Design Tokens/Colors" />

export const { palette } = createTheme();

# Color Palette

Our design system uses Material-UI's color palette to ensure visua

## Color Reference

<ColorPalette>
  <ColorItem
```

Color Palette

Our design system uses Material-UI's color palette to ensure visual consistency across all components.

Color Reference

Name	Swatches
Primary Main actions and key interface elements	dark #1565c0 · main #1976d2 · light #42a5f5
Secondary Secondary actions and complementary elements	dark #7b1fa2 · main #9c27b0 · light #ba68c8
Error Destructive actions and error states	dark #c62828 · main #d32f2f · light #ef5350
Warning	

Autodocs: Documentation That Writes Itself


```
const meta = {
  title: 'Atoms/Button',
  component: AcmeButton,
  tags: ['autodocs']
} satisfies Meta<typeof AcmeButton>;
```

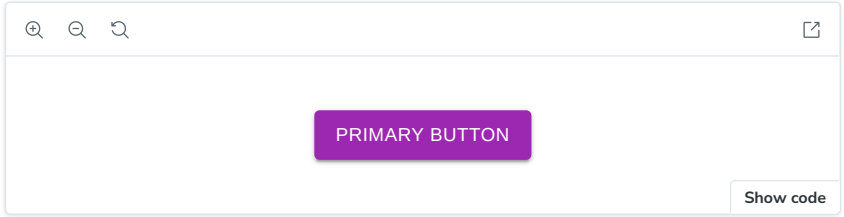
```
/**
 * The primary button is the main call-to-action for
 * a view. It uses the contained style with a solid
 * background color to draw attention. Use sparingly
 * - at most one primary button per section or view
 * to maintain emphasis.
 */
export const Primary: Story = {
  render: (props) => <AcmeButton {...props} />,
};
```

Button

A flexible button component supporting three variants: primary, secondary, and text.

- primary: Call-to-action button with contained styling (filled). Use at most one primary button per view for the most important action.
- secondary: Outlined button for secondary actions. Can be used multiple times alongside a primary button.
- text: Text-only button with no border or background for low-emphasis actions.

The `color` prop can be used to override the color (e.g. `color={"error"}` for a button warning of a destructive action).

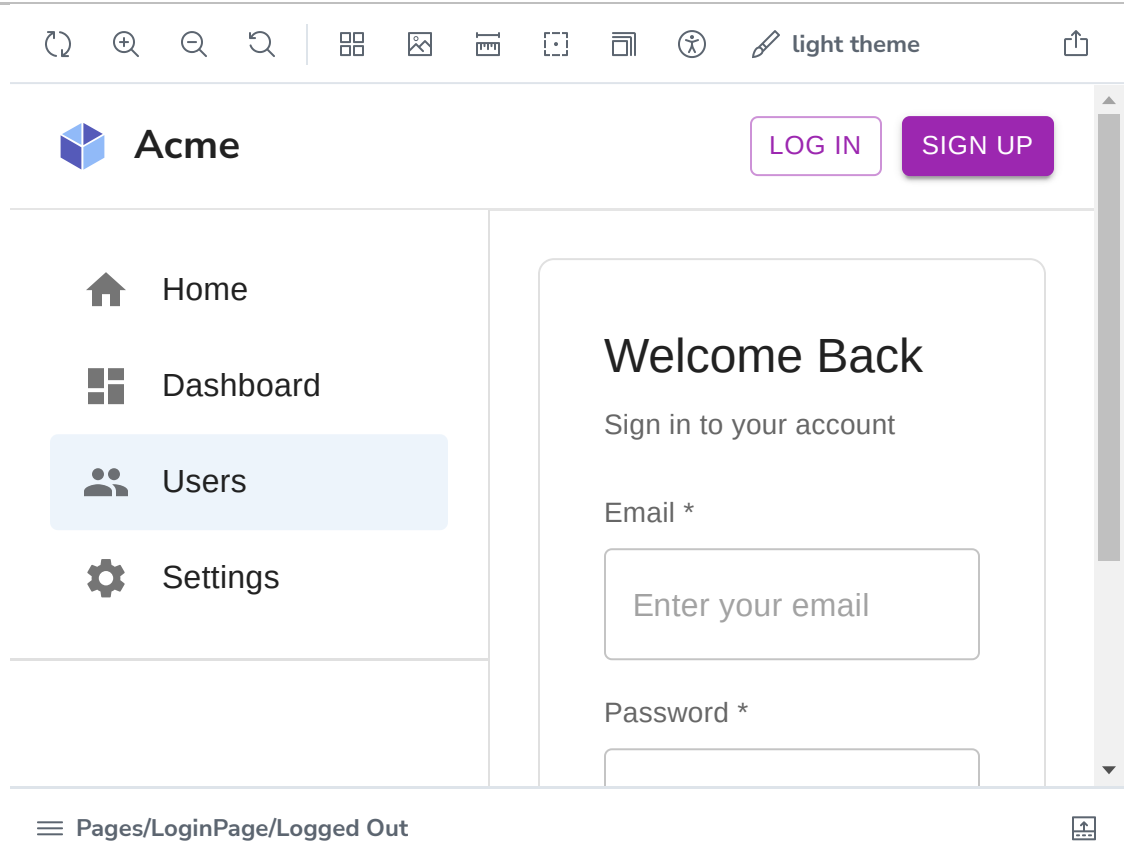


Name	Description	Default	Control
onClick	function	-	-

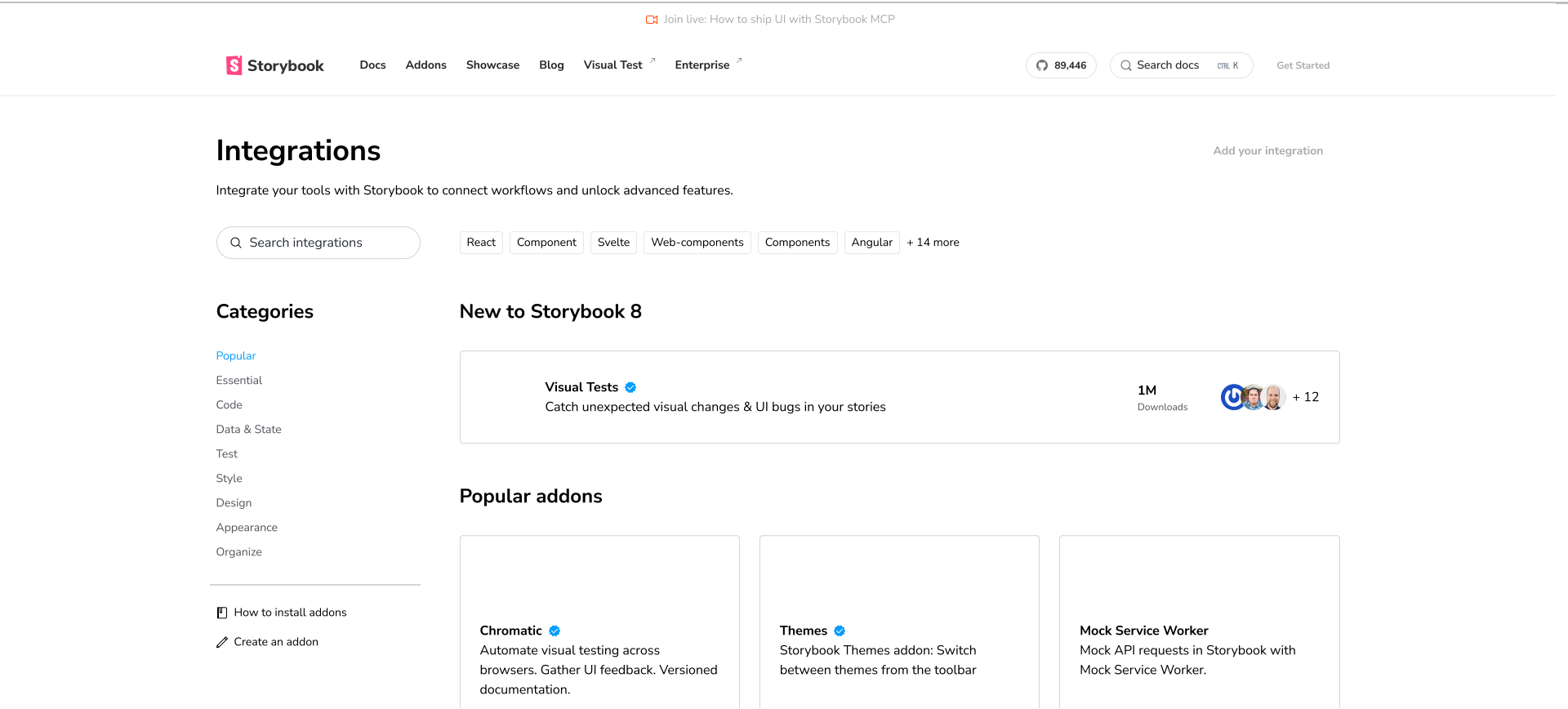
Plugins

- Theming (light/dark mode)
- Accessibility
- Visual testing
- Interaction testing

Discover at
https://storybook.js.org/addons



Plugins (storybook.js.org/addons)



How to Integrate

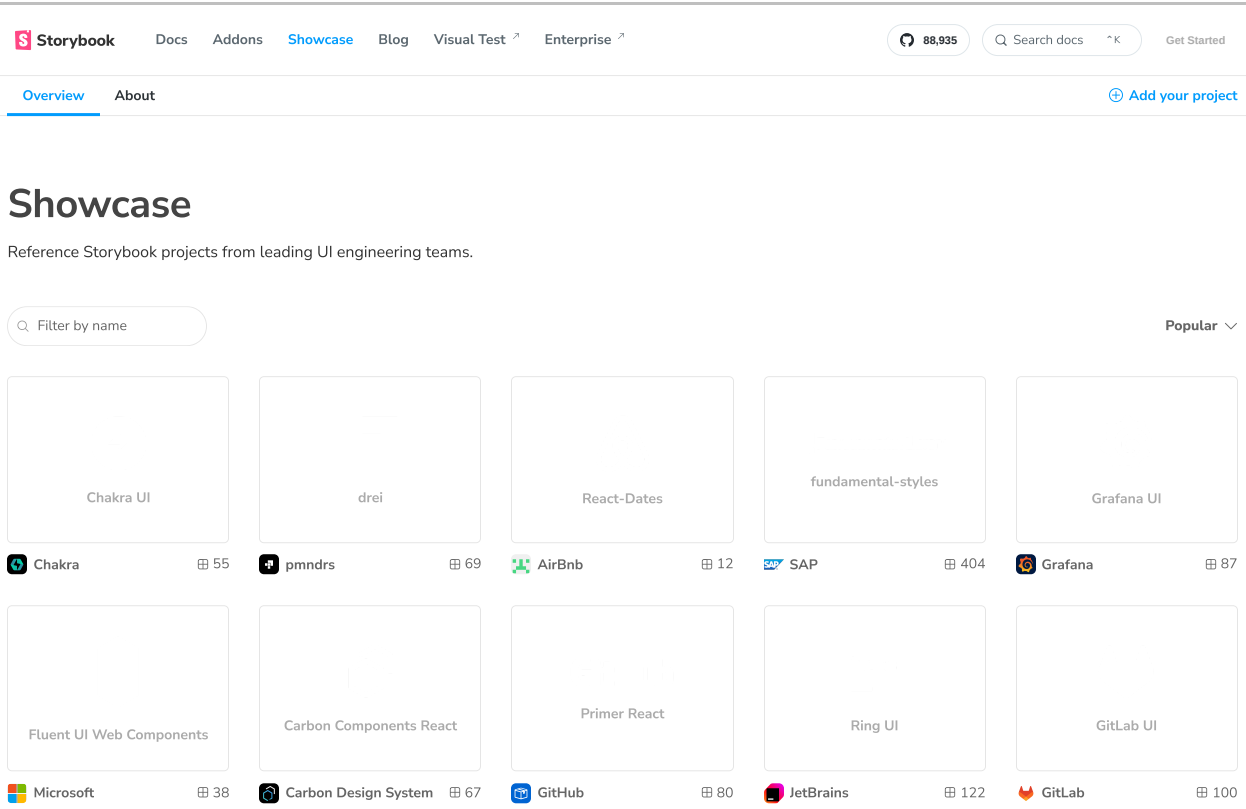
Add to an existing frontend

Documentation and visual testing for components of a frontend application

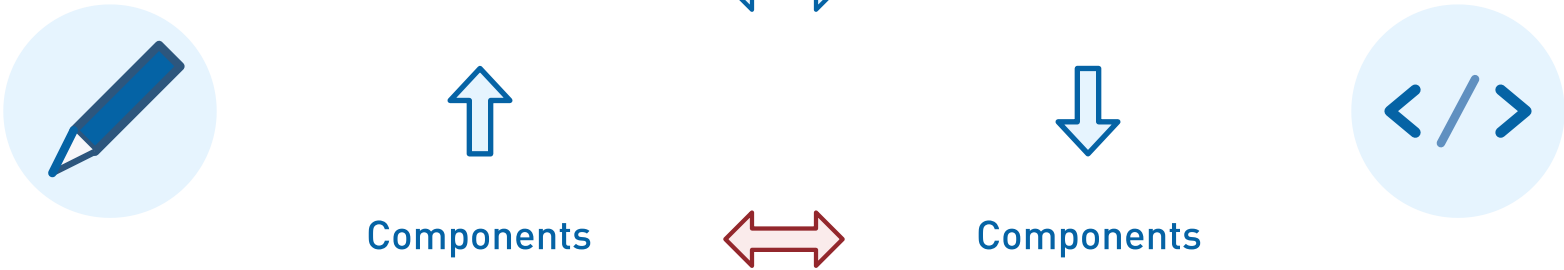
Standalone component library

Create a shared component library or design system for your organization

Get inspired (storybook.js.org/showcase)



Collaboration Reimagined

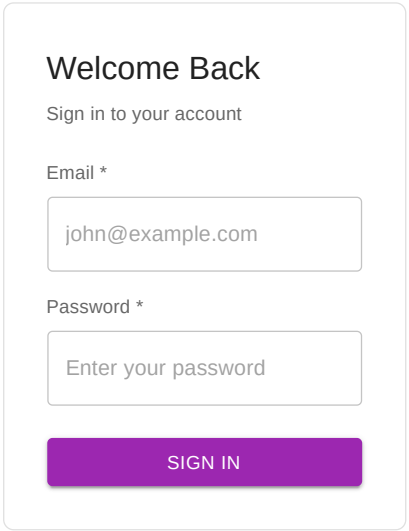


⇒ Components developed & reviewed **collaboratively**

Bonus: Cleaner Architecture

Storybook encourages better code organization

- **Component isolation** — Must work independently
- **Prop-driven UI** — Logical presentation layer separation
- **Reusability first** — Components designed as building blocks
- **Explicit contracts** — Props interface = component API



Bonus: Cleaner Architecture

Dependency injection

❌ Coupled to Backend

```
function LoginForm() {
  const [credentials, setCredentials] = useState({});
  const navigate = useNavigate();
  const handleSubmit = async (e) => {
    const res = backendClient.login(credentials);
    localStorage.setItem('token', res.token);
    navigate("/dashboard")
  };

  return (
    <form onSubmit={handleSubmit}>
      <input onChange={…} />
      <input type="password" onChange={…} />
      <button>Login</button>
    </form>
  );
}
```

✅ Prop-Driven Actions

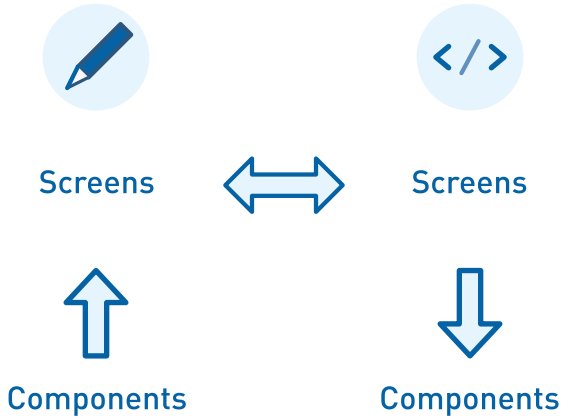
```
function LoginForm({ onLogin }) {
  const [credentials, setCredentials] = useState({});

  return (
    <form onSubmit={() => onLogin(credentials)}>
      <input onChange={…} />
      <input type="password" onChange={…} />
      <button>Login</button>
    </form>
  );
}
```

3 Conclusion

Bringing it all together

Old vs New Handover Flow

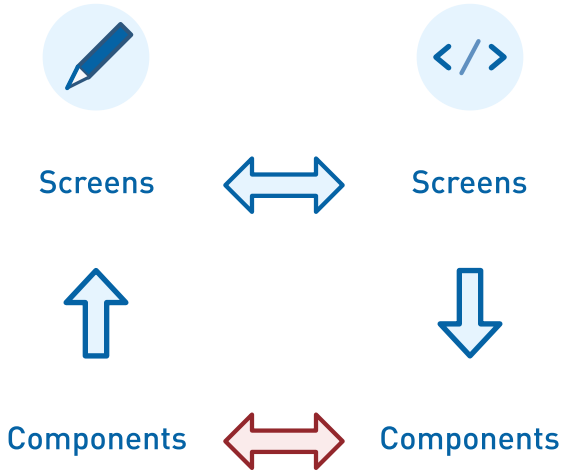
Before: Full-Screen handover



- **Designers** create full-screen designs in Figma
- **Developers** implement full-screen designs
- **Designers** review the implementation
- Separate tooling, each side is a black box

Old vs New Handover Flow

After: Collaboration



- **Designers** create Design Tokens, Atoms, ..., Pages
- **Developers** implement these components in Storybook
- Both sides share single source of truth
- Both sides know how the other is working

Caveats

Beware of Big Up-Front Design

❌ **Don't:**

- Create design systems without involving developers
- Force a big-bang migration to a new design system
- Create all the components from scratch

✅ **Do instead:**

- Implement each component as it's designed
- Refactor existing components
- Build on existing component libraries like MaterialUI or ChakraUI

Getting Started

Install Storybook

```
# Automatic setup for your framework
npm create storybook@latest
```

Getting Started

Start small

- Begin with **atoms**: Buttons, inputs, cards
- Add components you've adapted from libraries
- Create a few stories to understand the workflow
- Add stories for every new or changed component

Getting Started

Build on existing component libraries

Designers often have a Figma component library:

- Design tokens
- Component variants
- Usage guidelines
- Style guides

Use these it as your blueprint:

- Map **Design Tokens** to Theme configuration
- Create **Atoms** from UI primitives
- Build **Molecules** from Figma compositions
- Collaborate on Storybook stories together

Key Takeaways

- **Full-screen handover** loses component hierarchy information.
- **Atomic Design** provides a shared vocabulary for designers and developers.
- **Storybook** enables bi-directional handover on component level.
- **Component isolation** leads to cleaner architecture and allows independent, lightweight testing of components.

TNG TECHNOLOGY CONSULTING

Thank you for your attention

Any questions?

Franz Thoma
Principal Consultant
franz.thoma@tngtech.com