

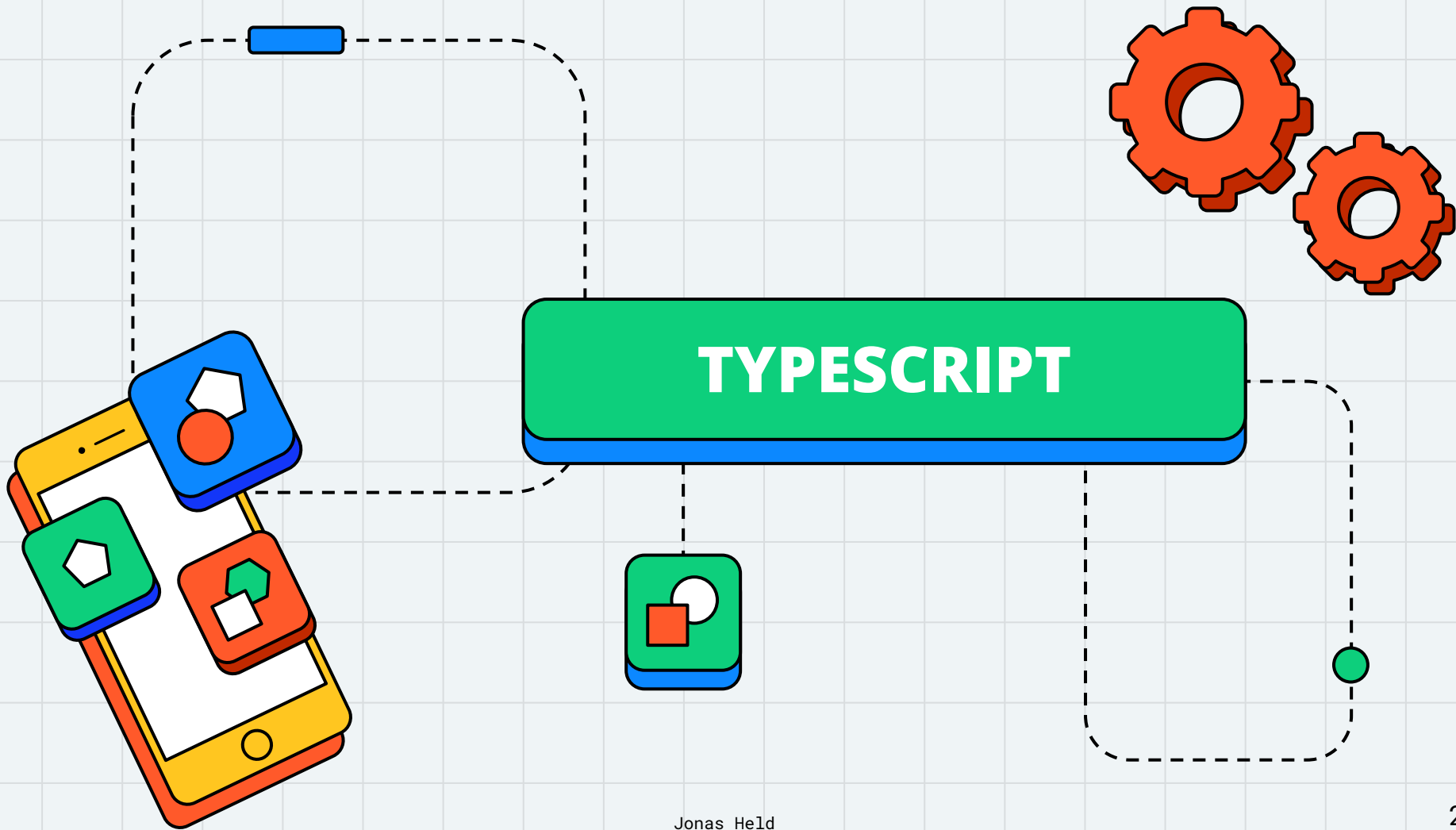


**STYLED COMPONENTS &  
STORYBOOK**

# **KOMPONENTENBASIERTE WEB-ENTWICKLUNG**

**IN REACT**

Tobias Helmrich, th138  
Jonas Held, jh257



# TYPESCRIPT

## DISCLAIMER:

Wir verwenden hauptsächlich TypeScript für diese Präsentation

```
1 type Props = {  
2   prop1: string;  
3   prop2?: number; // optional  
4   prop3: 'option1' | 'option2'; // union  
5 };
```



# VORWORT

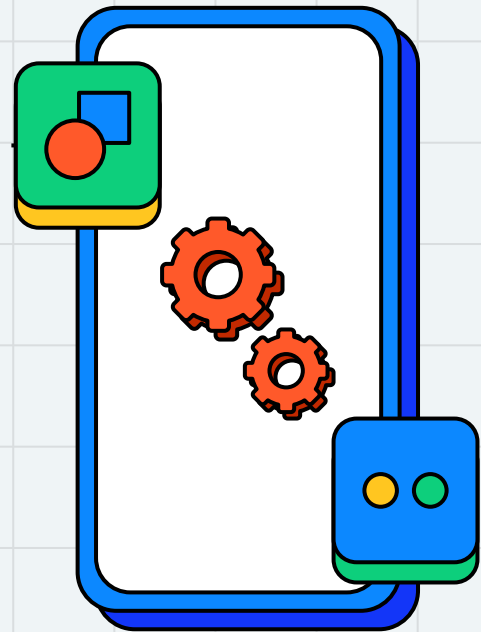
Komponentenbasierte Web-Entwicklung

# MODERNE WEB APPS

## COMPONENT-DRIVEN •

“The development and design practice of building user interfaces with modular components. UIs are built from the “bottom up” starting with basic components then progressively combined to assemble screens.”

<https://www.componentdriven.org>



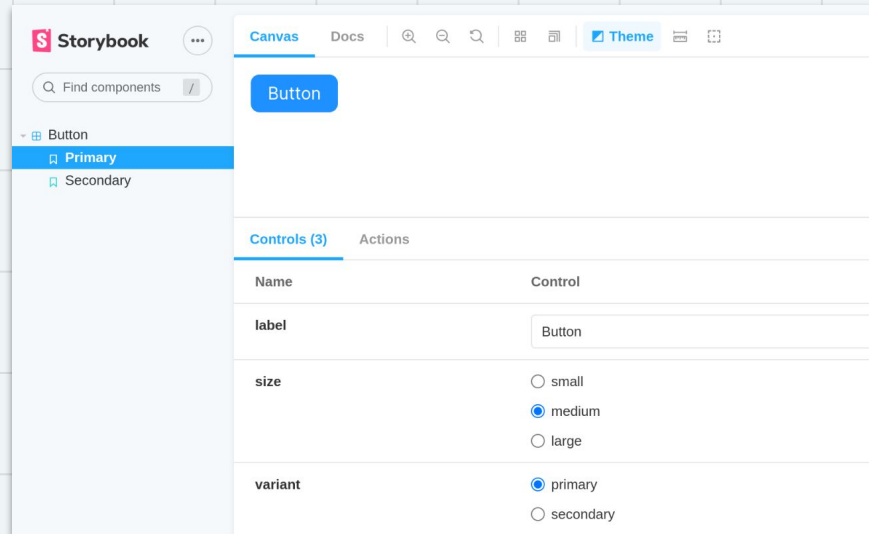
## ABER...

- Styling mit “normalem”, ausgelagertem CSS, CSS Modules, inline
- Oftmals Top-Down-Ansatz bei der Entwicklung von Komponenten

# STYLED COMPONENTS

```
1 import styled from 'styled-components';
2
3 const Button = styled.button`
4   background-color: dodgerblue;
5   color: white;
6   padding: 0.5rem 1rem;
7   border-radius: 8px;
8
9   &:hover {
10     filter: brightness(0.9);
11   }
12 `;
13
14 export default Button;
```

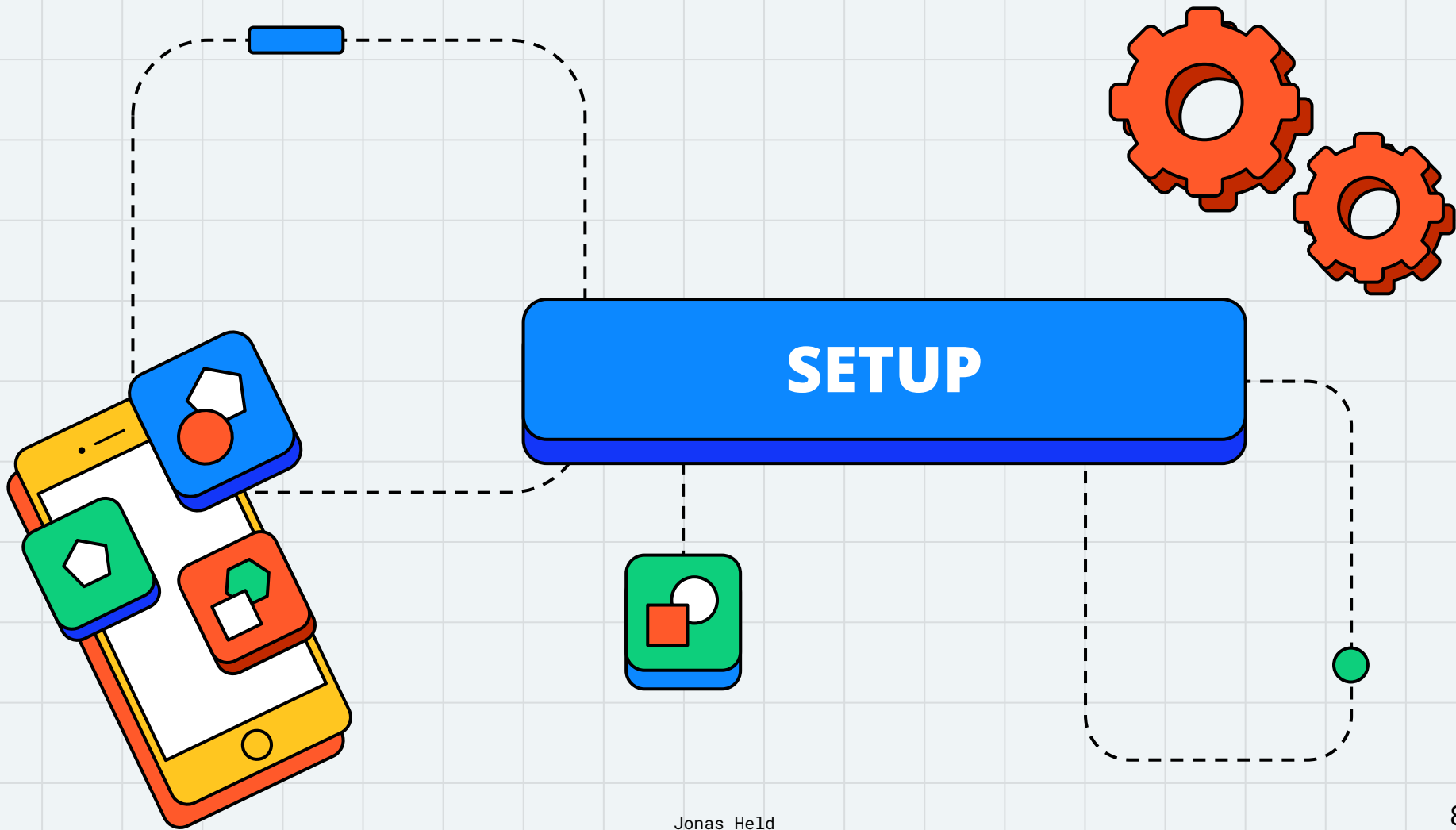
# STORYBOOK





# STYLED COMPONENTS

Komponenten... mit Style. 🏗️



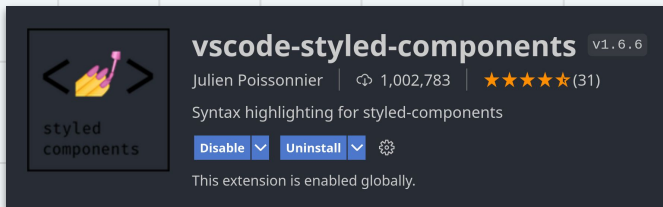


# INSTALLATION

Installieren von Styled Components in einer React App:

```
# Create React App mit TypeScript  
npx create-react-app my-project --template typescript  
cd my-project  
  
# npm  
npm install styled-components  
npm install --save-dev @types/styled-components  
  
# yarn  
yarn add styled-components  
yarn add --dev @types/styled-components
```

# Visual Studio Code Extension



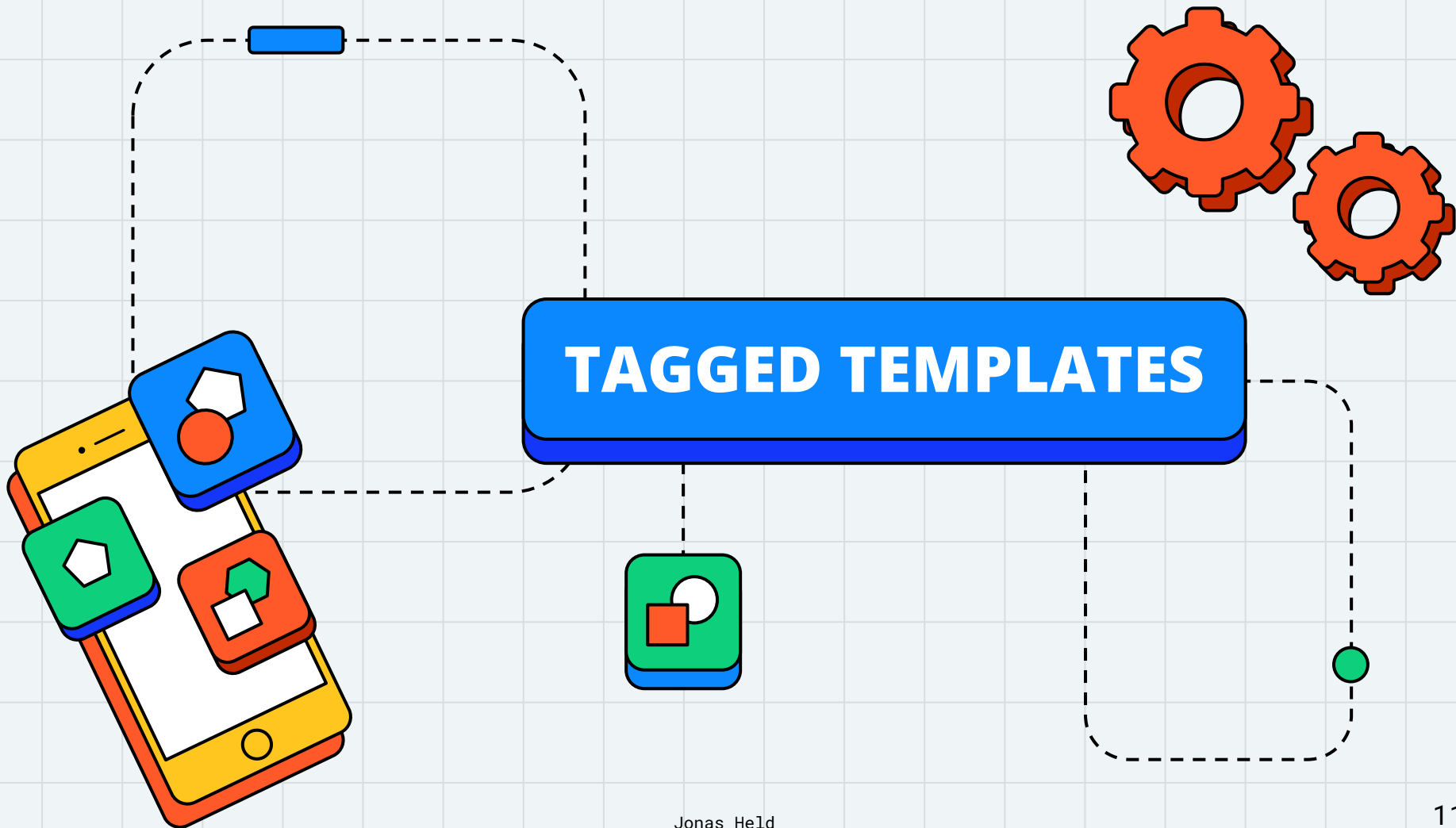
<https://marketplace.visualstudio.com/items?itemName=jpoissonnier.vscode-styled-components>

## Syntax-Highlighting und IntelliSense für Styled Components

```
1 const Button = styled.button`
2   background-color: dodgerblue;
3   color: white;
4   padding: 0.5rem 1rem;
5   border-radius: 8px;
6 `;
```



```
1 const Button = styled.button`
2   background-color: dodgerblue;
3   color: white;
4   padding: 0.5rem 1rem;
5   border-radius: 8px;
6 `;
```



# TEMPLATE LITERALS

- Der Inhalt des Templates wird einer Funktion übergeben
- Bei Backticks (``) wird eine Default-Funktion aufgerufen, die den Text und die Platzhalter in einen String zusammenfasst

```
1 const value = 42;  
2 const string = `I'm the answer to everything: ${value}`;  
3  
4 console.log(string); // I'm the answer to everything: 42
```

# TAG FUNCTION

- In JavaScript kann man eigene Funktionen für Templates verwenden
- Derartige Funktionen nennt man "tag"
- Tagged Templates sind einfach nur eine andere Art und Weise, Funktionen aufzurufen

```
1 const tag = () => {  
2   console.log("I'm just a function!");  
3 };  
4  
5 tag``; // I'm just a function!  
6 tag(); // I'm just a function!
```

# TAGGED TEMPLATE ARGUMENTS

- Erster Parameter ist ein Array bestehend aus allen Strings des Templates, getrennt durch die Platzhalter
- Darauf folgende Parameter sind die Werte der Interpolationen in der übergebenen Reihenfolge

```
1  const tag = ( ...args) => {  
2    console.log( ...args);  
3  };  
4  
5  const var1 = 'magic';  
6  const var2 = 'function';  
7  
8  tag`Is this ${var1} or just a simple ${var2}?`;  
9  // [ 'Is this ', ' or just a simple ', '?' ] magic function
```



The diagram features a central blue rounded rectangle containing the text 'WAS IST STYLED COMPONENTS?'. To the left is a yellow smartphone with three floating icons: a blue square with a white house and red circle, a green square with a white house, and an orange square with a green house and white square. A dashed line connects the top of the phone to a blue rectangle at the top left. To the right of the central box is a green square with a white circle and red square, connected by a dashed line. Further right is a dashed line ending in a green circle. In the top right corner are two interlocking orange gears.

# WAS IST STYLED COMPONENTS?




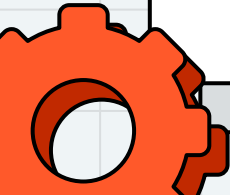


“Utilising tagged template literals [...] and the power of CSS, styled-components allows you to write actual CSS code to style your components. It also removes the mapping between components and styles – using components as a low-level styling construct could not be easier!”

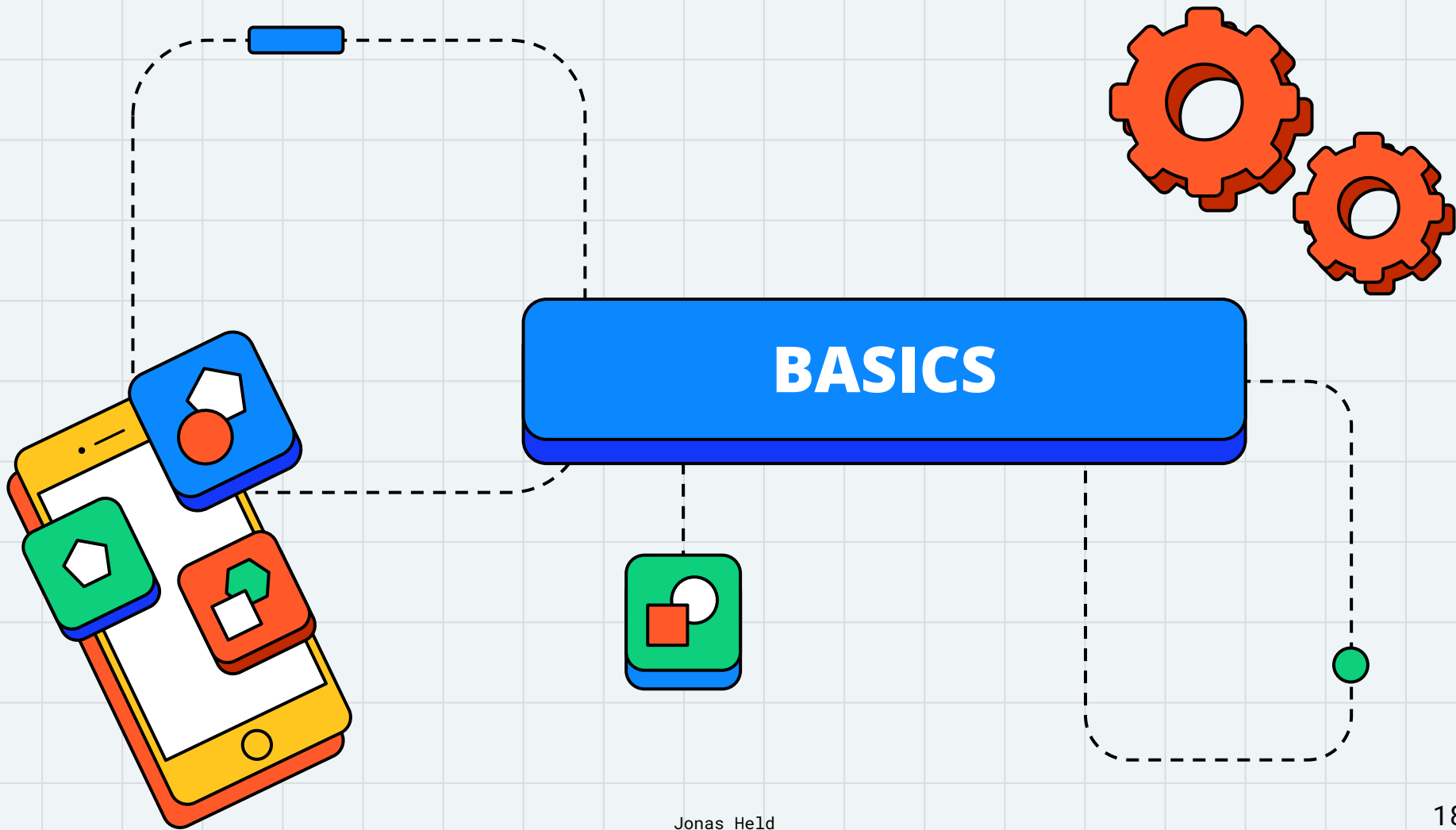
<https://styled-components.com/docs>





# STYLED COMPONENTS

- 
- 
- 
- 
- Über 35.000 GitHub stars (Stand November 2021)
  - **Motivation:** *“styled-components is the result of wondering how we could enhance CSS for styling React component systems”*
  - **Ansatz:**
    - Direkte Kopplung von Styles an Komponenten
    - Erstellen von gestylten Komponenten statt “herkömmlichem” Styling durch CSS-Module, Klassen oder Inline-Styling
  - **“CSS with Superpowers”**
    - CSS-Preprocessor Stylis erlaubt Sass-like Syntax mit Nesting und Vendor-Prefixing
    - CSS-in-JS ermöglicht JavaScript-Logik in CSS



# BASIC STYLING

- “Normales” CSS ohne kompliziertere Syntax
- Sass-like Syntax mit Nesting
- Komponenten-Selektor verwendbar

```
1 import styled from 'styled-components';
2 import Button from './Button';
3
4 const Container = styled.div`
5   height: 100vh;
6   display: flex;
7   justify-content: center;
8   align-items: center;
9   gap: 16px;
10
11   ${Button} {
12     // ...
13   }
14 `;
15
16 export default Container;
```

```
1 import styled from 'styled-components';
2
3 const Button = styled.button`
4   background-color: dodgerblue;
5   color: white;
6   padding: 0.5rem 1rem;
7   border-radius: 8px;
8
9   &:hover {
10     filter: brightness(0.9);
11   }
12 `;
13
14 export default Button;
```

# BASIC STYLING - KOMPONENTE

- Gestylte Komponenten können wie ganz normale React-Komponenten benutzt werden

```
1 import Button from './components/Button';
2 import Container from './components/Container';
3
4 const App = () => {
5   return (
6     <Container>
7       <Button>Button</Button>
8     </Container>
9   );
10 };
11
12 export default App;
```

A blue button with the text "Button" inside a white container.

# EXTENDING STYLES & `AS`

- Mit der `styled()`-Funktion kann man Styles erweitern oder React-Komponenten stylen
- Mit `as` kann man einen Style für ein anderes HTML-Element wiederverwenden, indem man das gerenderte Element austauscht

```
1 const ExtendedButton = styled(Button)`  
2   background-color: seagreen;  
3 `;
```

```
1 <Container>  
2   <Button as="a" href="#">  
3     Blue  
4   </Button>  
5   <ExtendedButton>Green</ExtendedButton>  
6 </Container>
```



# INTERPOLATION & PROPS

- String-Interpolation in Template Literals ermöglicht:
  - Zugriff auf Props mithilfe von Arrow-Functions
  - Verwenden von JavaScript im CSS-Template
- “Falsy” Interpolationen werden ignoriert

```
1 type ButtonProps = {  
2   variant?: 'primary' | 'secondary';  
3 };  
4  
5 const Button = styled.button<ButtonProps>`  
6   background-color: ${({ variant = 'primary' }) =>  
7     variant === 'secondary' ? 'seagreen' : 'dodgerblue'};  
8   color: white;  
9   padding: 0.5rem 1rem;  
10  border-radius: 8px;  
11  
12  &:hover {  
13    filter: brightness(0.9);  
14  }  
15 `;
```

# INTERPOLATION & PROPS - KOMPONENTE

```
1 <Container>  
2   <Button>Primary</Button>  
3   <Button variant="secondary">Secondary</Button>  
4 </Container>
```

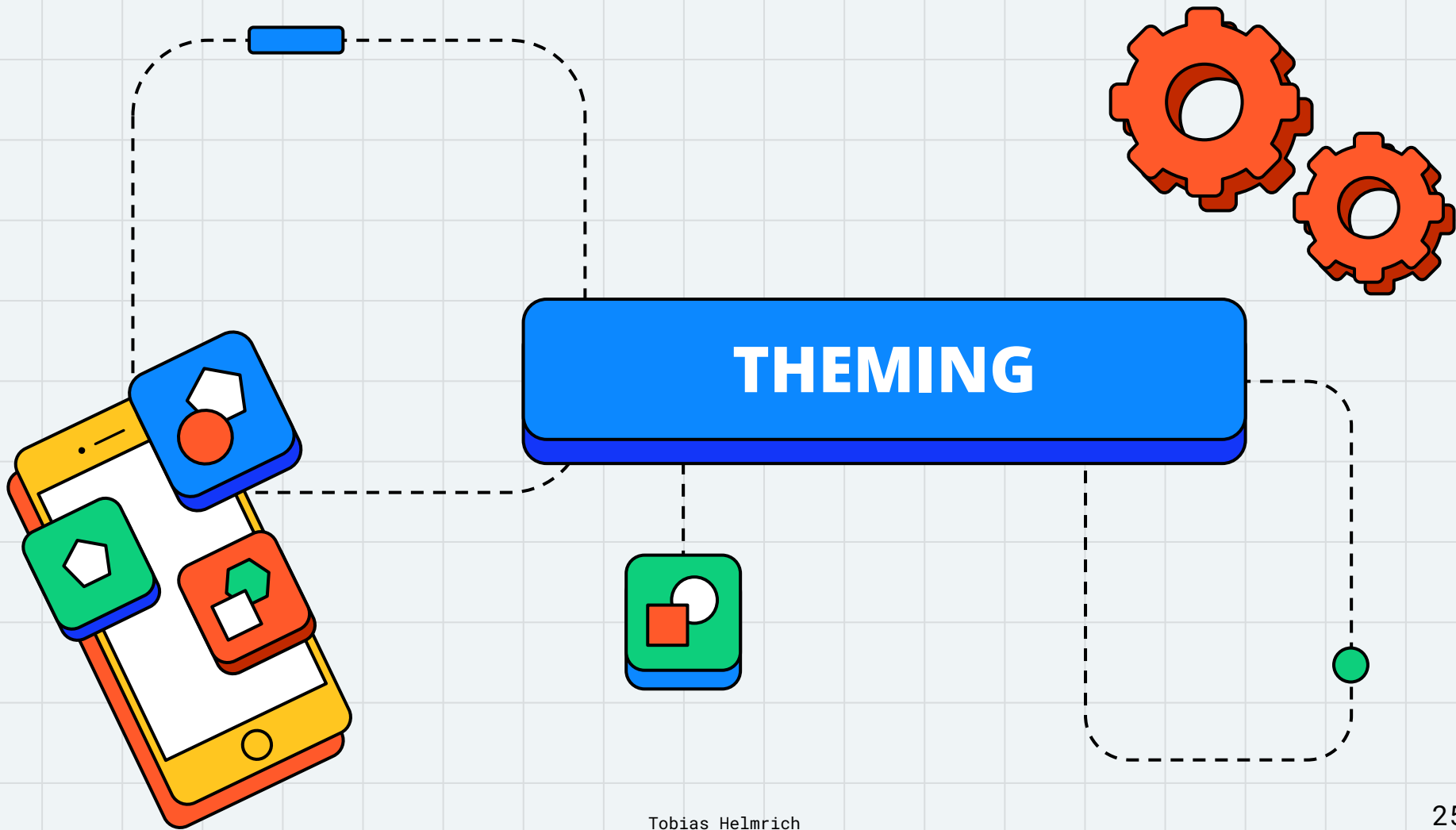


# ATTRIBUTES

`.attrs()` ermöglicht es HTML-Attribute (z.B. `type` bei `inputs`) und Props einer Styled Component zu setzen

```
1  type ButtonProps = {
2    variant?: 'primary' | 'secondary';
3  };
4
5  const Button = styled.button.attrs<ButtonProps>((props) => ({
6    onClick: () => {
7      console.log('test');
8    },
9    variant: props.variant || 'primary',
10  } <ButtonProps>`
11    background-color: ${({ variant }) =>
12      variant === 'secondary' ? 'seagreen' : 'dodgerblue'};
13    color: white;
14    padding: 0.5rem 1rem;
15    border-radius: 8px;
16
17    &:hover {
18      filter: brightness(0.9);
19    }
20  `);
```





# THEMING - THEMEPROVIDER

- Styled Components hat Theming-Support und stellt hierfür eine ``ThemeProvider``-Komponente bereit
- Benutzt hierfür intern React-Context, also kann man auch mit der ``useTheme()``-Hook auf das Theme zugreifen

```
1 import { ThemeProvider } from 'styled-components';
2 import theme from './theme';
3 import Button from './components/Button';
4 import Container from './components/Container';
5
6 const App = () => {
7   return (
8     <ThemeProvider theme={theme}>
9       <Container>
10         <Button>Primary</Button>
11         <Button variant="secondary">Secondary</Button>
12       </Container>
13     </ThemeProvider>
14   );
15 };
16
17 export default App;
```

# THEMING - TYPE

Um das Theme zu typisieren, erstellt man einen Theme-Type und überschreibt in einer Declaration-File den ``DefaultTheme``-Type von Styled Components

```
1  type Theme = {
2    colors: {
3      primary: string;
4      secondary: string;
5      background: string;
6      onPrimary: string;
7      onSecondary: string;
8      onBackground: string;
9    };
10   fontSizes: {
11     small: string;
12     medium: string;
13     large: string;
14   };
15 };
16
17 export default Theme;
```

```
// styled.d.ts
1  import 'styled-components';
2  import Theme from './Theme';
3
4  declare module 'styled-components' {
5    export interface DefaultTheme extends Theme {}
6  }
```

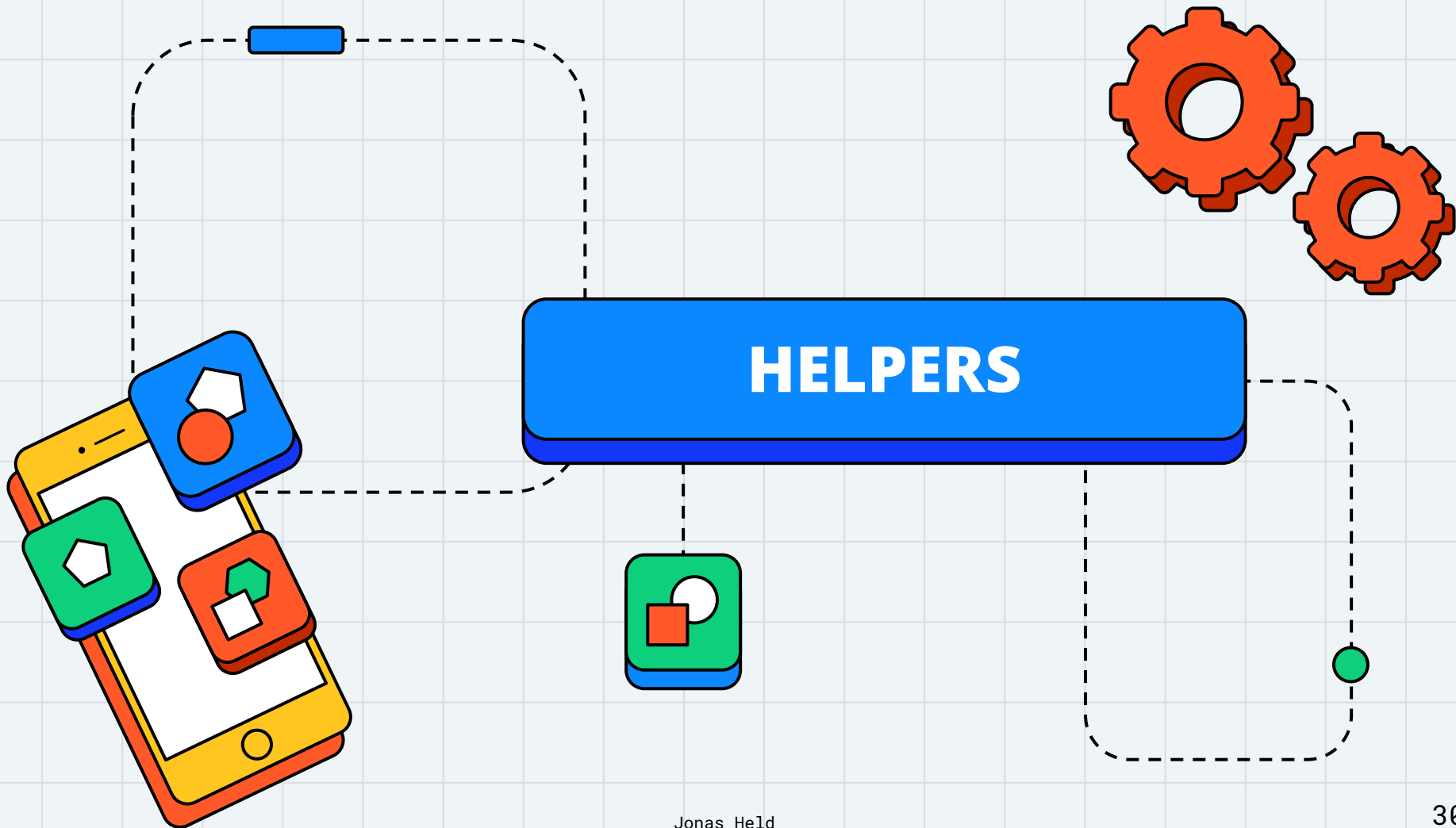
# THEMING - THEME-IMPLEMENTIERUNG

```
1  import Theme from './Theme';
2
3  const theme: Theme = {
4    colors: {
5      primary: 'dodgerblue',
6      secondary: 'seagreen',
7      background: 'white',
8      onPrimary: 'white',
9      onSecondary: 'white',
10     onBackground: 'black',
11   },
12   fontSizes: {
13     small: '12px',
14     medium: '16px',
15     large: '20px',
16   },
17 };
18
19 export default theme;
```

# THEMING - STYLED COMPONENT

``theme`` ist eine Prop, die Teil jeder Styled Component ist

```
1  type ButtonProps = {
2    variant?: 'primary' | 'secondary';
3  };
4
5  const Button = styled.button<ButtonProps>`
6    background-color: ${({ theme, variant = 'primary' }) =>
7      variant === 'secondary' ? theme.colors.secondary : theme.colors.primary};
8    color: ${({ theme, variant }) =>
9      variant === 'secondary' ? theme.colors.onSecondary : theme.colors.onPrimary};
10   padding: 0.5rem 1rem;
11   border-radius: 8px;
12
13   &:hover {
14     filter: brightness(0.9);
15   }
16 `;
```



# HELPERS

- `createGlobalStyle```
  - Erstellt eine Komponente, mit der man globale Styles (z.B. CSS-Resets oder Base-Stylesheets) festlegen kann
- `css```
  - Wird benötigt, wenn man innerhalb einer Interpolation ein Template Literal verwenden will
  - Kann außerdem eingesetzt werden, um wiederverwendbare CSS-Blöcke zu definieren
- `keyframes```
  - Wird verwendet, um keyframes zu definieren

# GLOBALSTYLE



```
1 import { createGlobalStyle } from 'styled-components';
2
3 const GlobalStyle = createGlobalStyle`
4   * {
5     font-size: ${({ theme }) => theme.fontSizes.medium};
6   }
7   body {
8     background-color: ${({ theme }) => theme.colors.background};
9   }
10  button {
11    cursor: pointer;
12    border: 0;
13  }
14 `;
15
16 export default GlobalStyle;
```

```
1 const App = () => {
2   return (
3     <ThemeProvider theme={theme}>
4       <GlobalStyle />
5       <Container>
6         <Button>Primary</Button>
7         <Button variant="secondary">Secondary</Button>
8       </Container>
9     </ThemeProvider>
10  );
11 };
```







## VORTEILE

- CSS-in-JS als logische Folge von HTML-in-JS => eigenständige Komponenten (Markup + Logik + Styling)
  - Keine Trennung zwischen Style und Komponente
  - Dynamic Styling: Logik in Styles möglich
  - Wiederverwendbarkeit
  - "Normales" CSS
  - Theming out of the box
  - Gute Docs
- 
- 


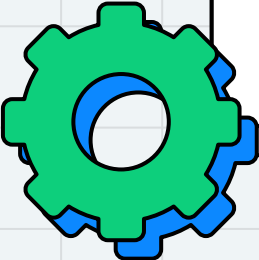
## NACHTEILE

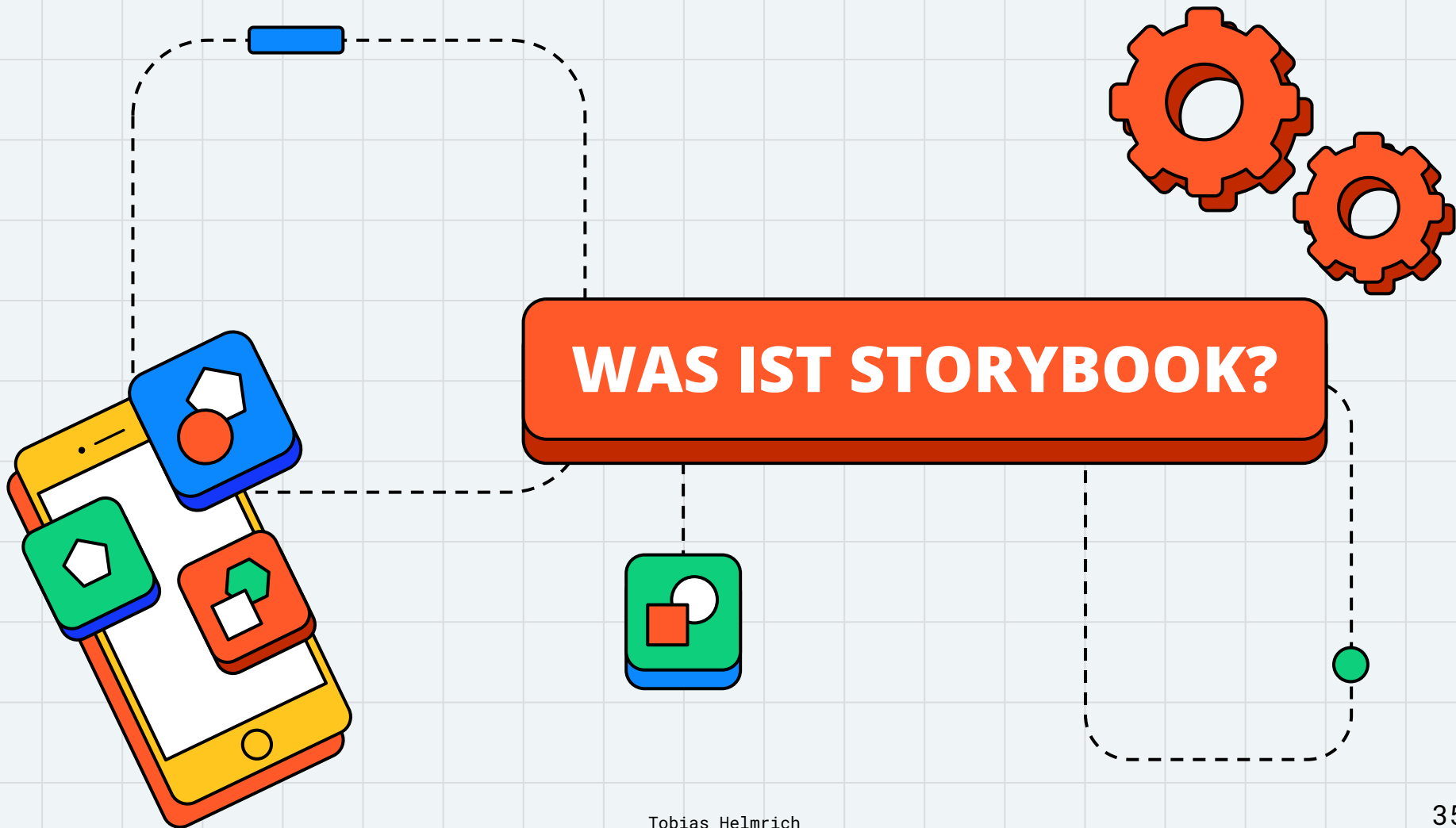
- Oftmals repetitive Interpolation notwendig
  - Gefahr, Komponente "aufzublähen"
- 
- 



# STORYBOOK

Komponenten isoliert entwickeln







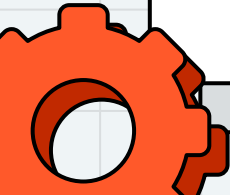

"Storybook is an open source tool for building UI components and pages in isolation. It streamlines UI development, testing, and documentation."

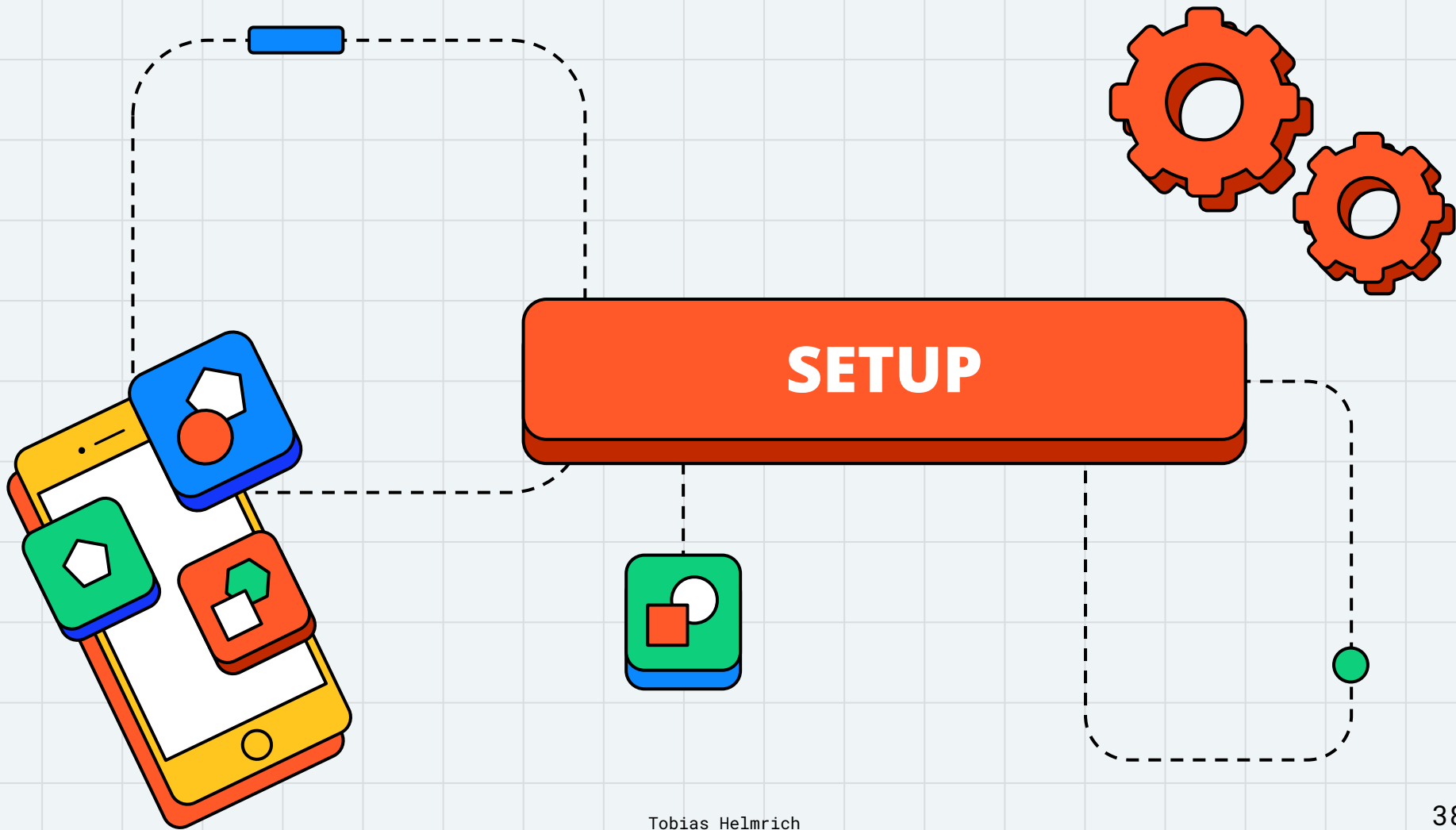
<https://storybook.js.org/>



# STORYBOOK



- Über 66.000 GitHub stars (Stand November 2021)
  - Framework-agnostisch (React, Vue, Svelte, Angular, etc.)
  - Warum?
    - Rahmenbedingungen, um UIs “richtig” entwickeln zu können:
      - Konfigurierter Development-Stack
      - Backend/API/Datenbank
- => Mit Storybook kann man Komponenten ohne großes Setup isoliert voneinander entwickeln, dokumentieren und testen
- 
- 



# HINZUFÜGEN VON STORYBOOK

```
npx sb init
```

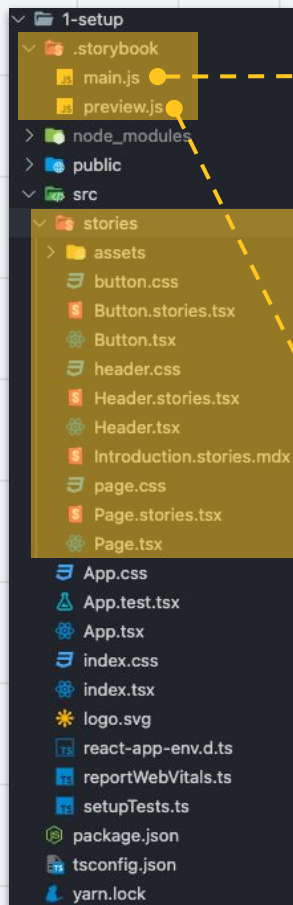
**ACHTUNG:** Storybook kann nur in einem Projekt installiert werden, in welchem bereits ein Framework aufgesetzt wurde! Storybook überprüft dann während der Installation die Projektdependencies und wählt so die passendste Konfiguration.

# EINSCHUB - BUG-ALARM!

- Stand CRA v4.0.3, Storybook v6.4.0 beim Ausführen mit ``npm run start`` oder ``yarn start`` Fehlermeldung weil Versionen der babel-loader-Packages von CRA und Storybook inkompatibel sind (siehe [GitHub Issue](#))
- Soll laut [Kommentar](#) unter Issue mit CRA v5-Release behoben werden
- **Temporäre Lösungen:**
  - Yarn resolutions
  - Explizites Setzen der babel-loader-Version
  - In .env-Datei ``SKIP_PREFLIGHT_CHECK=true`` setzen



# ORDNERSTRUKTUR NACH SETUP



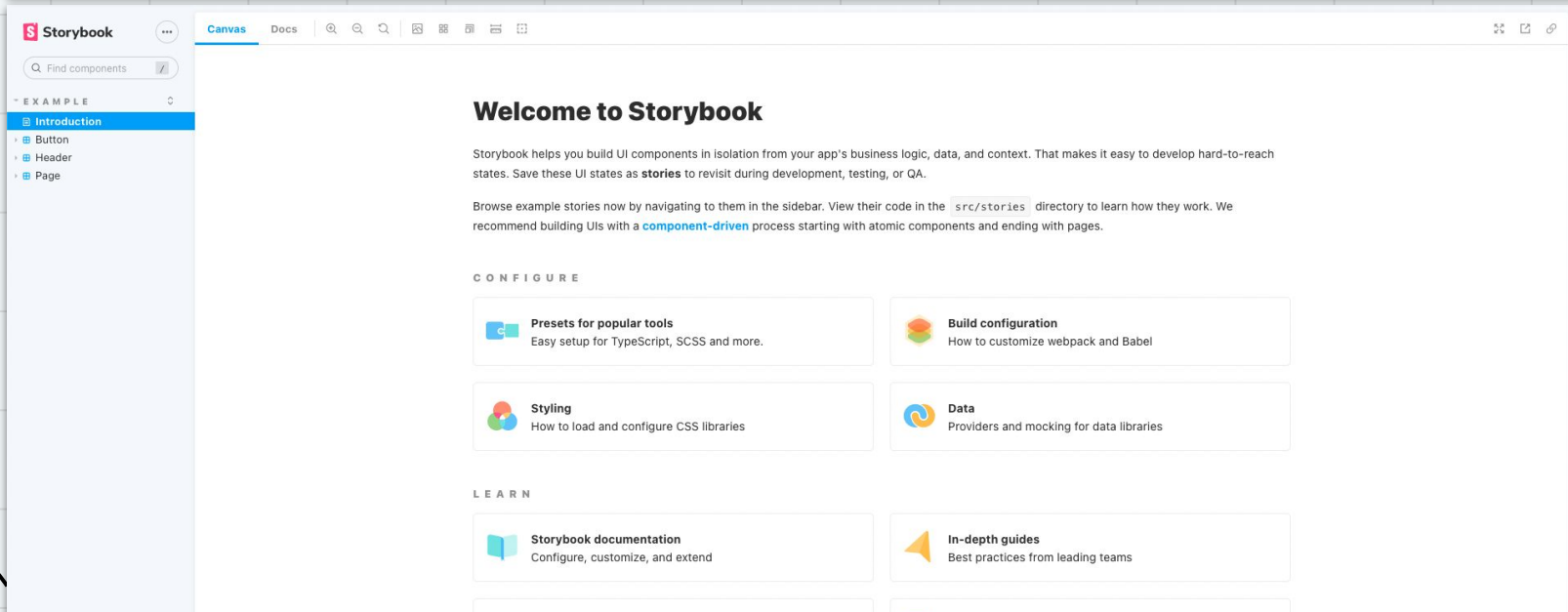
```
1 module.exports = {
2   stories: ['../src/**/*.stories.mdx',
3     '../src/**/*.stories.@(js|jsx|ts|tsx)'],
4   addons: [
5     '@storybook/addon-links',
6     '@storybook/addon-essentials',
7     '@storybook/preset-create-react-app',
8   ],
9 };
```

```
1 export const parameters = {
2   actions: { argTypesRegex: "^on[A-Z].*" },
3   controls: {
4     matchers: {
5       color: /(background|color)$/i,
6       date: /Date$/,
7     },
8   },
9 };
```

# ERSTER START

```
# - npm  
npm run storybook
```

```
# - yarn  
yarn storybook
```



The screenshot shows the Storybook web application interface. On the left, there is a sidebar with a search bar labeled "Find components" and a list of components under the heading "EXAMPLE". The components listed are "Introduction", "Button", "Header", and "Page". The "Introduction" component is currently selected and highlighted in blue. The main content area on the right displays the "Welcome to Storybook" page. This page includes a welcome message, a brief explanation of Storybook's purpose, and a section titled "CONFIGURE" with four cards: "Presets for popular tools", "Build configuration", "Styling", and "Data". Below this is a section titled "LEARN" with two cards: "Storybook documentation" and "In-depth guides".

**Welcome to Storybook**

Storybook helps you build UI components in isolation from your app's business logic, data, and context. That makes it easy to develop hard-to-reach states. Save these UI states as **stories** to revisit during development, testing, or QA.

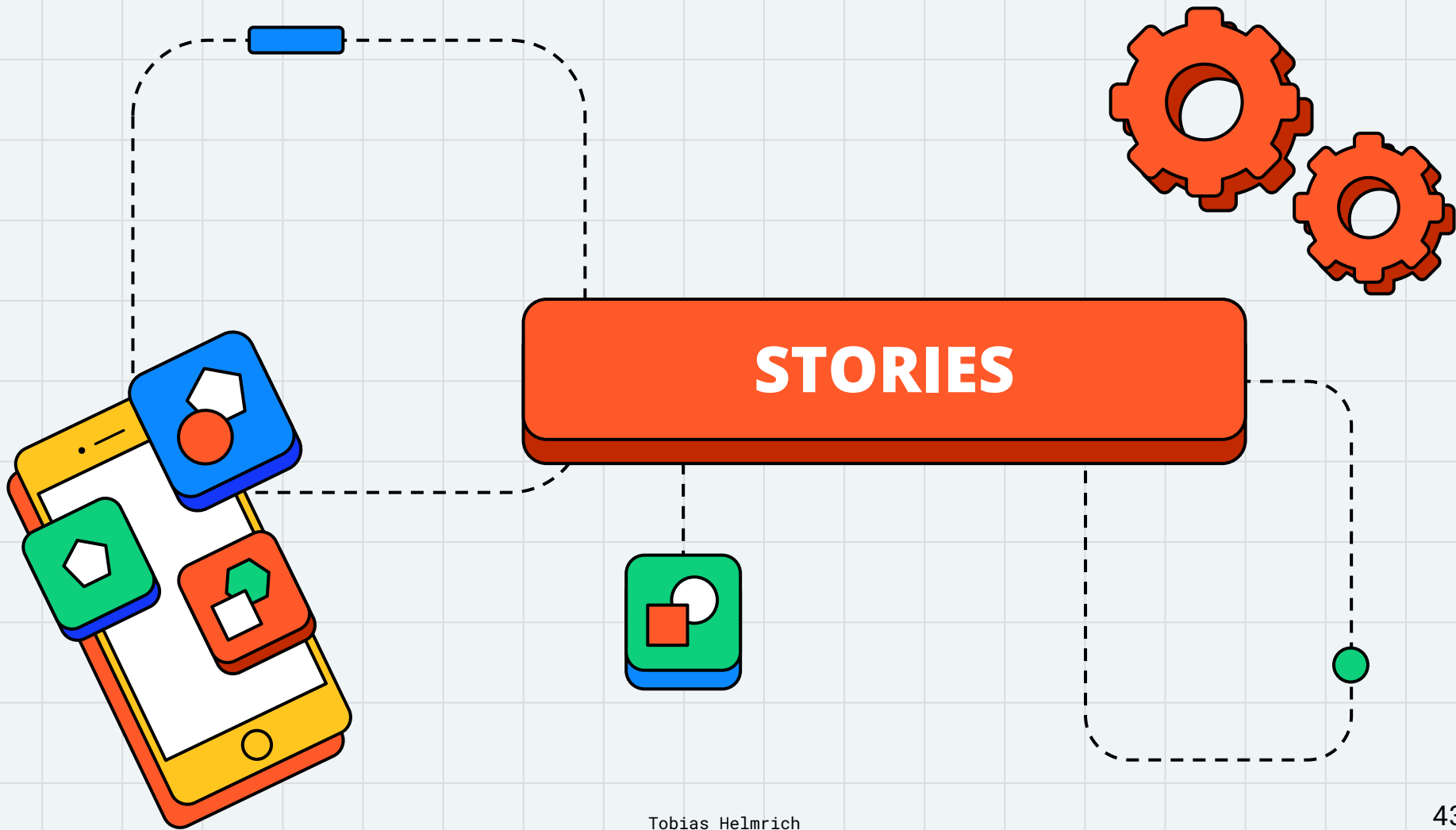
Browse example stories now by navigating to them in the sidebar. View their code in the `src/stories` directory to learn how they work. We recommend building UIs with a **component-driven** process starting with atomic components and ending with pages.

**CONFIGURE**

- Presets for popular tools**  
Easy setup for TypeScript, SCSS and more.
- Build configuration**  
How to customize webpack and Babel
- Styling**  
How to load and configure CSS libraries
- Data**  
Providers and mocking for data libraries

**LEARN**

- Storybook documentation**  
Configure, customize, and extend
- In-depth guides**  
Best practices from leading teams



# BUTTON-KOMPONENTE

```
// Button.tsx

1 import styled from 'styled-components';
2
3 export type ButtonProps = {
4   variant?: 'primary' | 'secondary';
5   size?: keyof Theme['fontSizes'];
6 };
7
8 const Button = styled.button<ButtonProps>`
9   background-color: ${({ theme, variant }) =>
10     variant === 'secondary' ? theme.colors.secondary : theme.colors.primary};
11   color: ${({ theme, variant }) =>
12     variant === 'secondary' ? theme.colors.onSecondary : theme.colors.onPrimary};
13   font-size: ${({ theme, size = 'medium' }) => theme.fontSizes[size]};
14   padding: 0.5rem 1rem;
15   border-radius: 8px;
16
17   &:hover {
18     filter: brightness(0.9);
19   }
20 `;
21
22 export default Button;
```

# STORIES

- **Story:** Funktion, die für übergebene Argumente den Zustand und somit den gerenderten Zustand einer Komponente zurückgibt
- Stories einer Komponente werden in ``.stories.[ts|tsx|js|jsx|mdx]``-Datei definiert
- Format der Stories: Component Story Format (CSF)
  - Komponenten-Metadaten als Default Export (muss vorhanden sein)
  - Jede Story ist ein Named Export

```
// Button.stories.tsx

1 import { Story, Meta } from '@storybook/react';
2 import Button, { ButtonProps } from './Button';
3
4 export default {
5   title: 'Controls/Button',
6   component: Button,
7 } as Meta<ButtonProps>;
8
9 export const Primary: Story<ButtonProps> = (args) => (
10   <Button { ... args}>Button</Button>
11 );
```

# STORY - STORYBOOK UI

The screenshot displays the Storybook interface for a 'Button' component. The left sidebar shows the 'CONTROLS' panel with 'Button' and 'Primary' variants. The main canvas shows a single blue button labeled 'Button'. The bottom panel, titled 'Controls (2)', lists the component's properties and their values.

Name	Control
size	<input type="radio"/> small <input type="radio"/> medium <input type="radio"/> large
variant	<input type="radio"/> primary <input type="radio"/> secondary

# EINSCHUB - 3 EBENEN

- In Storybook gibt es 3 Konfigurationsebenen:
  - Global - `preview.js`
  - Komponente - Default Export
  - Story - Named Export
- So können z.B. Argumente, Decorators oder Parameter auf Story-, Komponenten- oder globaler Ebene konfiguriert werden

```
1 export default {
2   title: 'Controls/Button',
3   component: Button,
4   args: {
5     size: 'medium',
6   },
7 } as Meta<ButtonProps>;
8
9 export const Primary: Story<ButtonProps> = (args) => (
10   <Button {... args}>Button</Button>
11 );
12 Primary.args = {
13   variant: 'primary',
14 };
```

Komponentenebene

Storyebene

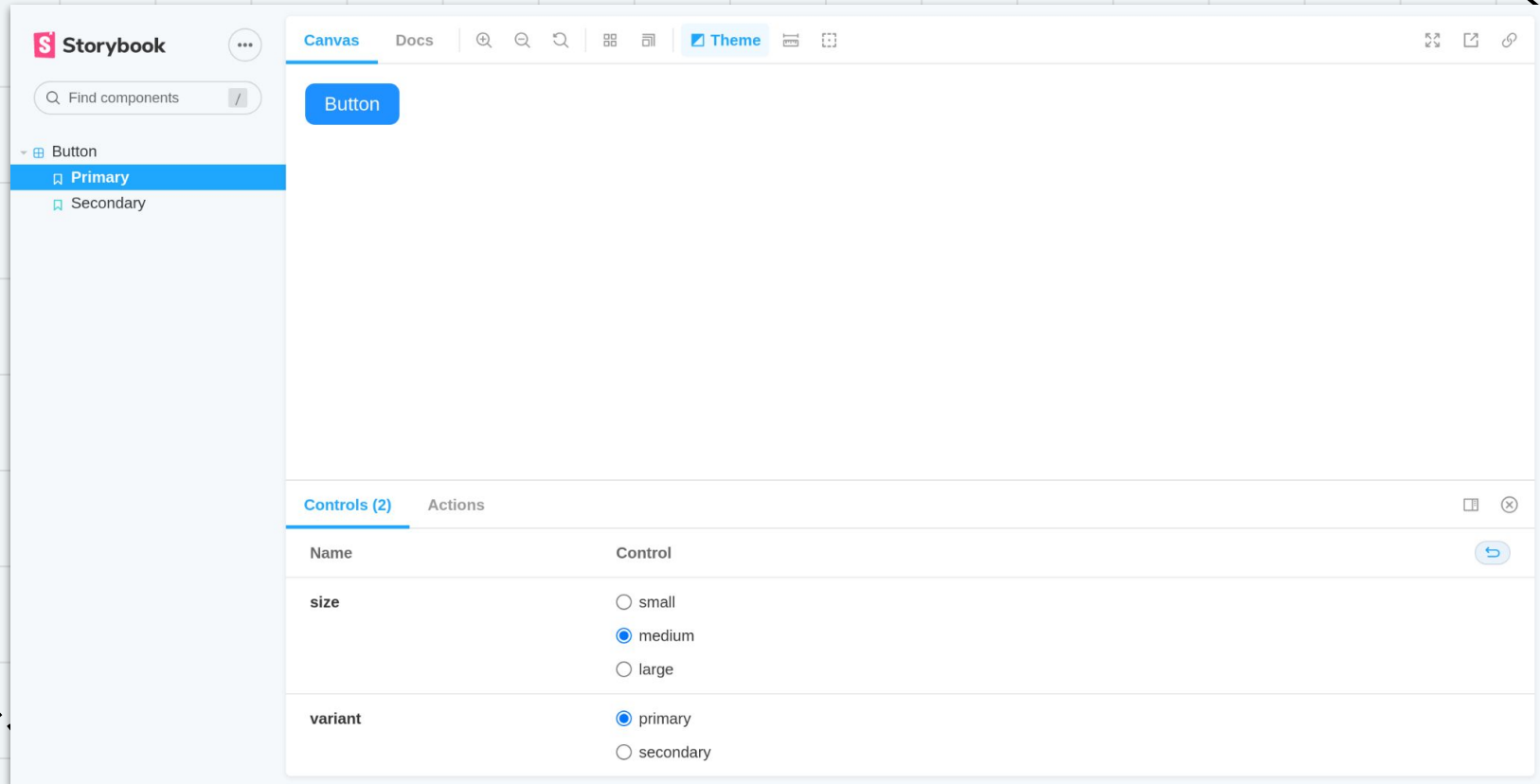
# TEMPLATE & ARGS

Verwenden eines wiederverwendbaren Templates für Stories

```
1  export default {
2    title: 'Button',
3    component: Button,
4    args: {
5      size: 'medium',
6    },
7  } as Meta<ButtonProps>;
8
9  const Template: Story<ButtonProps> = (args) => (
10    <Button { ... args}>Button</Button>
11  );
12
13  export const Primary = Template.bind({});
14  Primary.args = {
15    variant: 'primary',
16  };
17
18
19  export const Secondary = Template.bind({});
20  Secondary.args = {
21    variant: 'secondary',
22  };
```



# TEMPLATE - STORYBOOK UI



# ARGTYPES

- Definieren und Ändern von Argumenten
  - Documentation (``table``)
  - Verhalten von Addons wie ``control`` und ``action``

```
1 export default {  
2   title: 'Button',  
3   component: Button,  
4   argTypes: {  
5     children: {  
6       name: 'label',  
7     },  
8   },  
9   args: {  
10    children: 'Button',  
11    size: 'medium',  
12  },  
13 } as Meta<ButtonProps>;
```

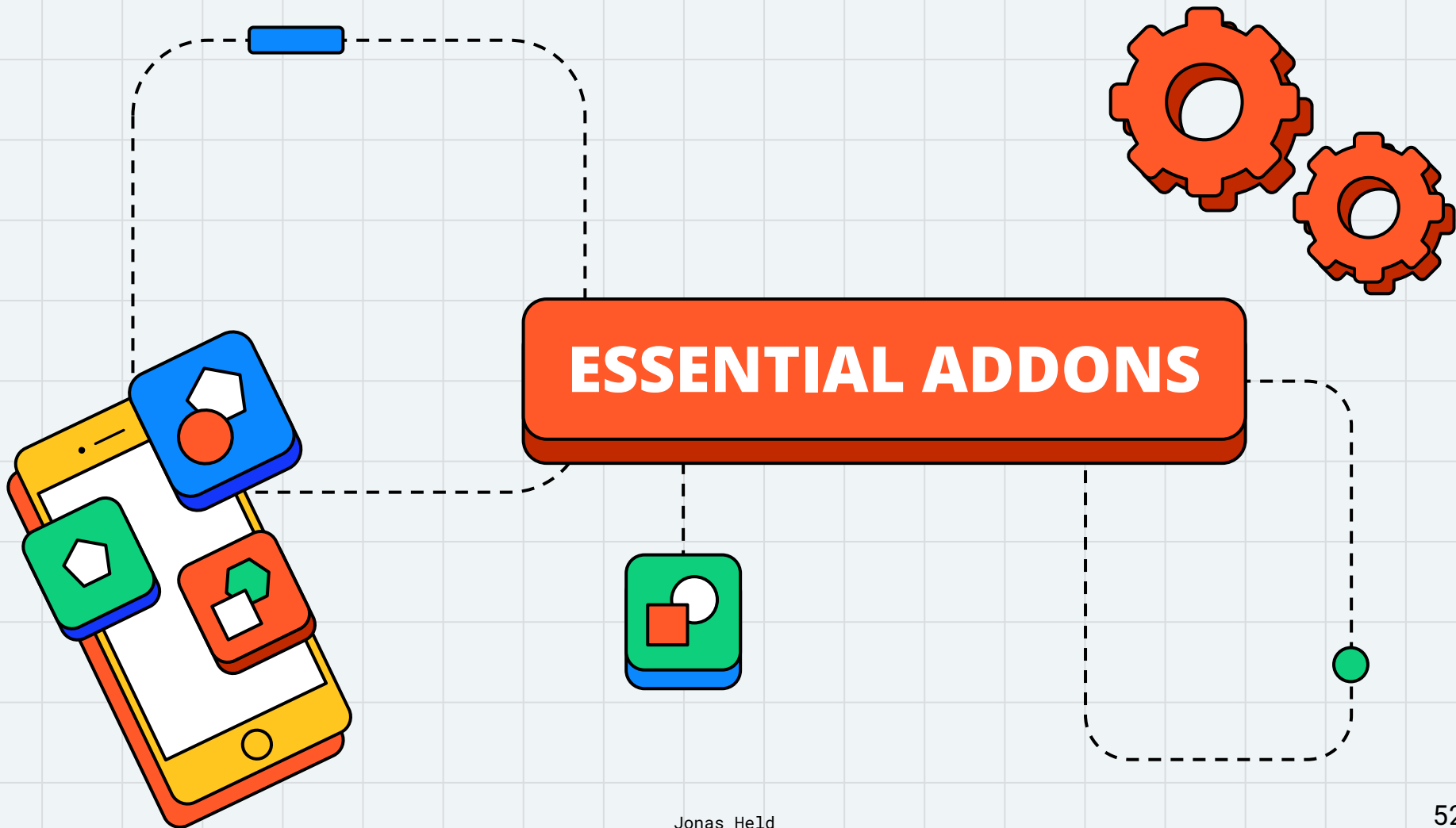
# ARGTYPES - STORYBOOK UI

The screenshot displays the Storybook interface for a Button component. The left sidebar shows a search bar and a component list with 'Button' expanded, highlighting 'Primary'. The main Canvas area shows a single blue 'Button' component. The bottom panel, titled 'Controls (3)', lists the component's props: 'label' (set to 'Button'), 'size' (set to 'medium'), and 'variant' (set to 'primary').

**Storybook UI Components:**

- Canvas:** Displays the rendered Button component.
- Controls (3):** A table of component props and their values.

Name	Control
label	Button
size	<input type="radio"/> small <input checked="" type="radio"/> medium <input type="radio"/> large
variant	<input checked="" type="radio"/> primary <input type="radio"/> secondary



# ESSENTIAL ADDONS

- Die meisten Features in Storybook sind Addons, die konfiguriert oder deaktiviert werden können
- Einige “Essentials” schon vorinstalliert und Teil der Standardfunktionalität von Storybook, wie z.B.:
  - Controls
  - Actions
  - Toolbars
  - Docs

# CONTROLS

- Input-Elemente, die es ermöglichen, Argumente dynamisch anzupassen
- Das Input-Element von Argumenten kann in ``argTypes`` unter ``control`` angepasst werden

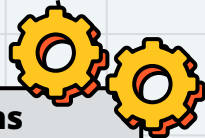
```
1 export default {
2   title: 'Button',
3   component: Button,
4   argTypes: {
5     size: {
6       control: {
7         type: 'select'
8       },
9       // ...
10    },
11    // ...
12  },
13 } as Meta<ButtonProps>;
```

# CONTROLS - STORYBOOK UI

The screenshot displays the Storybook interface. On the left, a sidebar contains the Storybook logo, a search bar labeled "Find components", and a component list with "Button" expanded, showing "Primary" and "Secondary" variants. The main area is titled "Canvas" and shows a "Button" component. Below the canvas, the "Controls (3)" panel is open, showing a table of controls for the "Button" component.

Name	Control
label	<input type="text" value="Button"/>
size	<input type="text" value="medium"/>
variant	<input checked="" type="radio"/> primary <input type="radio"/> secondary

# CONTROLS



Data Type	Control Type	Description	Options
<b>boolean</b>	boolean	checkbox input	
<b>number</b>	number	a numeric text box input	min, max, step
	range	a range slider input	min, max, step
<b>object / array</b>	object	json editor text input	
<b>array</b>	file	a file input that gives you a array of urls	accept
<b>enum</b>	radio	radio buttons input	
	inline-radio	inline radio buttons input	
	check	multi-select checkbox input	
	inline-check	multi-select inline checkbox input	
	select	select dropdown input	
	multi-select	multi-select dropdown input	
<b>string</b>	text	simple text input	
	color	color picker input that assumes strings are color values	presetColors
	date	date picker input	

<https://storybook.js.org/docs/react/essentials/controls>



# ACTIONS

Mit Actions kann man Argumente anzeigen die eine Callback-Funktion der Komponente erhalten hat

(Daten darstellen die ein Event-Handler übergeben bekommt)

```
1 export default {  
2   title: 'Button',  
3   component: Button,  
4   argTypes: {  
5     onClick: {  
6       action: 'click',  
7     },  
8     // ...  
9   },  
10  // ...  
11 } as Meta<ButtonProps>;
```

Controls (3) Actions (1)

» click: SyntheticBaseEvent {\_reactName: "onClick", \_targetInst: null, type: "click", nativeEvent: Object, target: undefined...}

# DOCS

- Alle Komponenten haben eine automatisch generierte zero-config DocsPage mit:
  - Allen Stories
  - Args Table
- Docs kann angepasst werden durch:
  - ``argTypes`` unter ``table``
  - Codekommentar
  - MDX
- **MDX** ist eine Mischung aus Markdown und JSX, welche mehr Anpassungsmöglichkeiten für die DocsPage ermöglicht.
  - Alternativ kann man sogar seine komplette Story in MDX definieren

# DOCS - STORYBOOK UI

The screenshot shows the Storybook Docs interface for a `Button` component. On the left, a sidebar lists the component and its variants: `Button`, `Primary` (selected), and `Secondary`. The main area is divided into three sections: a visual preview of the button, a code editor showing the component's props, and a table of prop documentation.

**Visual Preview:** A blue button with the text "Button".

**Code Editor:** Shows the JSX for the button with the following props:

```
<Button
  size="medium"
  variant="primary"
>
  Button
</Button>
```

**Prop Documentation Table:**

Name	Description	Default	Control
label	-	-	<input type="text" value="Button"/>
size	<code>"small"</code> <code>"medium"</code> <code>"large"</code>	-	<input type="radio"/> small <input checked="" type="radio"/> medium <input type="radio"/> large
variant	<code>"primary"</code> <code>"secondary"</code>	-	<input checked="" type="radio"/> primary <input type="radio"/> secondary

# DOCS - DEFAULTPROPS IN STYLED COMPONENTS

- Dem Docs-Generator (react-docgen-typescript) ist es nicht möglich aus einer Styled Component Default-Werte zu erkennen
- Man muss diese manuell angeben

```
1 export type ButtonProps = {
2   variant?: 'primary' | 'secondary';
3   size?: keyof Theme['fontSizes'];
4 };
5
6 const Button = styled.button<ButtonProps>`
7   // ...
8 `;
9
10 Button.defaultProps = {
11   variant: 'primary',
12   size: 'medium',
13 };
14
15 export default Button;
```

# ARGS TABLE VON CUSTOM ARGTYPES

- ``argTypes`` unter ``table``

```
1 export default {
2   title: 'Button',
3   component: Button,
4   argTypes: {
5     children: {
6       name: 'label',
7       table: {
8         type: { summary: 'string' },
9         defaultValue: { summary: 'Button' },
10      },
11    },
12  },
13  // ...
14 } as Meta<ButtonProps>;
```

# DOCS - STORYBOOK UI

Storybook

Find components /

Button

- Primary
- Secondary

Canvas Docs Theme

## Button

Button

Hide code

```
<Button size="large">
  Button
</Button>
```

Copy

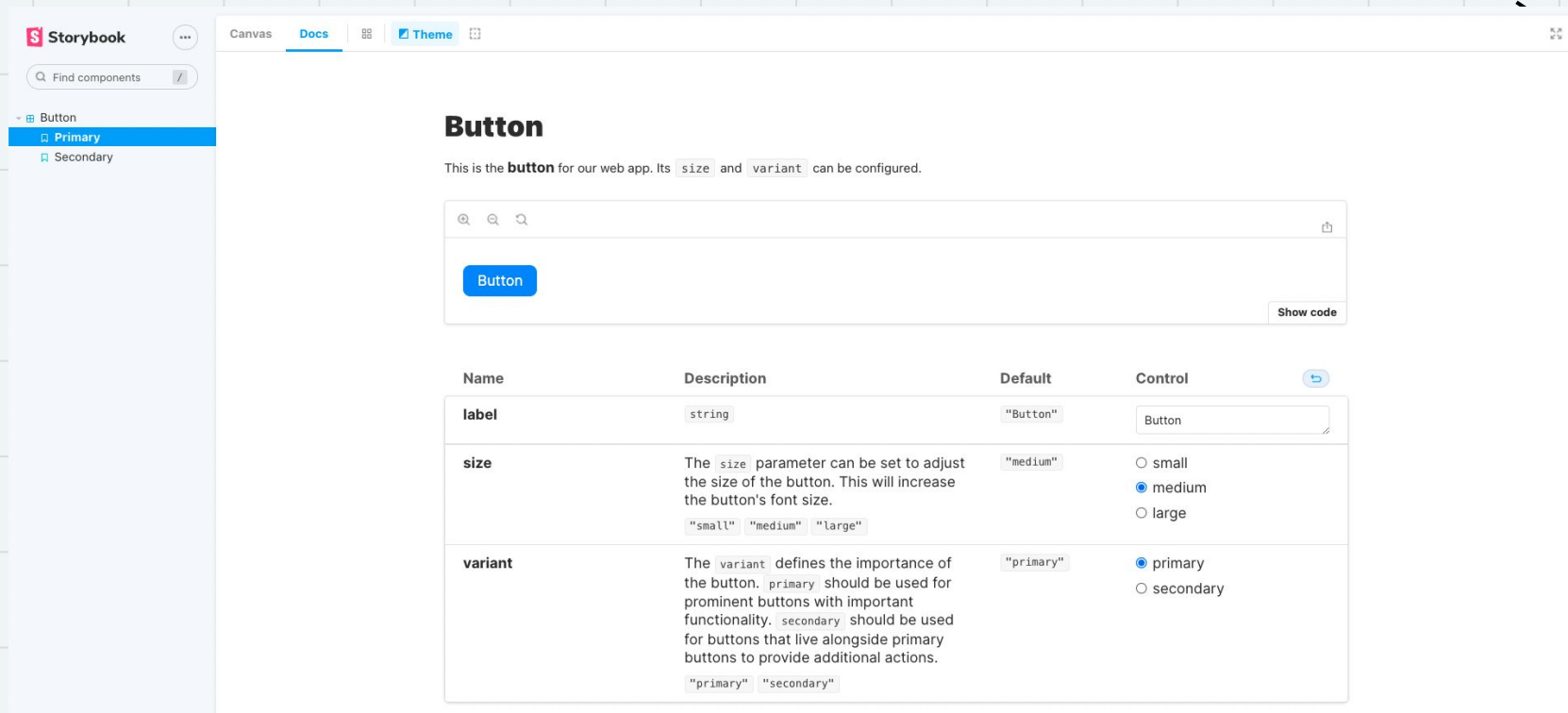
Name	Description	Default	Control
label	string	"Button"	<input type="text" value="Button"/>
size	"small" "medium" "large"	"medium"	<p><input type="radio"/> small</p> <p><input type="radio"/> medium</p> <p><input checked="" type="radio"/> large</p>
variant	"primary" "secondary"	"primary"	<p><input checked="" type="radio"/> primary</p> <p><input type="radio"/> secondary</p>

# DOCS DURCH KOMMENTARE

```
// Button.tsx

1 export type ButtonProps = {
2   /**
3    * The `variant` defines the importance of the button.
4    * `primary` should be used for prominent buttons with
5    * important functionality. `secondary` should be used for
6    * buttons that live alongside primary buttons to provide
7    * additional actions.
8    */
9   variant?: 'primary' | 'secondary';
10
11   /**
12    * The `size` parameter can be set to adjust the size of the
13    * button. This will increase the button's font size.
14    */
15   size?: keyof Theme['fontSizes'];
16 };
17
18 /**
19 * This is the button for our web app. Its `size` and `variant`
20 * can be configured.
21 */
22 const Button = styled.button<ButtonProps>`
23   // ...
24 `;
```

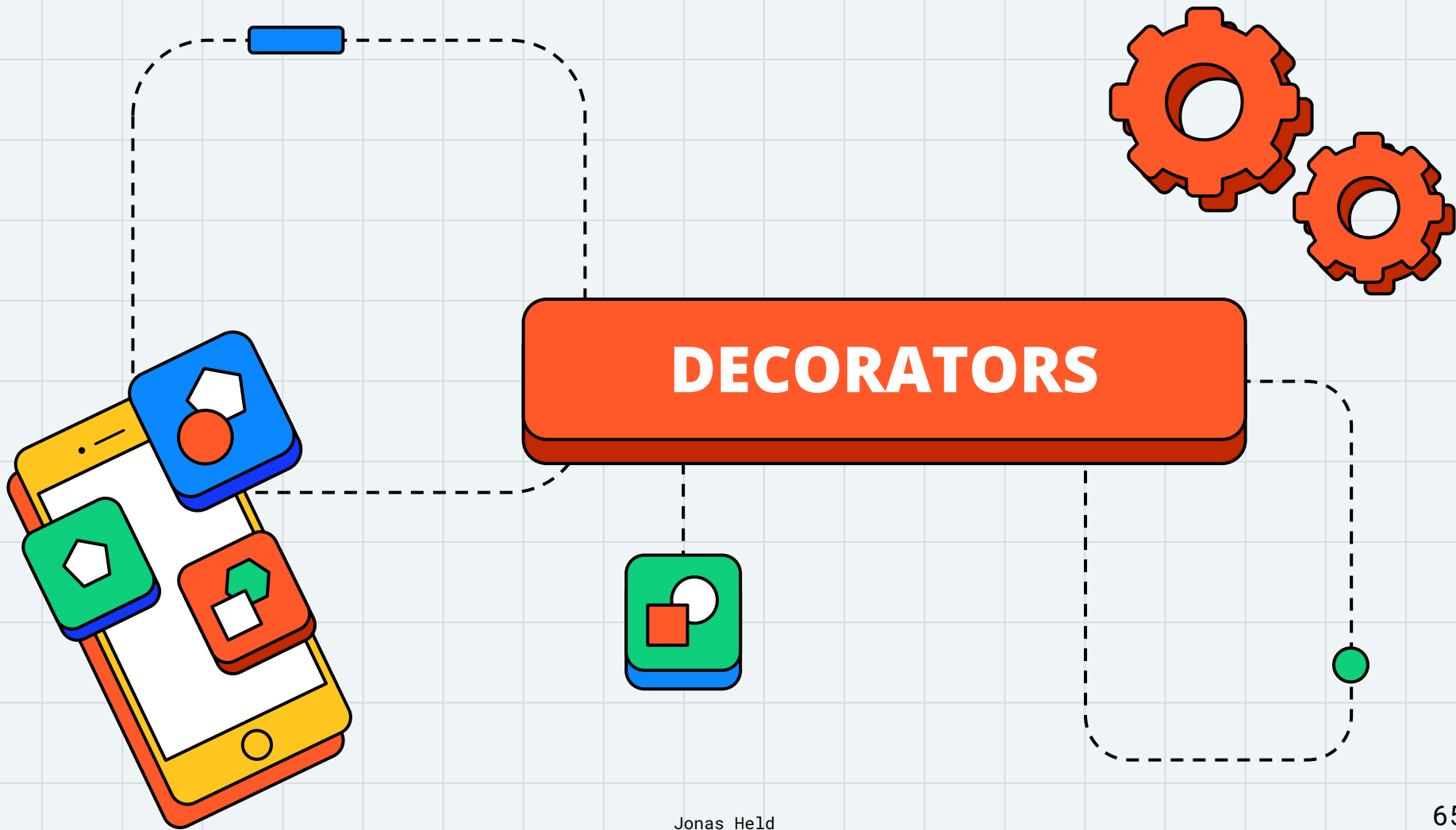
# DOCS DURCH KOMMENTARE - STORYBOOK UI



The screenshot displays the Storybook UI for the 'Button' component. On the left, a sidebar shows the component hierarchy: 'Button' (expanded), 'Primary' (selected), and 'Secondary'. The main area is titled 'Button' and includes a description: 'This is the **button** for our web app. Its `size` and `variant` can be configured.' Below this is a visual preview of a blue 'Button' component. To the right of the preview is a 'Show code' button. At the bottom, a table lists the component's props: 'label', 'size', and 'variant', each with a description, default value, and control options.

Name	Description	Default	Control
<b>label</b>	<code>string</code>	<code>"Button"</code>	<input type="text" value="Button"/>
<b>size</b>	The <code>size</code> parameter can be set to adjust the size of the button. This will increase the button's font size. <code>"small"</code> <code>"medium"</code> <code>"large"</code>	<code>"medium"</code>	<input type="radio"/> small <input checked="" type="radio"/> medium <input type="radio"/> large
<b>variant</b>	The <code>variant</code> defines the importance of the button. <code>primary</code> should be used for prominent buttons with important functionality. <code>secondary</code> should be used for buttons that live alongside primary buttons to provide additional actions. <code>"primary"</code> <code>"secondary"</code>	<code>"primary"</code>	<input checked="" type="radio"/> primary <input type="radio"/> secondary





# DECORATORS

- Decorators sind i.d.R. Wrapper um Stories, mit welchen man Funktionalität und Kontext (z.B. für Styling) oder (Mock-)Daten z.B. durch React-Kontext zur Verfügung stellen kann
- Ein Decorator bekommt 2 Argumente übergeben:
  - 1. Argument: Story
  - 2. Argument: Story-Context (args, globals, parameters, ...)
- Können auf allen 3 Ebenen definiert werden

```
// .storybook/preview.js  
  
1 export const decorators = [  
2   (Story, context) => (  
    // ...  
11  ),  
12  ];
```

# EINSCHUB - HEAD ERWEITERN

Wenn man den head z.B. mit links für Stylesheets oder Fonts erweitern will kann man im .storybook-Ordner eine preview-head.html-Datei erstellen und Tags hinzufügen:

```
// preview-head.html  
  
1 <link rel="preconnect" href="https://fonts.googleapis.com" />  
2 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />  
3 <link  
4   href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;700;900&display=swap"  
5   rel="stylesheet"  
6 />
```

Analog dazu kann auch der body angepasst werden, hierfür erstellt man stattdessen eine preview-body.html

# EINSCHUB - THEME

```
1 import Theme from './Theme';
2
3 const base: Omit<Theme, 'colors'> = {
4   fonts: {
5     primary: 'Inter, sans-serif',
6   },
7   fontSizes: {
8     small: '12px',
9     medium: '16px',
10    large: '20px',
11  },
12 };
13
14 // ...
```

```
// ...
14 export const light: Theme = {
15   ...base,
16   colors: {
17     primary: 'dodgerblue',
18     secondary: 'seagreen',
19     background: 'white',
20     onPrimary: 'white',
21     onSecondary: 'white',
22     onBackground: 'black',
23   },
24 };
25
26 export const dark: Theme = {
27   ...base,
28   colors: {
29     primary: 'dodgerblue',
30     secondary: 'seagreen',
31     background: 'black',
32     onPrimary: 'black',
33     onSecondary: 'black',
34     onBackground: 'white',
35   },
36 };
```

# EINSCHUB - GLOBALSTYLE-KOMPONENTE

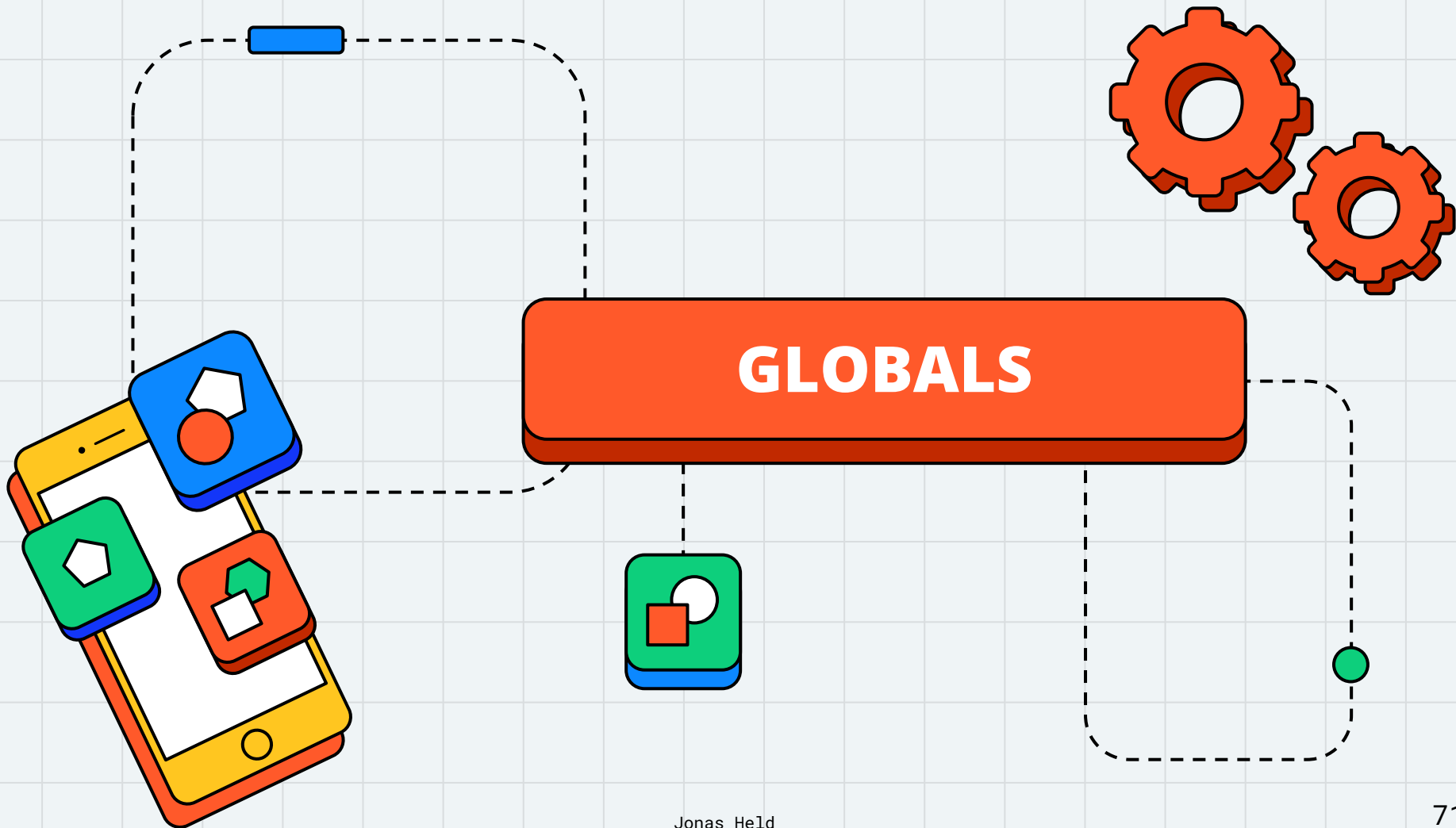
```
1 import { createGlobalStyle } from 'styled-components';
2
3 const GlobalStyle = createGlobalStyle`
4   * {
5     font-family: ${({ theme }) => theme.fonts.primary};
6     font-size: ${({ theme }) => theme.fontSizes.medium};
7   }
8   body {
9     background-color: ${({ theme }) => theme.colors.background};
10  }
11  button {
12    cursor: pointer;
13    border: 0;
14  }
15 `;
16
17 export default GlobalStyle;
```

# GLOBAL DECORATORS

Um die globalen Styles nun auf jede Komponente anzuwenden, können wir einen Decorator schreiben. Hierzu exportieren wir das ``decorators``-Array in `preview.js`:

```
// .storybook/preview.js

1 import { ThemeProvider } from 'styled-components';
2 import { light, dark } from '../src/theme';
3 import GlobalStyle from '../src/components/Globalstyle/Globalstyle';
4
5 export const decorators = [
6   (Story) => (
7     <ThemeProvider theme={light}>
8       <GlobalStyle />
9       <Story />
10    </ThemeProvider>
11  ),
12 ];
```



# GLOBALS

- Globale Daten für Stories
  - Zugriff über `context.globals`
  - Verwendung meist in globalen Decorators
  - z.B. Definieren von Themes
- Können der Toolbar hinzugefügt werden



# GLOBALTYPES & TOOLBAR

```
// .storybook/preview.js

1 import { ThemeProvider } from 'styled-components';
2 import { light, dark } from '../src/theme';
3 import GlobalStyle from '../src/components/Globalstyle/Globalstyle';
4
5 export const globalTypes = {
6   theme: {
7     name: 'Theme',
8     description: 'Styled Component theme',
9     defaultValue: 'light',
10    toolbar: {
11      icon: 'mirror',
12      items: ['light', 'dark'],
13      showName: true,
14    },
15  },
16  };
17
```

Canvas

Docs



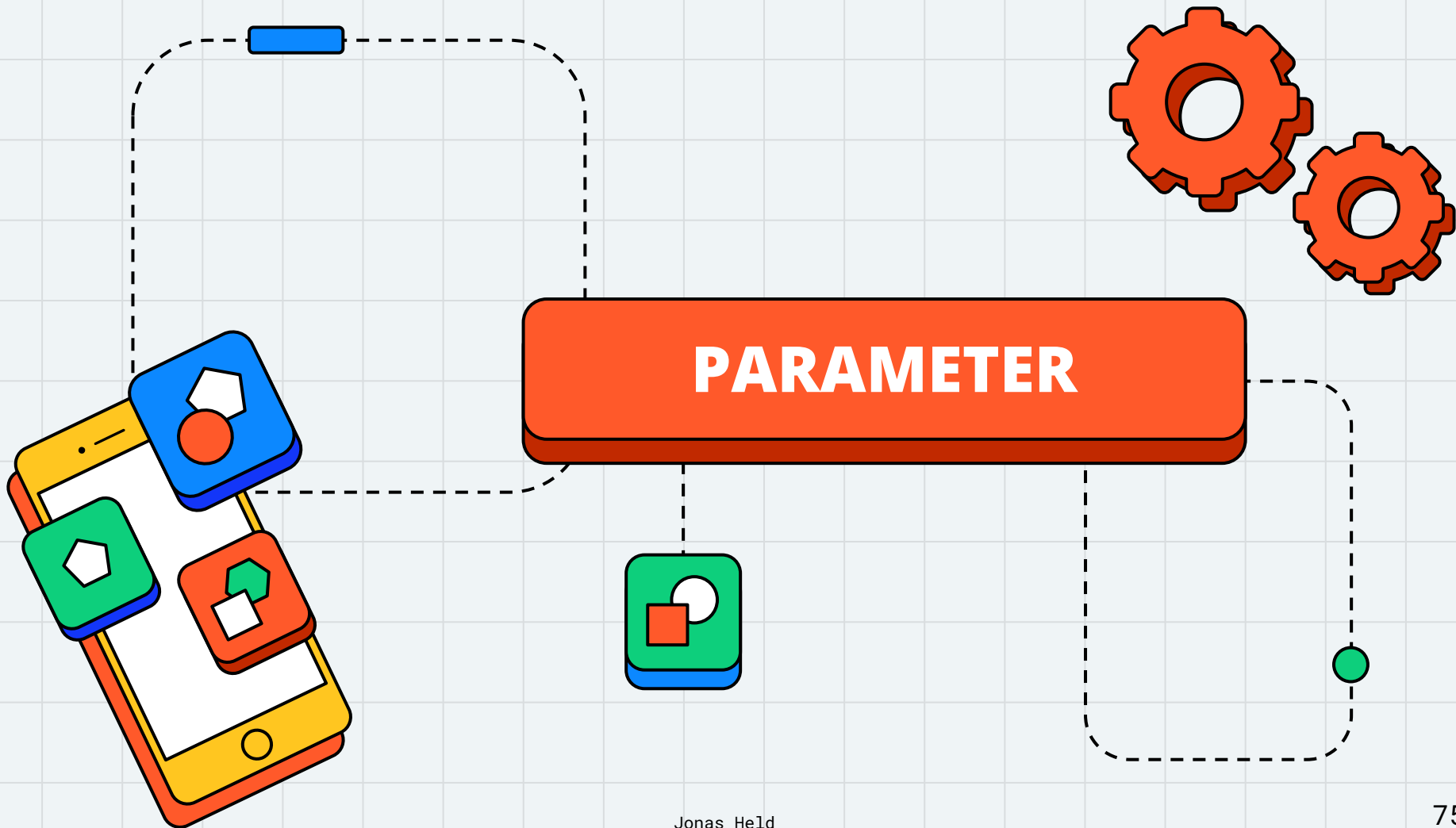
Theme



# DECORATOR WITH GLOBALS

```
// .storybook/preview.js

// ...
18 export const decorators = [
19   (Story, context) => (
20     <ThemeProvider theme={context.globals.theme === 'dark' ? dark : light}>
21       <GlobalStyle />
22       <Story />
23     </ThemeProvider>
24   ),
25 ];
```



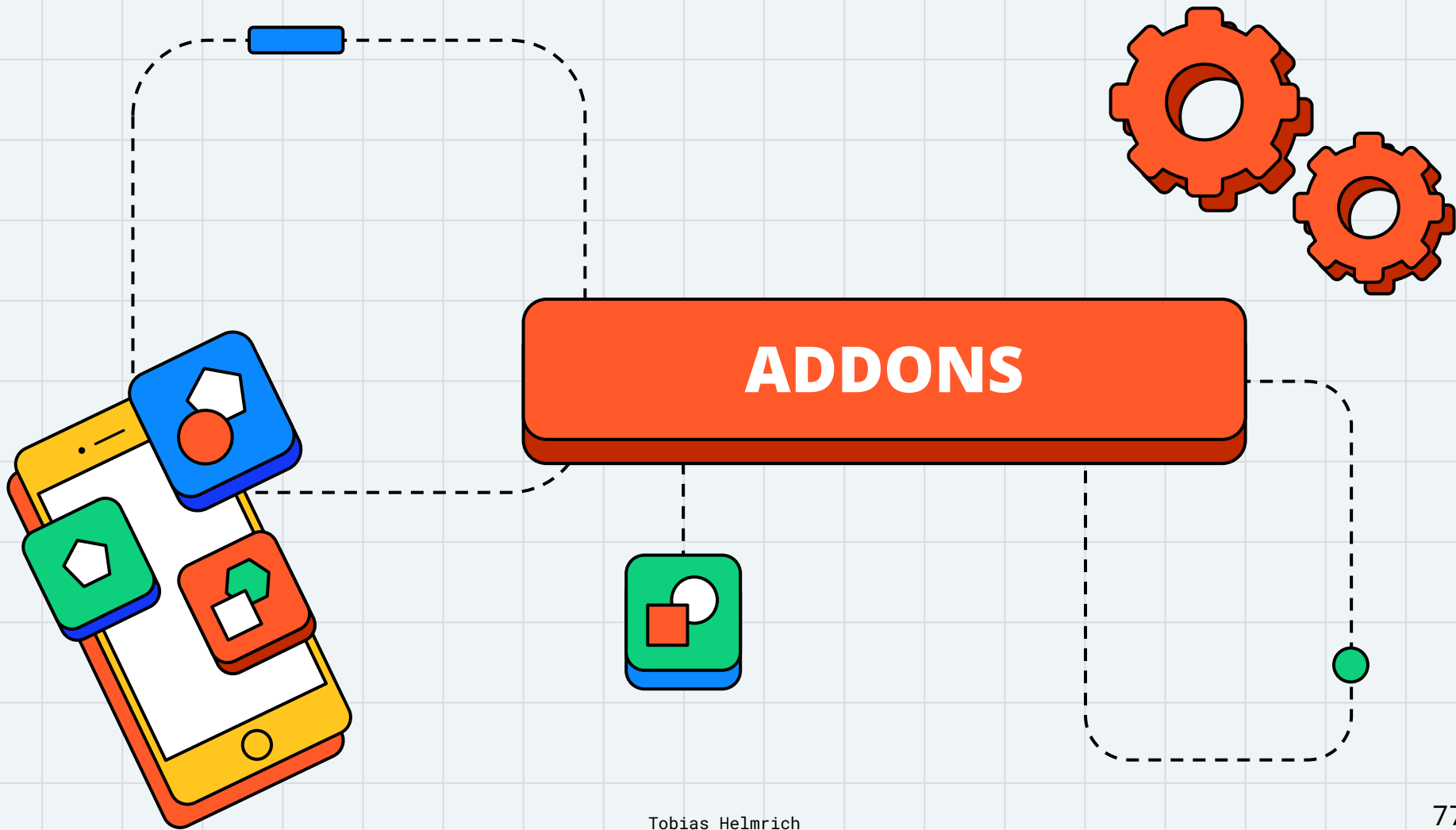
# PARAMETER

- Mit Parametern kann man Storybook-Features und Addons konfigurieren
- Können auf allen 3 Ebenen definiert werden

```
// .storybook/preview.js

// ...
27 export const parameters = {
28   actions: {
29     // ...
30   },
31   controls: {
32     // ...
33     sort: 'alpha',
34     exclude: ['ref', 'theme', 'as', 'forwardedAs'],
35   },
36   backgrounds: { disable: true },
37 };

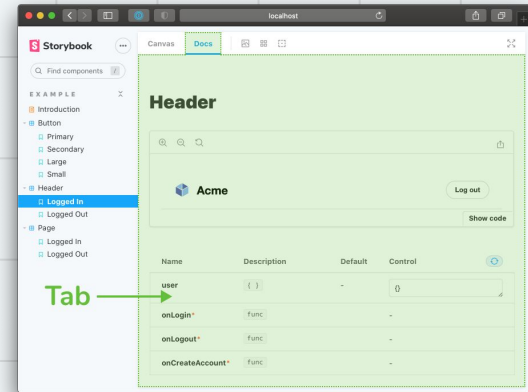
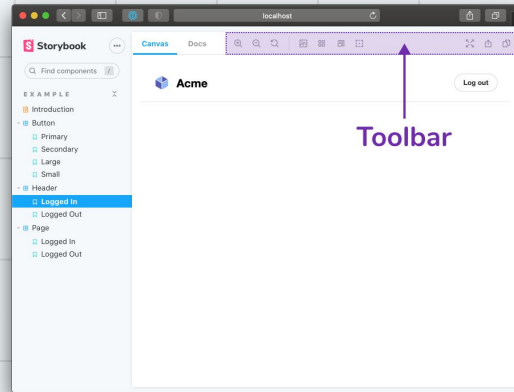
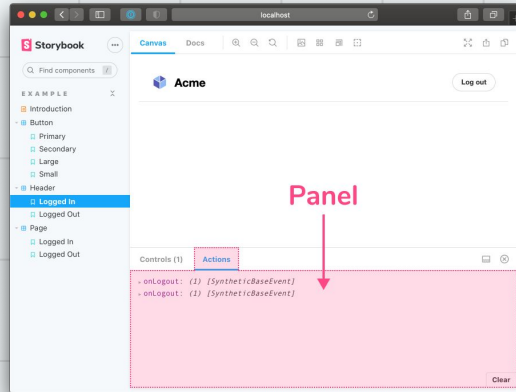
```



# ADDONS

- Umfassender Addon-Katalog mit “offiziellen” Addons und Addons von anderen Entwicklern für verschiedene Einsatzzwecke, Frameworks und Libraries, mit welchen man Storybook erweitern kann
- Auch eigenes Erweitern durch Addons möglich
- Arten von Addons:
  - UI-based-Addons (z.B. Controls oder Toolbars)
  - Preset-Addons (z.B. preset-create-react-app)

# ARTEN VON UI ADDONS



# INSTALLATION EINES ADDONS - BEISPIEL: A11Y

```
# - npm  
npm install --save-dev @storybook/addon-a11y
```

```
# - yarn  
yarn add --dev @storybook/addon-a11y
```

```
// .storybook/main.js  
  
1 module.exports = {  
1   // ...  
1   addons: [  
1     // ...  
1     '@storybook/addon-a11y'  
1   ],  
1   // ...  
1 };  
  
1
```



# A11Y-ADDON

Storybook

Find components

CONTROLS

Button

Primary

Canvas Docs Theme

Button

Controls (2) Actions Accessibility

1 Violations 3 Passes 0 Incomplete

Highlight results

Elements must have sufficient color contrast

Ensures the contrast between foreground and background colors meets WCAG 2 AA contrast ratio thresholds

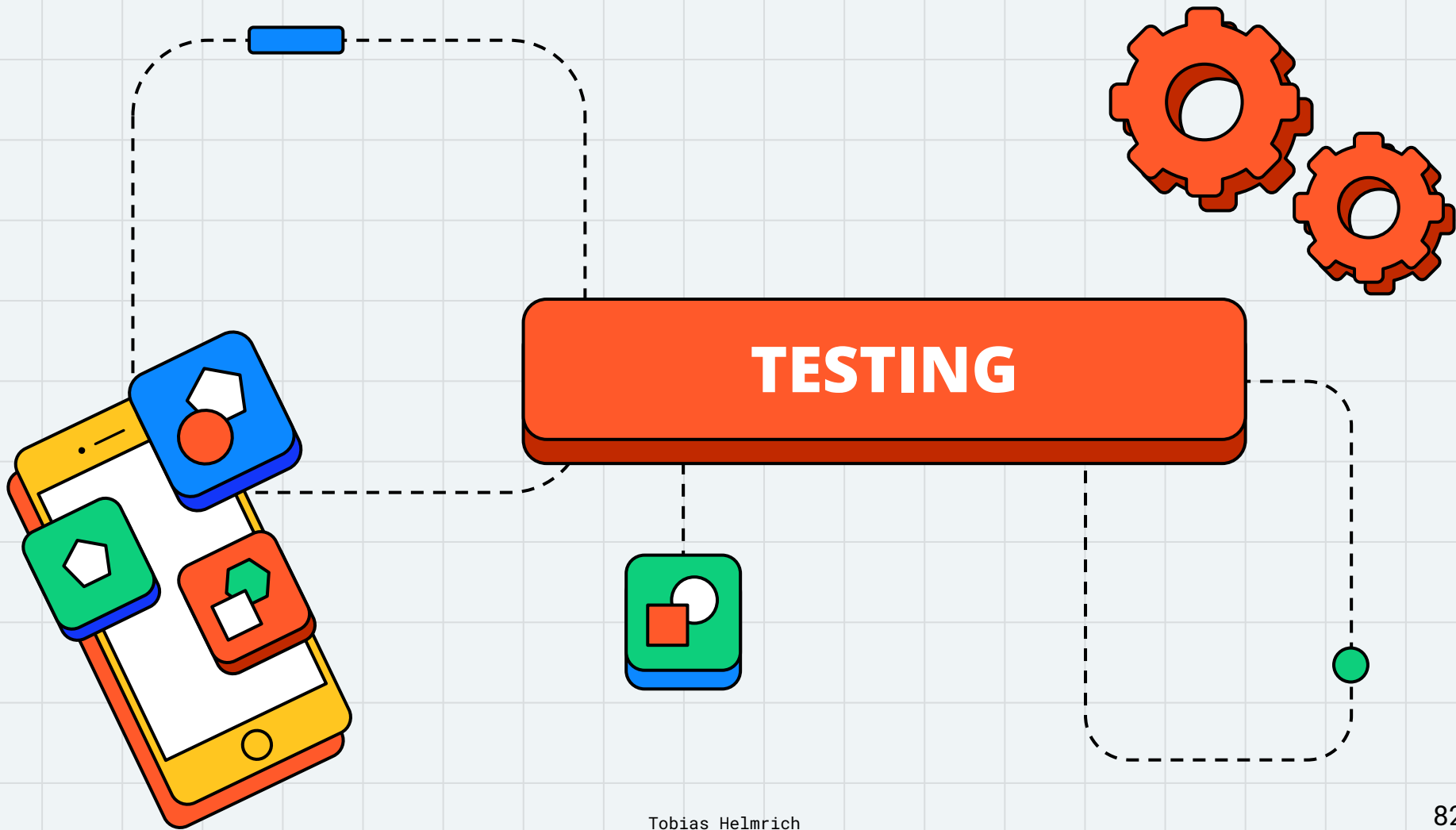
[More info...](#)

1. `.sc-bdvvtL`

**Serious**  Element has insufficient color contrast of 3.23 (foreground color: #ffffff, background color: #1e90ff, font size: 12.0pt (16px), font weight: normal). Expected contrast ratio of 4.5:1

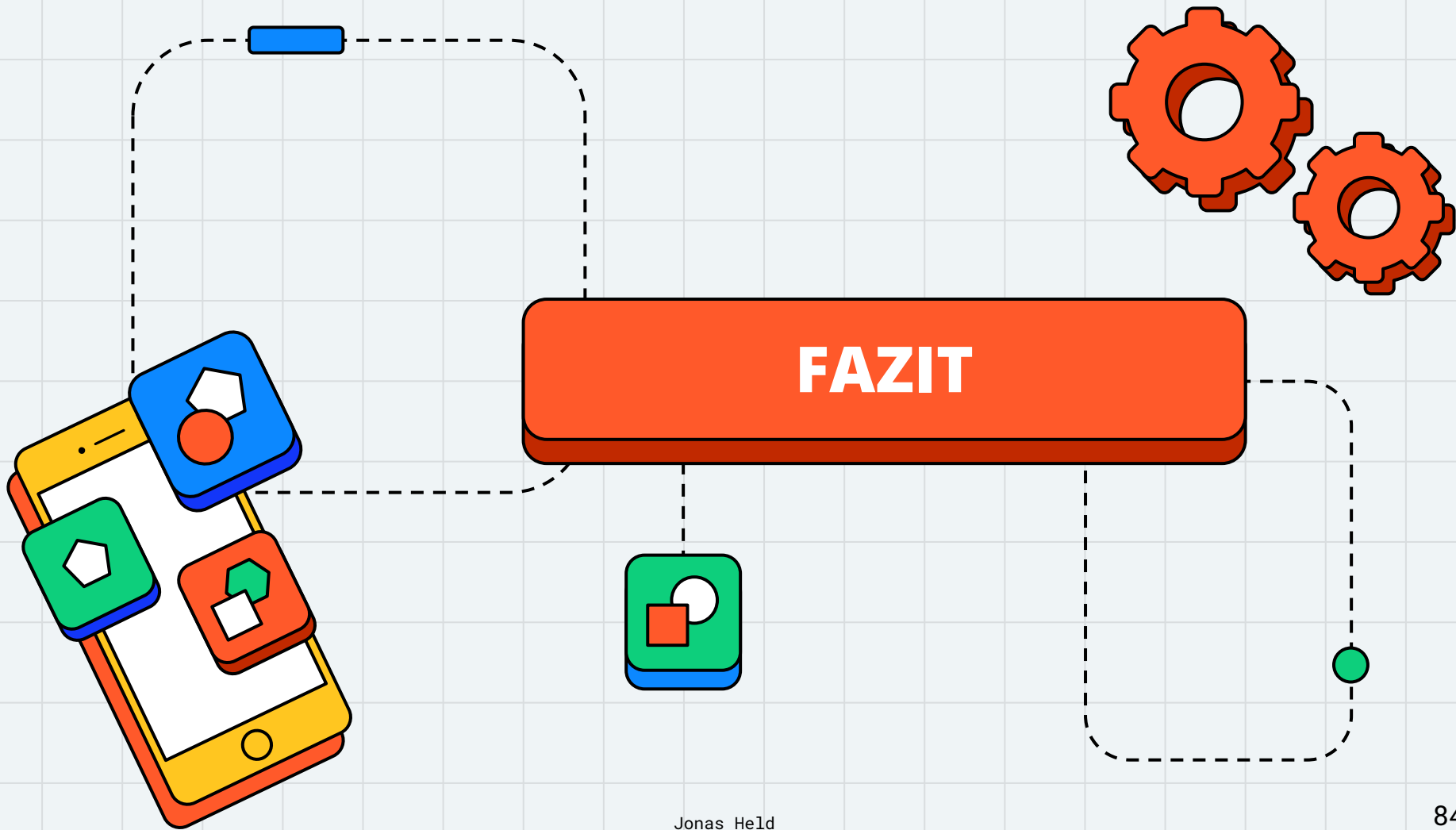
cat.color wcag2aa wcag143

Tests completed



# TESTING

- Manuelles Testen von Komponenten durch Ansehen mithilfe verschiedener Tools als Teil des Grundprinzips von Storybook
- Integrieren weiterer Tests durch Addons möglich:
  - Accessibility-Tests mit a11y-Addon
  - Visual- und Snapshot-Tests mit StoryShots-Addon
  - Interaction-Tests mit Interactions-Addon





## VORTEILE

- Gute, umfangreiche "offizielle" Tutorials
- Reibungsloses Setup für viele Frameworks
- Zero-config Doc-Generation
- Sehr anpassbar
- Ermöglicht "bottom up"-Entwicklung
- Erleichtert Kollaboration zwischen Designern und Entwicklern
- "Eigene Component-Library"

## NACHTEILE

- Unvollständige Docs/API-Spezifikation
- Schwierigkeiten bei Doc-Generation im Zusammenspiel mit Styled Components
- Teilweise mangelhafte Typisierung
- Hot Reload mit Problemen

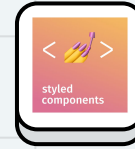
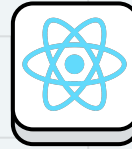


THANK

YOU!

# FRAGEN?

Tobias Helmrich, th138  
Jonas Held, jh257



CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**

# QUELLEN

- <https://storybook.js.org/docs/react/get-started/introduction>
- <https://storybook.js.org/tutorials/intro-to-storybook/>
- <https://storybook.js.org/tutorials/design-systems-for-developers/>
- [https://www.youtube.com/watch?v=2j9rSur\\_mnk](https://www.youtube.com/watch?v=2j9rSur_mnk)
- [https://youtu.be/BkgU\\_-KGK9w](https://youtu.be/BkgU_-KGK9w)
- <https://www.youtube.com/watch?v=1C4Kf9Pqu0k>
- <https://www.componentdriven.org/>
- <https://bradfrost.com/blog/post/atomic-web-design/>
- <https://www.youtube.com/watch?v=UPrn14gBmhc>
- <https://styled-components.com/docs>
- <https://rangle.io/blog/styled-components-styled-systems-and-how-they-work/>