Storybook

Storybook is a tool which we use to manage our design system and the components which are a part of it. Storybook consists of *stories:* each story represents a component and a case in which it is used. To show a wide variety of use cases is good both documentation wise and for troubleshooting – it might be possible to reproduce a bug for an edge case in a story.

Storybook is:

- A good way to publish our design system with its implementations
- Used as a tool for documentation
- Used for debugging and displaying edge cases

How to create stories

Stories for a component should be placed next to the component file. The Storybook file requires the same name as the component file. For example, a story for `SomeComponent.tsx` will have the file name `SomeComponent.story.tsx`. If a story should be internal, not visible in production, name the file `SomeComponent.story.internal.tsx`.

Writing stories

When writing stories, we use the CSF format. For more in-depth information on writing stories, see Storybook's documentation on writing stories.

With the CSF format, the default export defines some general information about the stories in the file:

- `title`: Where the component is going to live in the hierarchy
- `decorators`: A list which can contain wrappers or provide context, such as theming

```
// In MyComponent.story.tsx

import MyComponent from './MyComponent';

export default {
  title: 'General/MyComponent',
  component: MyComponent,
  decorators: [ ... ],
}
```

When it comes to writing the actual stories, you continue in the same file with named exports. The exports are turned into the story name.

```
// Will produce a story name "some story"
export const someStory = () => <MyComponent />;
```

If you want to write cover cases with different values for props, then using knobs is usually enough. You don't need to create a new story. This will be covered further down.

Categorization

We currently have these categories:

- **Docs Overview** - Guidelines and information regarding the design system
- **Forms** - Components commonly used in forms such as different kind of inputs
- **General** - Components which can be used in a lot of different places
- **Visualizations** - Data visualizations
- **Panel** - Components belonging to panels and panel editors

Writing MDX documentation

An MDX file is basically a markdown file with the possibility to add jsx. These files are used by Storybook to create a "docs" tab.

Link the MDX file to a component's stories

To link a component's stories with an MDX file you have to do this:

```
// In TabsBar.story.tsx

import { TabsBar } from './TabsBar';

// Import the MDX file
import mdx from './TabsBar.mdx';

export default {
  title: 'General/Tabs/TabsBar',
  component: TabsBar,
  parameters: {
    docs: {
      // This is the reference required for the MDX file
      page: mdx,
    },
  },
};
```

MDX file structure

There are some things that the MDX file should contain:

- When and why the component should be used
- Best practices - dos and don'ts for the component

- Usage examples with code. It is possible to use the `Preview` element to show live examples in MDX
- Props table. This can be generated by doing the following:

```
// In MyComponent.mdx

import { Props } from '@storybook/addon-docs/blocks';
import { MyComponent } from './MyComponent';

<Props of={MyComponent} />;
```

MDX file without a relationship to a component

An MDX file can exist by itself without any connection to a story. This can be good for writing things such as a general guidelines page. Two things are required for this to work:

- The file needs to be named `*.story.mdx`
- A `Meta` tag must exist that says where in the hierarchy the component lives. It can look like this:

```
<Meta title="Docs Overview/Color Palettes"/>

# Guidelines for using colors

...
```

You can add parameters to the Meta tag. This example shows how to hide the tools:

```
<Meta title="Docs Overview/Color Palettes" parameters={{ options: { isToolshown: false }}}/>

# Guidelines for using colors

...
```

Documenting component properties

A quick way to get an overview of what a component does is by looking at its properties. That's why it is important that we document these in a good way.

Comments

When writing the props interface for a component, it is possible to add a comment to that specific property, which will end up in the Props table in the MDX file. The comments are generated by react-docgen and are formatted by writing `/** */`.

```
interface MyProps {
  /** Sets the initial values, which are overridden when the query returns a value*/
  defaultValues: Array<T>;
}
```

Controls

The controls addon provides a way to interact with a component's properties dynamically and requires much less code than knobs. We're deprecating knobs in favor of using controls.

Migrating a story from Knobs to Controls As a test, we migrated the button story. Here's the guide on how to migrate a story to controls.

1. Remove the `@storybook/addon-knobs` dependency.
2. Import the Story type from `@storybook/react`

   ```
   import { Story } from @storybook/react
   ```

3. Import the props interface from the component you're working on (these must be exported in the component).

   ```
   import { Props } from './Component'
   ```

4. Add the Story type to all stories in the file, then replace the props sent to the component and remove any knobs.

   Before

   ```
   export const Simple = () => {
     const prop1 = text('Prop1', 'Example text');
     const prop2 = select('Prop2', ['option1', 'option2'], 'option1');

     return <Component prop1={prop1} prop2={prop2} />;
   };
   ```

   After

   ```
   export const Simple: Story<Props> = ({ prop1, prop2 }) => {
     return <Component prop1={prop1} prop2={prop2} />;
   };
   ```

5. Add default props (or args in Storybook language).

   ```
   Simple.args = {
     prop1: 'Example text',
     prop2: 'option 1',
   };
   ```

6. If the component has advanced props type (ie. other than string, number, boolean), you need to specify these in an `argTypes`. This is done in the default export of the story.

```
export default {
  title: 'Component/Component',
  component: Component,
  argTypes: {
    prop2: { control: { type: 'select', options: ['option1', 'option2'] } },
  },
};
```

Best practices

- When creating a new component or writing documentation for an existing one, always cover the basic use case it was intended for with a code example.
- Use stories and knobs to create edge cases. If you are trying to solve a bug, try to reproduce it with a story.
- Do not create stories in the MDX, always create them in the `*.story.tsx` file.