Week 05: Component libraries

Agenda

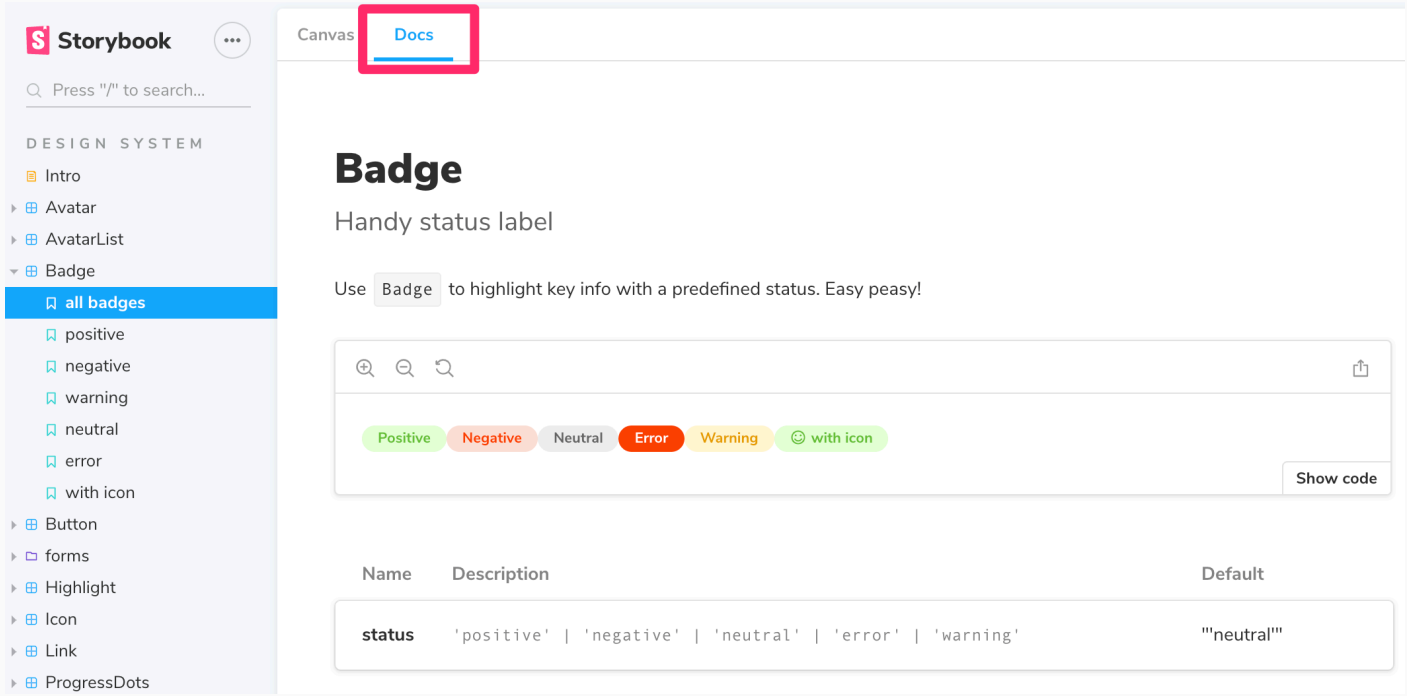
- Recap
- Storybook
- Hooks
- React Query

Recap

Let us recap the lecture with a Kahoot!

Storybook

A frontend workshop for seamless design, testing, and documentation of UI components.



Storybook - Examples

Examples of Storybooks that you may know:

- JetBrains
- Gitlab

Storybook - Features

- **Component Explorer**: Browse a component library, view the different states of each component and develop them interactively.
- **Documentation**: Document components directly within your Storybook.
- **Testing**: Write visual test cases for your components to prevent bugs.

Storybook - Installation

```
npx storybook@latest init
```

Choose **`React + Vite`** as your template.

Storybook should run automatically. The next time you want to run it, use:

```
npm run storybook
```

Make sure to check the docs

Storybook - Your first story

Let's create a simple Avatar story. We start by creating a new `avatar.tsx` file in the `src/stories` folder.

```tsx
import React from "react";
import "./avatar.css";

export interface AvatarProps {
  src: string;
  alt: string;
  size?: "small" | "medium" | "large";
}

const Avatar: React.FC<AvatarProps> = ({ src, alt, size = "medium" }) => {
  return <img src={src} alt={alt} className={`avatar avatar--${size}`} />;
};

export default Avatar;
```

Storybook - Your first story

Next we create give our component basic styles in `avatar.css`:

```
.avatar {
  width: 200px;
  height: 200x;
  border-radius: 50%;
  object-fit: cover;
}
.avatar--small {
  width: 100px;
  height: 100px;
}
.avatar--medium {
  width: 200px;
  height: 200px;
}
.avatar--large {
  width: 300px;
  height: 300px;
}
```

Storybook - Your first story

Finally, we register a new story for our Avatar component in `avatar.stories.tsx`:

```tsx
import type { Meta, StoryObj } from "@storybook/react";
import Avatar from "./avatar";

const meta: Meta<typeof Avatar> = {
  title: "Example/Avatar",
  component: Avatar,
  parameters: {
    layout: "centered",
  },
  tags: ["autodocs"],
  argTypes: {
    size: {
      control: "select",
      options: ["small", "medium", "large"],
    },
    src: { control: "text" },
    alt: { control: "text" },
  },
} satisfies Meta<typeof Avatar>;
```

Storybook - Your first story

... export the meta object

```
export default meta;
```

Storybook - Your first story

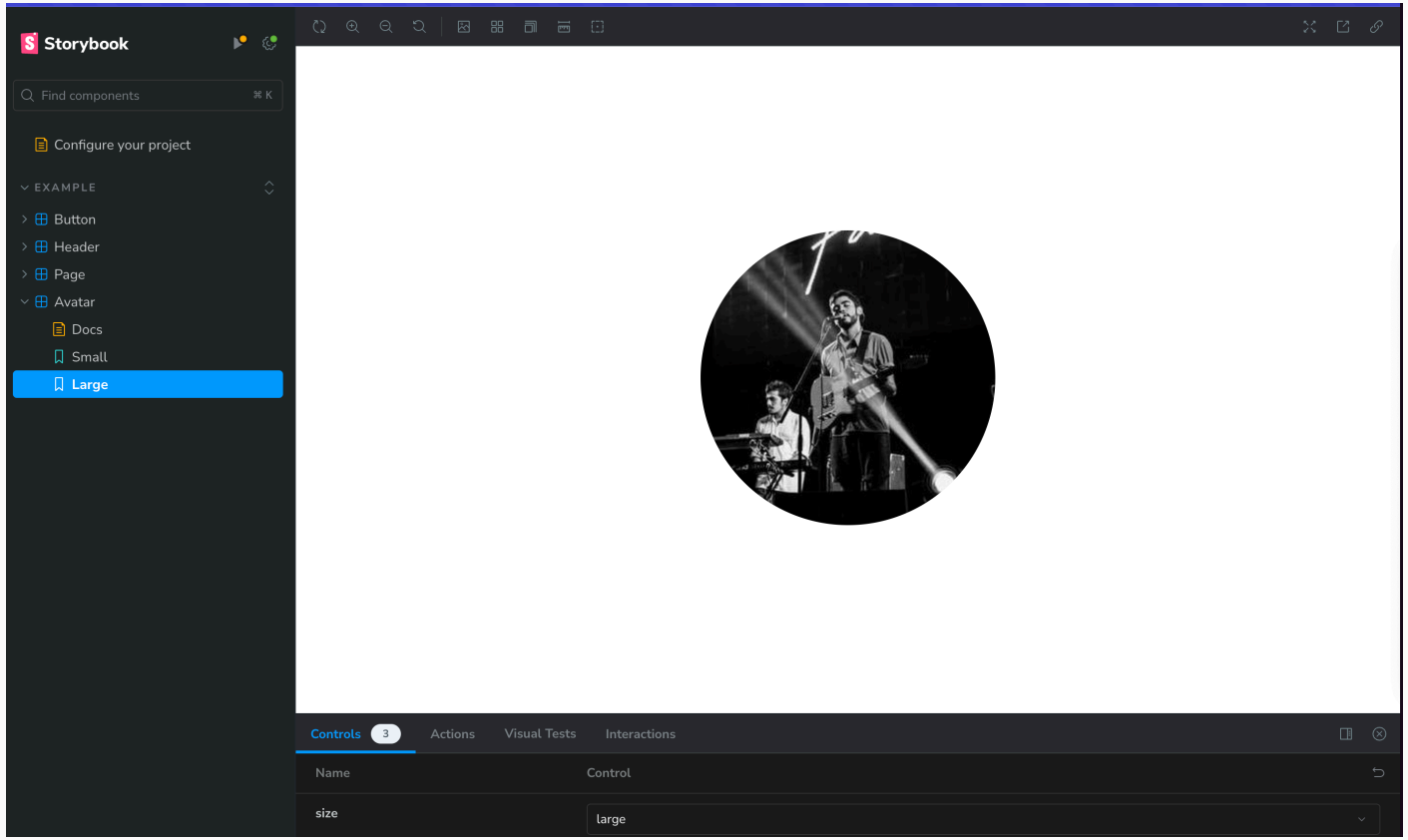
... and create its variants:

```
type Story = StoryObj<typeof meta>;
export const Small: Story = {
  args: {
    size: "small",
    src: "https://picsum.photos/100/100",
    alt: "Small Avatar",
  },
};

export const Large: Story = {
  args: {
    size: "large",
    src: "https://picsum.photos/300/300",
    alt: "Large Avatar",
  },
};
```

Storybook - Your first story

Congratulations! You have created your first story.
Now you can run Storybook and see your Avatar component in action.



React Hooks

React Hooks

Hooks let you use different React features from your components.

A basic **useState** example:

```
import React, { useState } from "react";

const Example: React.FC = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
};
```

Rules of Hooks

Before using hooks, you need be aware of the following rules:

- 🔴 **Only call hooks at the top level**: Don't call hooks inside loops, conditions, or nested functions.
- 🔴 **Dont call hooks after early returns**: Don't call hooks after a conditional return statement.
- 🔴 **Avoid hooks in event handlers**: Don't call hooks in event handlers, or other callbacks.
- 🔴 **Don't call hooks in class components**: Hooks are for functional components only
- 🔴 **Don't call hooks inside functions passed to useMemo, useReducer, or useEffect**

Introducing Hooks

useState: State management

```
import React, { useState } from "react";

const Example: React.FC = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
};
```

Introducing Hooks

useEffect: Side effects - avoid when possible

```
import React, { useState, useEffect } from "react";

const Example: React.FC = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
};
```

Introducing Hooks

useMemo: Memoization - should be used to avoid unnecessary re-renders and expensive calculations

```
import React, { useState, useMemo } from "react";

const Example: React.FC = () => {
  const [count, setCount] = useState(0);
  const memoizedCount = useMemo(() => count * 2, [count]);

  return (
    <div>
      <p>Memoized value: {memoizedCount}</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
};
```

Introducing Hooks

useCallback: Similar to `useMemo`, but for functions

```
import React, { useState, useCallback } from "react";

const Example: React.FC = () => {
  const [count, setCount] = useState(0);
  const increment = useCallback(() => setCount(count + 1), [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={increment}>Click me</button>
    </div>
  );
};
```

Custom Hooks

You can create your own hooks to reuse stateful logic across your application.

```
import { useState, useEffect } from "react";

export const useWindowWidth = () => {
  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => setWidth(window.innerWidth);
    // In a real-world scenario, you would probably want to debounce this
    window.addEventListener("resize", handleResize);
    return () => window.removeEventListener("resize", handleResize);
  }, []);

  return width;
};
```

React Query

React Query

A powerful library for data fetching, caching and state management.

When looking at its documentation, make sure you are looking at the latest version.

Install it with npm:

```
npm install react-query
```

React Query - Basic usage

Querying data

```
import { useQuery } from "react-query";

const fetchUser = async (id: number) => {
  const res = await fetch(`/api/users/${id}`);
  return res.json();
};

const User: React.FC<{ id: number }> = ({ id }) => {
  const { data, isLoading } = useQuery({
    queryKey: ["user", id],
    queryFn: fetchUser,
  });

  if (isLoading) return <div>Loading...</div>;

  return <div>{data.name}</div>;
};
```

React Query - Basic usage

Mutating data

```
import { useMutation } from "react-query";
const updateUser = async (id: number, name: string) => {
  const res = await fetch(...); // truncated
  return res.json();
};

const User: React.FC<{ id: number }> = ({ id }) => {
  const [name, setName] = useState("");
  const { mutate } = useMutation({
    mutationKey: ["user", id],
    mutationFn: updateUser,
  });
  return (
    <div>
      <input value={name} onChange={(e) => setName(e.target.value)} />
      <button onClick={() => mutate(id, name)}>Update</button>
    </div>
  );
};
```

Live coding: It's time to get working.

You can find the assignment in the Interactive Syllabus.

Questions?

Thank you for your attention!