Advanced Web Programming

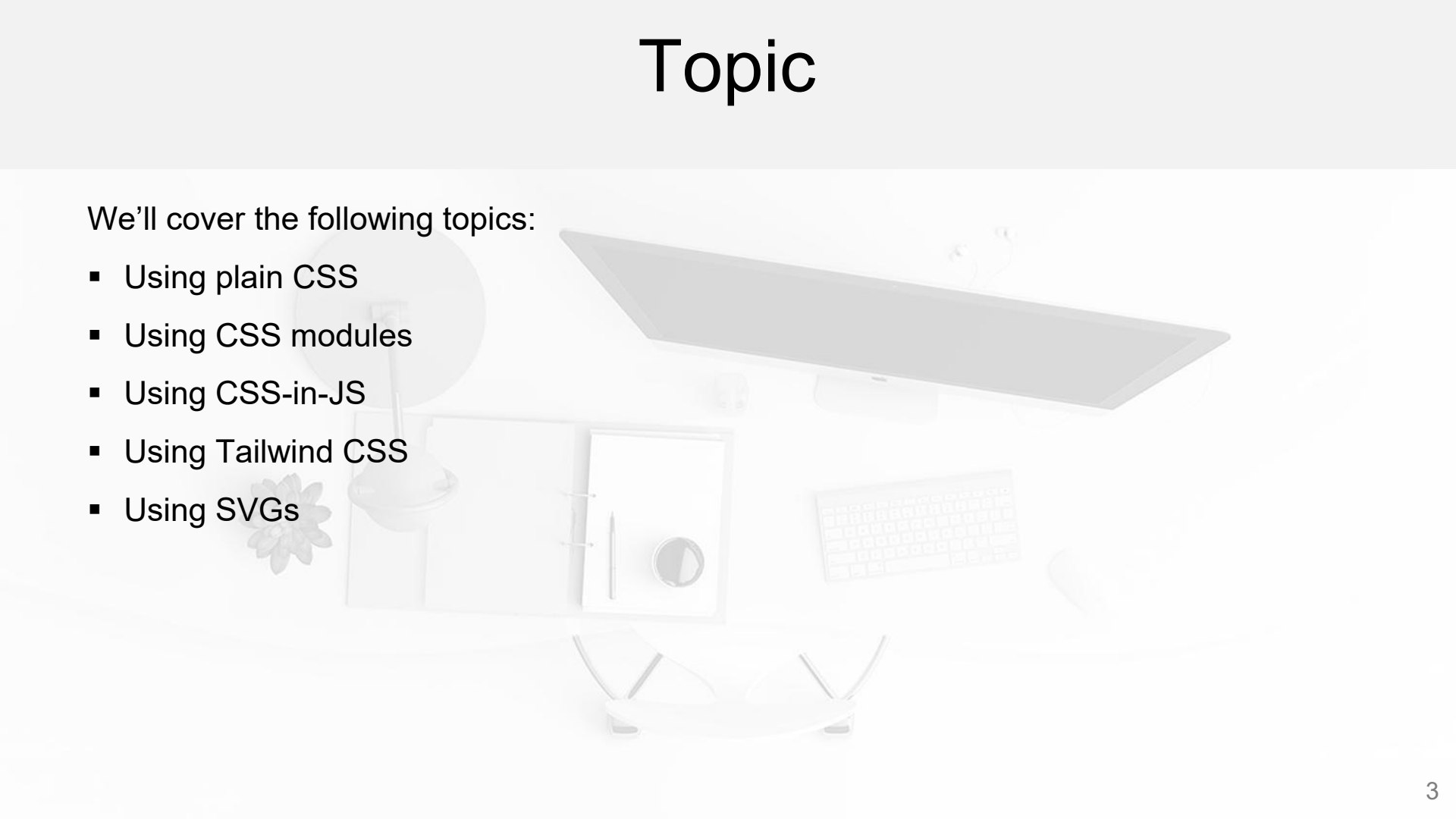
Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

Approaches to Styling React Frontends

Topic

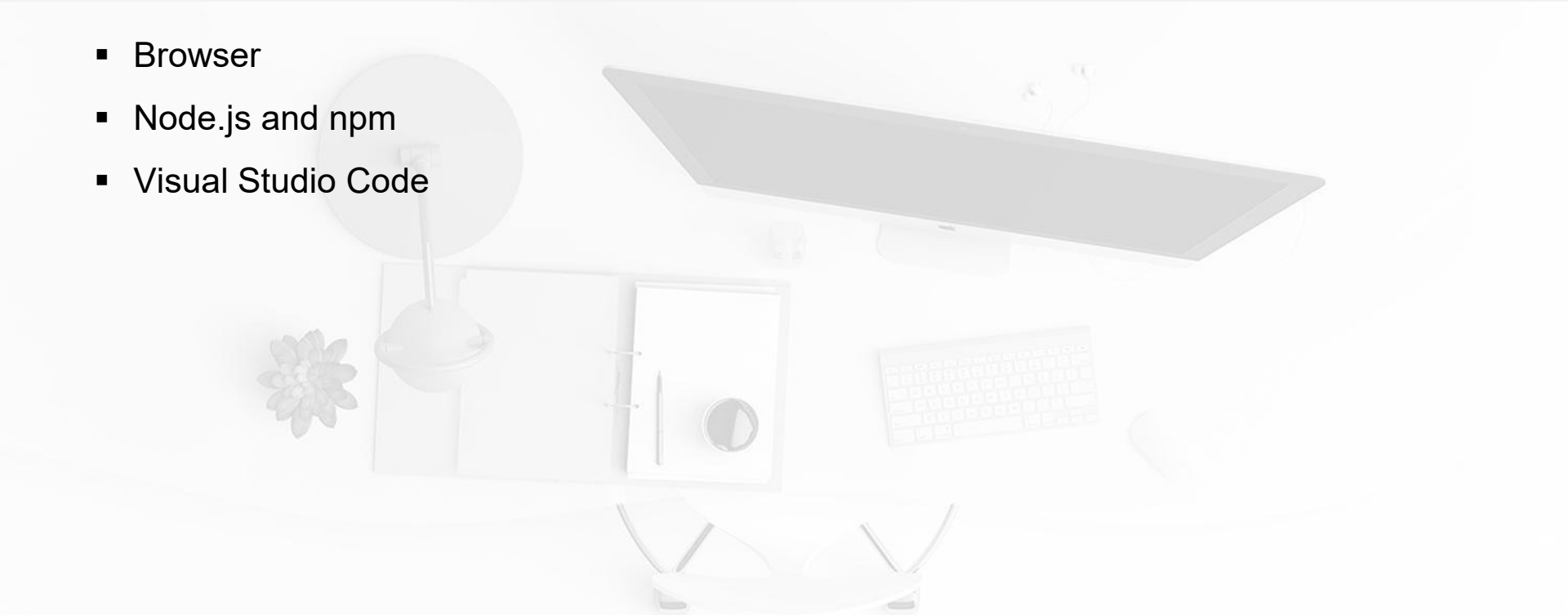
We'll cover the following topics:

- Using plain CSS
- Using CSS modules
- Using CSS-in-JS
- Using Tailwind CSS
- Using SVGs



Technical requirements

- Browser
- Node.js and npm
- Visual Studio Code



1. Using plain CSS

Creating the project

- The project we will be using is the one we completed at the end of Chapter 3.
- Carry out the following steps:
  - Download and Extract the attached project.
  - Open Visual Studio Code in the **myapp** subfolder
  - Run the following command to install all the dependencies: **npm i**

1. Using plain CSS

Understanding how to reference CSS

- Vite has already enabled the use of plain CSS in the project. Look in **App. tsx**, it already uses plain CSS: import './App.css';
- React uses a **className** attribute rather than class because class is a reserved word in JavaScript.
- The CSS import statement is a webpack feature. As webpack processes all the files, it will include all the imported CSS in the bundle.

1. Using plain CSS

Understanding how to reference CSS

- Carry out the following steps to explore the CSS bundle:
  - Leave the redundant CSS classes in **App.css**.
  - Open the **index.tsx** file and you'll notice that **index.css** is imported without CSS classes are referenced with this file.
  - Run the following command in the terminal to produce a production build:

    npm run build

  - After a few seconds, the build artifacts will appear in a build folder at the project's root.

1. Using plain CSS

Understanding how to reference CSS

- Carry out the following steps to explore the CSS bundle:
  - Open **index.html** in the build folder and notice all the whitespace has been removed because it is optimized for production.
  - Open up the referenced CSS file. All the whitespace has been removed because it is optimized for production.

  Notice that it **contains all the CSS** from index.css and App.css, **including the redundant** App-header and App-logo CSS classes.

1. Using plain CSS

Recap

Understanding how to reference CSS

The key point here is that **webpack doesn’t remove any redundant CSS** – it will include all the content from all the CSS files that have been imported.

1. Using plain CSS

Using plain CSS in the alert component

- Carry out the following steps:
  - Add a CSS file called **Alert.css** in the **src** folder.
  - Add the CSS classes

```
.container {
  display: inline-flex;
  flex-direction: column;
  text-align: left;
  padding: 10px 15px;
  border-radius: 4px;
  border: 1px solid transparent;
}
```

```
.container.warning {
  color: #e7650f;
  background-color: #f3e8da;
}
.container.information {
  color: #118da0;
  background-color: #dcf1f3;
}
```

1. Using plain CSS

Using plain CSS in the alert component

- Carry out the following steps:
  - Add the CSS classes

```
.header {
  display: flex;
  align-items: center;
  margin-bottom: 5px;
}
```

```
.header-icon {
  width: 30px;
}
```

```
.header-text {
  font-weight: bold;
}
```

```
.close-button {
    border: none;
    background: transparent;
    margin-left: auto;
    cursor: pointer;
}
```

1. Using plain CSS

Using plain CSS in the alert component

- Carry out the following steps:
  - Add the CSS classes

```
.content {
  margin-left: 30px;
  color: #000;
}
```

  - Open **Alert.tsx** and add an import statement for the CSS file:

```
import './Alert.css';
```

1. Using plain CSS

Using plain CSS in the alert component

- Carry out the following steps:
  - Add the CSS class name references in the **alert** JSX
  - Move the **close button** so that it is located inside the header container, under the header element
  - Start the app

```
<div className={`container ${type}`}>
  <div className="header">
    <span
      role="img"
      aria-label={type === 'warning' ? 'Warning' : 'Information'}
      className="header-icon"
    >
      {type === 'warning' ? '⚠' : 'ℹ'}
    </span>
    <span className="header-text">{heading}</span>
    {closable && (
      <button aria-label="Close" onClick={handleCloseClick} className="close-button">
        <span role="img" aria-label="Close">
          ✖
        </span>
      </button>
    )}
  </div>
  <div className="content">{children}</div>
</div>
```

1. Using plain CSS

Experiencing CSS clashes

Follow these steps:

- Open **App.tsx** and change the referenced CSS class from App to container on the div element.

```
<div className="container">
  <Alert ...>
    ...
  </Alert>
</div>
```

- Open **App.css** and rename the App CSS class to container and also add 20px of padding to it.
- Run the app and give your comments.

```
.container {
  text-align: center;
  padding: 20px;
}
```

1. Using plain CSS

Recap

Experiencing CSS clashes

- Plain CSS classes are scoped to the whole app and not just the file it is imported into. This means that CSS classes can clash if they have the same name.

- A solution to CSS clashes is to carefully name them using **BEM** (Block, Element, Modifier). For example container in the App component could be called App__container, and container in the Alert component could be called Alert__container. However, this requires discipline from all members of a development team.

1. Using plain CSS

Recap

- Vite configures webpack to process CSS so that CSS files can be imported into React component files
- All the styles in an imported CSS file are applied to the app – there is no scoping or removing redundant styles

2. Using CSS modules

Understanding CSS modules

- CSS modules is an open source library available on GitHub at https://github.com/ css-modules/css-modules, which can be added to the webpack process to facilitate the automatic scoping of CSS class names.
- A CSS module is a CSS file that filename has an extension of **.module.css**.
- A CSS module file is imported into a React component file as follows:

import styles from './styles.module.css';

2. Using CSS modules

Understanding CSS modules

- The CSS class name mapping information variable *(styles)* is an object containing property names corresponding to the CSS class names.
- Each class name property contains a value of a scoped class name to be used on a React component.

```
{
  container: "MyComponent_container__M7tzC",
  error: "MyComponent_error__vj8Oj"
}
```

- The **scope CSS class name** starts with the **component filename**, then the **original CSS class name**, followed by **a random string**.

2. Using CSS modules

Understanding CSS modules

- Styles within a CSS module are referenced in a component's **className** attribute:

```
<span className={styles.error}>A bad error</span>
```

- The CSS class name on the element would then resolve to the scoped class name.
- Projects created using Vite already have CSS modules installed and configured with webpack.

2. Using CSS modules

Using CSS modules in the alert component

- Carry out the following steps:
  - Start by renaming **Alert.css** to **Alert.module.css**
  - Open **Alert.module.css** and change the CSS class names to **camel case** rather than kebab case.

```
...
.headerIcon {
    ...
}
.headerText {
    ...
}
.closeButton {
    ...
}
```

2. Using CSS modules

Using CSS modules in the alert component

- Carry out the following steps:
  - Open **Alert.tsx** and change the CSS import statement to import the CSS module

```
import styles from './Alert.module.css';
```

2. Using CSS modules

Using CSS modules in the alert component

- Carry out the following steps:
  - In the JSX, change the class name references to use the scoped names from the CSS module.
  - Start the app.
  - Inspect the elements in the DOM using the browser's DevTools. You will see that the alert component is now using scoped CSS class names.

```
<div className={`${styles.container} ${styles[type]}`}>
  <div className={styles.header}>
    <span
      ...
      className={styles.headerIcon}
    >
      {type === "warning" ? "⚠" : "i"}
    </span>
    {heading && (
      <span className={styles.headerText}>{heading}</
        span>
    )}
    {closable && (
      <button
        ...
        className={styles.closeButton}
      >
        ...
      </button>
    )}
  </div>
  <div className={styles.content}>{children}</div>
</div>
```

2. Using CSS modules

Using CSS modules in the alert component

- Carry out the following steps:
  - Add a redundant CSS class at the bottom of **Alert.module.css**:

```
.redundant {
  color: red;
}
```

  - Create a production build by executing npm run build.
  - Open the bundled CSS file, and give some comments.

2. Using CSS modules

Recap

- CSS modules allow CSS class names to be automatically scoped to a React component. This prevents styles for different React components from clashing.
- CSS modules isn’t a standard browser feature; instead, it is an open source library that can be added to the webpack process.
- CSS modules are pre-installed and pre-configured in projects created with Vite. Similar to plain CSS, redundant CSS classes are not pruned from the production CSS bundle.

3. Using CSS-in-JS

Understanding CSS-in-JS

- **CSS-in-JS is a type of library**. Popular examples of CSS-in-JS libraries are styled-components and Emotion.
- Emotion generates styles that are scoped. However, you write the CSS in JavaScript.

```
<span
  css={css`
    font-weight: 700;
    font-size: 14;
  `}
>
  {text}
</span>
```

3. Using CSS-in-JS

Understanding CSS-in-JS

- Having styles directly on the component allows a developer to fully understand the component without having to visit another file.

→ This obviously **increases the file size**, which can make the code harder to read.

- Child components can be identified and extracted out of the file to mitigate large file sizes. Alternatively, styles can be extracted from component files into a JavaScript function that is imported.

3. Using CSS-in-JS

Understanding CSS-in-JS

- A massive **benefit** of CSS-in-JS is that you can **mix logic into the style**, which is really useful for highly interactive apps.

```
<span
  css={css`
    font-weight: ${important ? 700 : 400};
    font-size: ${mobile ? 15 : 14};
  `}
>
  {text}
</span>
```

3. Using CSS-in-JS

Understanding CSS-in-JS

- JavaScript string interpolation is used to define the conditional statement.

```
<span
  className={`${important ? "text-important" : ""} ${
    mobile ? "text-important" : ""
  }`}
>
  {text}
</span>
```

- If a style on an element is highly conditional, then CSS-in-JS is arguably easier to read and certainly easier to write.

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Vite doesn't install and set up Emotion, so we first need to install Emotion: **npm i @emotion/react**
  - Open **Alert.tsx** and remove the CSS module import.
  - Add an import for the css prop from Emotion with a special comment at the top of the file:

```
/** @jsxImportSource @emotion/react */
import { css } from '@emotion/react';
import { useState } from 'react';
```

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Replace all the className props with the equivalent Emotion css attributes.

```
<div
  css={css`
    display: inline-flex;
    flex-direction: column;
    text-align: left;
    padding: 10px 15px;
    border-radius: 4px;
    border: 1px solid transparent;
    color: ${type === "warning" ? "#e7650f" : "#118da0"};
    background-color: ${type === "warning"
      ? "#f3e8da"
      : "#dcf1f3"};
  `}
>
  ...
</div>
```

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Style the header container:

```
<div
  css={css`
    display: flex;
    align-items: center;
    margin-bottom: 5px;
  `}
>
  <span
    role="img"
    aria-label={type === 'warning' ? 'Warning' : 'Information'}
```

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Style the icon

```
<span
  role="img"
  aria-label={type === "warning" ? "Warning" :
    "Information"}
  css={css`
    width: 30px;
  `}
>
  {type === "warning" ? "⚠" : "i"}
</span>
```

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Style the heading

```
<span
  css={css`
    font-weight: bold;
  `}
>
  {heading}
</span>
```

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Style the close button

```
{closable && (
  <button
    aria-label="Close"
    onClick={handleCloseClick}
    css={css`
      border: none;
      background: transparent;
      margin-left: auto;
      cursor: pointer;
    `}
  >
```

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Style the message container

```
<div
  css={css`
    margin-left: 30px;
    color: #000;
  `}
>
  {children}
</div>
```

3. Using CSS-in-JS

Using Emotion in the alert component

- Carry out the following steps:
  - Run the app and give some comments.
  - Create a production build by executing **npm run build**.
  - Open the bundled CSS file from the build/static/css folder. Notice that the Emotion styles are not there.

    This is because Emotion generates the styles at runtime via JavaScript rather than at build time.

3. Using CSS-in-JS

Recap

- Styles for a CSS-in-JS library are defined in JavaScript.
- Emotion’s styles can be defined directly on a JSX element using a css attribute.
- A huge benefit is that **conditional logic** can be added directly to the styles.
- Emotion styles are **applied at runtime** rather than at build time because they depend on JavaScript variables.

4. Using Tailwind CSS

Understanding Tailwind CSS

- Tailwind is a set of prebuilt CSS classes that can be used to style an app. It is referred to as a **utility-first CSS framework**.
- Exameple CSS classes: bg-white, bg-orange-500.
- Tailwind contains a nice color palette that can be customized.
- The utility classes can be used together to style an element

```
<button className="border-none rounded-md bg-emerald-700 text-white cursor-pointer">
  ...
</button>
```

4. Using Tailwind CSS

Understanding Tailwind CSS

- Tailwind can specify that a class should be applied when the element is in a hover state by prefixing it with hover:

```
<button className="md border-none rounded-md bg-emerald-700
text-white cursor-pointer hover:bg-emerald-800">
  ...
</button>
```

4. Using Tailwind CSS

Recap

Understanding Tailwind CSS

- We **don’t write new CSS classes** for each element we want to style – instead, we use a large range of well-thought-through existing classes.
- A benefit of this approach is that it helps an app look nice and consistent.

4. Using Tailwind CSS

Installing and configuring Tailwind CSS

- Carry out the following steps:
  - Start by installing Tailwind: **npm install tailwindcss @tailwindcss/vite**
  - Configure the Vite plugin:

    Add the @tailwindcss/vite plugin to your Vite configuration.

```
import { defineConfig } from 'vite'
import tailwindcss from '@tailwindcss/vite'
export default defineConfig({
  plugins: [
    tailwindcss(),
  ],
})
```

4. Using Tailwind CSS

Installing and configuring Tailwind CSS

- Carry out the following steps:
  - Import Tailwind CSS

    Add an @import to your CSS file that imports Tailwind CSS.

    @import "tailwindcss";

4. Using Tailwind CSS

Using Tailwind CSS

- Carry out the following steps:
  - Open **Alert.tsx** and start by removing the special emotion comment and the css import statement from the top of the file.
  - Replace the css attribute with a **className** attribute on the outermost div element

```
<div
  className={`inline-flex flex-col text-left px-4 py-3
    rounded-md border-1 border-transparent`}
>
  ...
</div>
```

4. Using Tailwind CSS

Using Tailwind CSS

- Carry out the following steps:
  - Still on the outermost div element, add the following conditional styles using string interpolation:

```
<div
  className={`inline-flex flex-col text-left px-4 py-3
rounded-md border-1 border-transparent ${
    type === 'warning' ? 'text-amber-900' : 'text-
      teal-900'
  } ${type === 'warning' ? 'bg-amber-50' : 'bg-teal-
    50'}`}
>
  ...
</div>
```

4. Using Tailwind CSS

Using Tailwind CSS

- Carry out the following steps:
  - Replace the css attribute with a **className** attribute on the header container

```
<div className="flex items-center mb-1">
  <span role="img" ... > ... </span>
  <span ... >{heading}</span>
  {closable && ...}
</div>
```

4. Using Tailwind CSS

Using Tailwind CSS

- Carry out the following steps:
  - Replace the css attribute with a **className** attribute on the icon

```
<span role="img" ... className="w-7">
  {type === 'warning' ? '⚠' : 'i'}
```

  - Replace the css attribute with a **className** attribute on the heading

```
<span className="font-bold">{heading}</span>
```

4. Using Tailwind CSS

Using Tailwind CSS

- Carry out the following steps:
  - Replace the css attribute with a **className** attribute on the close button

```
{closable && (
  <button
    ...
    className="border-none bg-transparent ml-auto cursor-
      pointer"
  >
    ...
  </button>
)}
```

4. Using Tailwind CSS

Using Tailwind CSS

- Carry out the following steps:
  - Replace the css attribute with a **className** attribute on the message container

```
<div className="ml-7 text-black">
  {children}
</div>
```

  - Run the app

4. Using Tailwind CSS

Recap

Using Tailwind CSS

- No CSS class name scoping occurs.
- There is no need for any scoping because the classes are general and reusable and not specific to any element.

4. Using Tailwind CSS

Using Tailwind CSS

- Carry out the following steps:
  - Create a production build by executing npm run build
  - Open the bundled CSS file from the build/static/css folder. See and give some comments.
- **Note**:

  Tailwind **doesn’t add all its CSS classes** – that would produce amassive CSS file!

  Instead, it only adds the CSS classes used in the app.

4. Using Tailwind CSS

Recap

- Tailwind is a well-thought-through collection of reusable CSS classes that can be applied to React elements
- Tailwind has a nice default color palette and a 4px spacing scale, both of which can be customized
- Tailwind is a plugin for PostCSS and executed at build time
- Tailwind does not incur a runtime performance penalty like Emotion
- Only classes used on React elements are included in the CSS build bundle

5. Using SVGs

Understanding how to use SVGs in React

- SVG (Scalable Vector Graphics) is made up of points, lines, curves, and shapes based on mathematical formulas rather than specific pixels. This allows them to scale when resized without distortion.
- Vite configures webpack to use SVG files in the App component when a project is created.

```
import logo from './logo.svg';
...
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
```

5. Using SVGs

Understanding how to use SVGs in React

- An alternate way of referencing SVGs is to reference them as a component

```
import { ReactComponent as Logo } from './logo.svg';

function SomeComponent() {
  return (
    <div>
      <Logo />
```

5. Using SVGs

Adding SVGs to the alert component

- Carry out the following steps:
  - Create three files called cross.svg, info.svg, and warning.svg in the src folder.
  - Open **Alert.tsx** and add the following import statements to import the SVGs

```
import { ReactComponent as CrossIcon } from './cross.svg';
import { ReactComponent as InfoIcon } from './info.svg';
import { ReactComponent as WarningIcon } from './warning.svg';
```

5. Using SVGs

Adding SVGs to the alert component

- Carry out the following steps:
  - Update the span element containing the emoji icons to use SVG icon components

```
{type === 'warning ' ? (
  <WarningIcon className="fill-amber-900 w-5 h-5" />
) : (
  <InfoIcon className="fill-teal-900 w-5 h-5" />
)}
```

5. Using SVGs

Adding SVGs to the alert component

- Carry out the following steps:
  - Update the emoji close icon to the SVG close icon

```
<button
  aria-label="Close"
  onClick={handleCloseClick}
  className="border-none bg-transparent ml-auto cursor-
    pointer"
>
  <CrossIcon />
</button>
```

  - Run the app

5. Using SVGs

Recap

- Vite can help us configure Webpack to bundle SVG files
- The default import for an SVG file is the path to the SVG, which can then be used in an img element
- A named import called ReactComponent can be used to reference the SVG as a React component in JSX

Summary

- **Plain CSS** could be used to style React apps, but all the styles in the imported CSS file are bundled regardless of whether a style is used. Also, the styles are **not scoped** to a specific component
- **CSS modules**, which is an open source library pre-installed and pre-configured in projects created with Vite, allows us to write plain CSS files imported in a way that scopes styles to the component. It **resolved the CSS clashing problem** but didn't remove redundant styles.

Summary

- **CSS-in-JS libraries**, which allow styles to be defined directly on the React component. In this approach, **conditional-style logic** can be implemented more quickly. The **small performance cost** of this approach is because of the styles being created at runtime.
- **Tailwind** provides **a set of reusable CSS classes** that can be applied to React elements. Tailwind classes are included in the production build.
- **Vite configures webpack to enable the use of SVG files**. SVGs can be referenced as a path in an img element or as a React component using a ReactComponent named import.


Q&A