11 SATURDAY
Week 41 Day (284-081)

OCTOBER

OCT 25	M	T	W	T	F	S	S	M	T	W	T	F	S	S
			01	02	03	04	05	06	07	08	09	10	11	12
	13	14	15	16	17	18	19	20	21	22	23	24	25	26
	27	28	29	30	31									

Appointments

09 Suspense is used to manage asynchronous data fetching and code-splitting allowing you to display loading indicator

10 while the data load or code ~~or~~ ~~is~~ being fetched. When suspense is encountered by react it knows there might be

11 a delay in rendering & it can handle that situation gracefully.

13 # Episode-10 - Styling our app

14

(1) First way to write CSS

15 The first way to create an index.css add css a/c to class name.

16

(2) Second way to write saas & scss

17 saas (Syntactically Awesome Style Sheet).

saas is a CSS ~~pre~~ processor meaning it extends

12 SUNDAY CSS with ~~a~~ extra features like: variables, nesting, mixins, functions and inheritance.

Browsers do not understand saas it must compiled it to CSS.

It is compiled using saas compiler and you can use build tools like vite, gulp, parcel, etc. to let them handle compilation or you can run this command

saas input.saas output.css

2025 saas input.scss output.css

Appointments

SCSS (Sassy CSS) is one of the syntaxes of Saas

Saas has two syntaxes

1. Saas (original, indented syntax)
2. SCSS (CSS-like syntax - most popular)

Sass VS SCSS (writing-style)

1. Saas : No {} braces, No semicolons (;), use indentation

```
$primary-color: blue
body
    background: #f5f5f5
h1
    color: $primary-color
```

2. SCSS : uses braces {} & semicolons (;)
Very similar to CSS, E

```
$primary-color: blue
body {
    background: #f5f5f5;
    }
h1 {
    color: $primary-color;
    }
```

Both are need to compiled into CSS before sending to browser.

09 (3) Recommened way (Lot of big companies used it) :- Styled-components

It is a popular CSS-JS library used mainly with React & React Native that lets you write CSS inside your JS files, scoped to individual components. It removes unused CSS automatically & can styled components dynamically using props.

10 Traditional CSS

```
.button {
  background: blue;
  color: white;
}
```

```
<Button className="button">Click Me</Button>
```

Styled CSS

```
import styled from 'styled-components';
const Button = styled.button`
  background: blue;
  color: white;
`;
```

```
<Button>Click Me</Button>
```

=) Dynamic Styling with Props

```
const Button = styled.button`
  background: ${props => (props.primary ? "blue" : "gray")};
  color: white;
`;
```

<Button primary> Primary </Button> (blue)
<Button> Secondary </Button> (gray)

(4) material UI, chakra UI, bootstrap : CSS libraries and frameworks.
They give us pre-styled components. We just import it into our app and use it

=) Extending Styles

```
const Button = styled.button`
  padding: 10px;
  border-radius: 5px;
`;
```

Appointments

```
const DangerButton = styled(Button)`
  background: red;
```

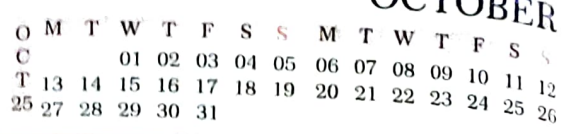
=) Styling based on theme

```
import { ThemeProvider } from "styled-components";
const themes = {
  colors: {
    primary: "#1ADFF2"
  }
};
const Button = styled.button`
  background: ${props => props.theme.colors.primary};
`;
<ThemeProvider theme={theme}>
  <Button>Theme Button</Button>
</ThemeProvider>
```

How to use in Real-life

One Base button + variants using props;

```
export const Button = styled.button`
  padding: 10px;
  border-radius: 6px;
  border: none;
  background: ${({theme, variant}) => {
  background: ${({variant, theme}) =>
    theme.button[variant].bg};
```



Appointments

```
color: ${({ variant, theme }) =>
    theme.button[variant].color};
`;
```

Theme (Centralized styles)

```
const theme = {
  button: {
    primary: {
      bg: "#1DA1F2",
      color: "white",
    },
    secondary: {
      bg: "blue",
      color: "white",
    }
  }
}
```

```
<Button variant="primary">Save</Button>
```

Using css helper (Advanced & clean)

```
import styled, {css} from 'styled-components';
const variants = {
  primary: css`
    background: "blue";
    color: "white;
  `,
  secondary: css`
    background: "gray";
    color: "white";
  `
```

Appointments

```
export const Button = styled.button`
    padding: 10px 16px;
    border-radius: 5px;
    ${({variants}) => variants[variants]};
`;

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

(4) material UI, chakra UI, bootstrap : CSS libraries
They give pre-styled components & we just import it & we use it.

(5) Tailwind CSS
We will install tailwind css package using
npm i -D tailwindcss postcss
where postcss is used to ~~comm~~ transforming CSS with Javascript.
We will create .postcssrc and parcel will read this file and it will tell parcel that we are using tailwind.
Tailwind CSS give us className to use any css in our app
If we want to give specific value in width or others we can use '[ ]'
w-[200px]

Appointments Pro and Cons of Tailwind CSS

09

Pro

10 We don't have to move between files.
• It is very lightweight as parcel will
11 bundle (import only css) only css which is
required. If we use m-4, p-4 or otherwise many
12 times, it will only include that many times
it is dynamic

Cons

13 If we have to write a lot of css for
something we have to add more & more
14 things in className which makes our code
looks ugly & verbose.

15

16 Media query in Tailwind
uses like sm, md, lg, xl

17

<div className = "bg-pink-100 sm: bg-yellow-
19 SUNDAY Header

If my device size is greater than sm
then show yellow Header else show pink

Dark mode in Tailwind

<div className = "text-slate-500 dark: text-slate-400
Header

2025 So in dark mode it has text color has
text-slate-400 otherwise text-slate-500.

Appointments

CSS can be added in 3 ways in HTML doc

09

① Inline :- by using style attribute inside HTML element

```
<h1 style="color:blue">A Blue Heading</h1>
```

10

11

② Internal :- by using <style> element in the section

12

```
<html>
  <head>
    <style>
      body {background-color: blue;}
      h1 {color: blue;}
      p {color: red;}
    </style>
  </head>
```

13

14

15

16

③ External :- by using a <link> element to link to external CSS file

```
<link rel="stylesheet" href="styles.css">
```

Q In tailwind.config.js what does all the keys mean (content, theme, extend, plugins)?

1. content key : specifies the files that tailwind CSS should analyze to generate its utility classes.

```
module.exports = {
  content: [ './src/**/*.html',
             './src/**/*.js',
             // Add more file paths as needed
  ],
}
```

Appointments

09 (2) theme key : Defines the default styles and configuration for various aspect of Tailwind CSS, such as colors, spacing, fonts & more.

10

11 (3) extend key : Extends or overrides the default configuration provided by Tailwind CSS.

12

```
13 module.exports = {
       extend: {
14       colors: {
             primary: '#3490dc',
15           secondary: '#ffed4a',
             // Add more
16         }
         // Other extensions
17     },
       // Other config
   }
```

(4) plugins key Allows you to use or define custom plugins to extend or modify Tailwind CSS functionality.

```
module.exports = {
    plugins: [
        require('@tailwindcss/forms'),
    ],
}
```

Appointments

Why do we have .postcssrc file?

Reasons

- **Plugin config:** The main purpose of this file is to configure the plugins that PostCSS should use during the CSS transformation process. These plugins can handle tasks such as autoprefixing, minification and syntax enhancements.
- **Custom config:-** You may need a .postcssrc file if you want to customize the behaviour of PostCSS beyond the default settings provided by build tool (~~webpack~~ webpack) - This allows you to have fine-grained control over the PostCSS transformations.
- **Presets and Options:** PostCSS plugins often come with various options and presets that you can configure based on your project's needs. The .postcssrc file is a convenient place to define these presets and options.
- **Maintainability:** Separating the PostCSS configuration into its own file makes the build configuration more maintainable and organized. It allows you to centralize PostCSS-related settings and keep them distinct from other build tool config.
- **Sharing config:** Having a dedicated config file makes it easier to share & reuse PostCSS config across different ~~chats~~ projects. It can be particularly useful in larger development ecosystem where consistent styles and build processes are desired.

NOVEMBER