Welcome - Session 5 Overview

- **Styling methods in React (Web)**
- **Styling in React Native**
- **CSS Modules, CSS-in-JS, and utility-first CSS**
- **Styling best practices**
- **Responsive and adaptive UI**

Styling Approaches in React (Web)

Plain CSS files (global)

CSS Modules (scoped to components)

Inline styles (style={{}})

CSS-in-JS (styled-components, Emotion)

Utility CSS (TailwindCSS)

Plain CSS and CSS Modules

```
/* App.css */
.title { color: blue; }

// App.jsx
import './App.css';
<h1 className="title">Hello</h1>

CSS Modules:
import styles from './App.module.css';
<h1 className={styles.title}>Hello</h1>
```

Helps avoid class name collisions

Inline Styles & Conditional Styling

```
const style = {
  padding: '1rem',
  backgroundColor: isActive ? 'green' : 'gray'
};

<div style={style}>Click Me</div>
```

- Useful for dynamic values
- Not recommended for large-scale styling

CSS-in-JS with Styled-Components

```
import styled from 'styled-components';

const Button = styled.button`
  background: blue;
  color: white;
  padding: 0.5rem 1rem;
`;

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

CSS with Tailwind

```
<button className="bg-blue-500 text-white px-4 py-2 rounded">
  Click Me
</button>
```

- Fast styling with pre-defined classes
- No need to write custom CSS
- Good for consistent spacing, colors, and layout

Responsive Design with Media Queries

```
@media (max-width: 768px) {
  .container { flex-direction: column; }
}
```

- Use em, %, or rem units for flexibility
- Tailwind: md:flex-col, sm:text-center, etc.

Styling in React Native

- Use the StyleSheet API
- Styles are JS objects
- No CSS files

```
const styles = StyleSheet.create({
  container: { padding: 20, backgroundColor: 'white'},
  title: { fontSize: 24, color: 'black' }
});

<View style={styles.container}>
  <Text style={styles.title}>Hello</Text>
</View>
```

Conditional & Reusable Styles (React Native)

```
<Text style={[styles.label, isError && styles.error]}>Message</Text>
```

- Use arrays to combine styles
- Add conditional styles based on props or state

Custom Themes and Global Styles

- Web:
  - Use context or CSS variables for theming
  - Example: Light/Dark mode toggle

- Native:
  - Use context or a UI library like NativeBase or React Native Paper

Best Practices for Styling

- Keep styles close to components (e.g., CSS Modules, Styled Components)
- Be consistent with naming and spacing
- Avoid deep nesting in styles
- Abstract repetitive styles into shared files
- Prefer responsive/adaptive design

Styling Libraries and Frameworks

- Web:
  - TailwindCSS
  - Styled Components
  - Emotion
  - Bootstrap
- Native:
  - NativeBase
  - React Native Paper
  - Tamagui (cross-platform)

Recap & Q&A

- Multiple ways to style React and React Native apps
- Choose method based on scale, team, and maintainability
- Emphasize reusability, responsiveness, and simplicity