### 1. Understanding the Need for Styling React Components

**Objective:**

To provide a clean, user-friendly interface and enhance the visual structure of the UI, styling is an essential part of any frontend application. In React, component-based architecture allows us to encapsulate both logic and presentation, making styling a crucial concern at the component level.

**Key Reasons for Styling:**

- **Improved User Experience**: Well-styled components guide user interaction and make the interface intuitive.
- **Component Reusability**: Encapsulated styles help make components modular and reusable across the application.
- **Maintainability**: Organized styles reduce the complexity of managing styles in large applications.
- **Brand Consistency**: Applying consistent styling across components ensures branding and design coherence.

**Common Styling Challenges in React:**

- Global CSS can lead to **class name conflicts** across components.
- Managing **large-scale UI styles** can become disorganized without modularization.
- Need for **dynamic styling** based on state or props.

---

### 2. Working with CSS Modules in React

**Objective:**

CSS Modules provide a mechanism to apply scoped and modular styles to React components, avoiding the drawbacks of global CSS.

**What is a CSS Module?**

A CSS Module is a .css file in which all class and animation names are scoped locally by default. This ensures styles do not clash with other components.

**How to Use CSS Modules:**

1. **Create a CSS Module file** with the naming convention:

**ComponentName.module.css**

2. **Import the styles** into your component:

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

3. **Apply styles** using JSX syntax:

** Content **

**Example:**

**CohortDetails.module.css**

```
.box {
  width: 300px;
  margin: 10px;
  padding: 10px 20px;
  border: 1px solid black;
  border-radius: 10px;
}
```

**CohortDetails.js**

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

function CohortDetails() {
  return <div className={styles.box}>Cohort Details</div>;
}
```

**Benefits:**

- **Locally scoped styles**: Prevents unintended overrides
- **Modular design**: Easier to maintain and refactor
- **Supports all CSS features**: Media queries, pseudo-classes (:hover, :focus, etc.)

---

**3. Working with Inline Styles in React**

**Objective:**

To apply styling directly within the component using JavaScript objects, particularly useful for **dynamic or conditional styling**.

**What are Inline Styles?**

Inline styles in React are defined as plain JavaScript objects where CSS properties are written in camelCase format and applied directly to elements using the style attribute.

**Syntax:**

```
const styleObject = {
  color: 'green',
  fontSize: '18px'
};

<h1 style={styleObject}>Title</h1>
```

**Dynamic Example:**

```
const statusStyle = {
  color: status === 'ongoing' ? 'green' : 'blue'
};

<h3 style={statusStyle}>Status: {status}</h3>
```

**Benefits:**

- Best suited for **conditionally styled elements**
- Quick and easy to implement for **one-off styles**
- Useful for toggling styles based on **state or props**

**Limitations:**

- Cannot use pseudo-selectors (:hover, :focus)
- No support for media queries
- Less reusable for large or complex styling rules