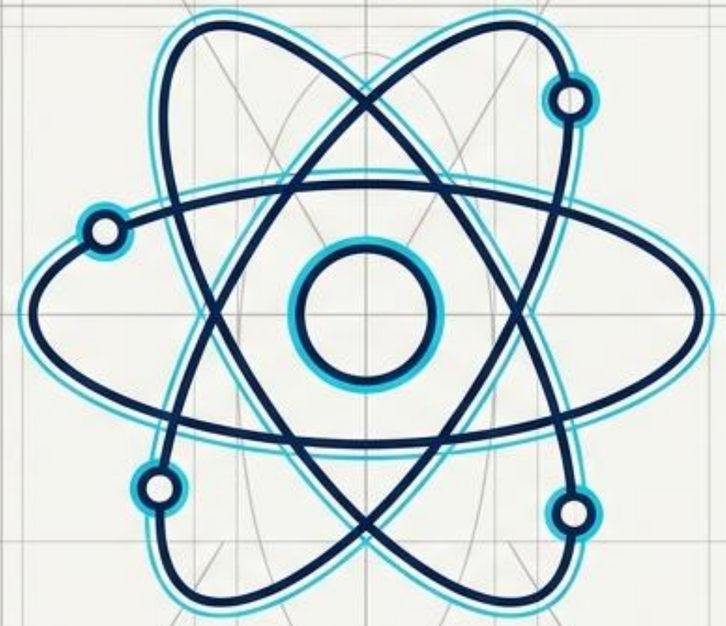
@7scribes contact@7scribes.com 03418949391



Mastering React Components

Building Modular User Interfaces.

Transform complexity into clarity using self-contained UI blocks.

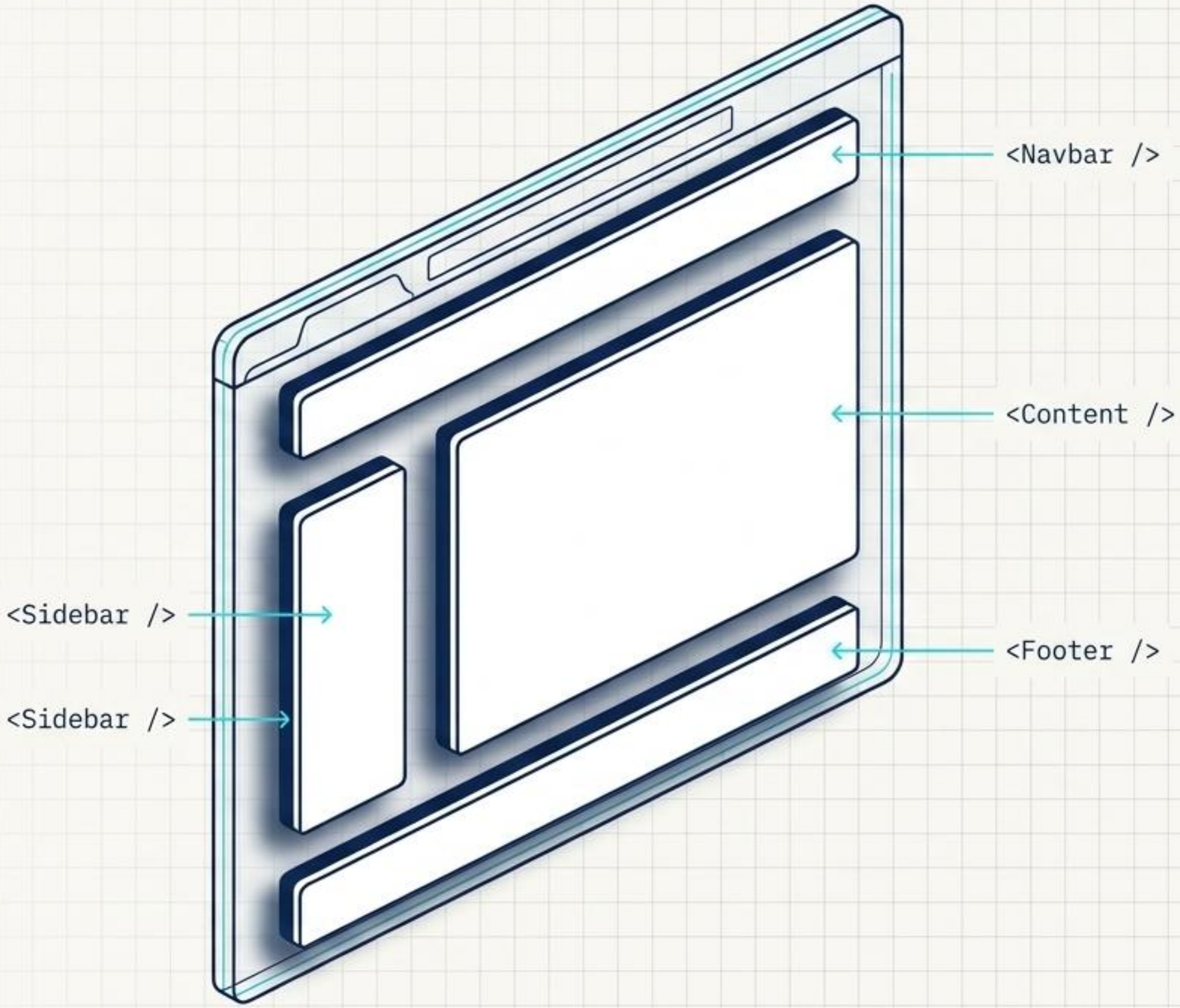
A Carefully Composed System

Components are not merely pieces of code; they are self-contained ideas.

- Every visual element is shaped through small, meaningful units.
- Each unit is responsible for a distinct part of the interface.
- Together, they create structured, living applications.

Visualizing the Modular Webpage

Imagine a webpage not as a single entity, but as a collection of smaller elements working together.



The Anatomy of a Component

```
function Navbar() { return <h2>This is the Navbar</h2>; }
function Footer() { return <p>This is the Footer</p>; }

function App() {
  return (
    <div>
      <Navbar />
      <Footer />
    </div>
  );
}
```

Define Independence.
Defines how a specific part of the screen should appear independently.

Compose Together.
Brings the independent components together to form a complete interface.

Two Paradigms: The Evolution of Components

Functional Components

Elegant, simple JavaScript functions. The modern standard for React development.

Class Components

ES6 classes requiring a dedicated render method. The structured, legacy approach.

Functional Components: Dynamic by Design

Written as simple JavaScript functions, they can accept inputs—called **props**—to adapt dynamically based on data.

Class Components: The Structured Approach

An earlier style of writing React applications, requiring ES6 classes and a dedicated method to return the user interface.

```
class Greeting extends React.Component {
  render() {
    return <h2>Hello, {this.props.name}!</h2>;
  }
}
```

Note the requirement of the render() method and the use of this.props to access dynamic inputs.

The Component Paradigm Matrix

Aspect	Functional Components	Class Components
Syntax	Simple JavaScript function	ES6 class
Complexity	Easy to understand	More complex
Code Length	Short and concise	More verbose
Readability	High	Moderate
Usage Trend	Modern and widely used	Older approach

Bringing Order to Complexity

Why manage a massive block of code when you can write it once and use it everywhere?

The Definition

```
function Button() {
  return <button>Click Me</button>;
}
```

The Implementation

```
function App() {
  return (
    <div>
      <Button />
      <Button />
      <Button />
    </div>
  );
}
```

Reusing a single component reduces repetition, effort, and allows each part of the interface to be understood in isolation.

The Component Lifecycle

Every component follows a journey from creation to removal.

```
componentDidMount() {
  console.log('Mounted');
}
```

1. Mounting

The component appears.

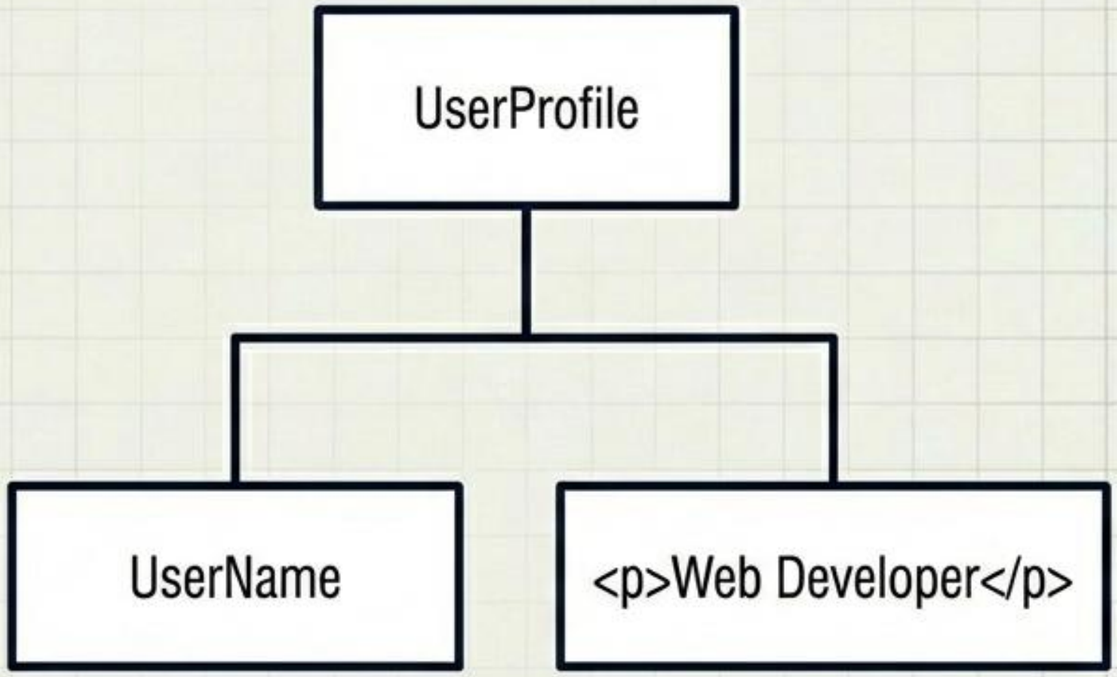
2. Updating

Data changes. The UI reacts.

3. Unmounting

The component is removed.

Best Practices: Intentional Composition




```
function UserName() {
  return <h3>Rahat Hussain</h3>;
}

function UserProfile() {
  return (
    <div>
      <UserName />
      <p>Web Developer</p>
    </div>
  );
}
```

- ✓ Divide logic into smaller, manageable parts.
- ✓ Use clear and meaningful names.
- ✓ Reuse components wherever possible.
- ✓ Keep logic separate from UI.

The Advantages of Scale

Easy Debugging

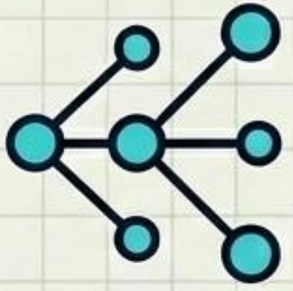
Trace issues directly to a single isolated component (e.g., `Content`) without affecting the rest of the application (e.g., `Header`).

Code Reusability

Write once, deploy endlessly across the application footprint.

High Performance

Improve overall application speed through highly efficient, localized updates.



Scalable Design

Support massive, infinitely scalable application architectures effortlessly.

Core Concepts Refresher

What is a component?

A reusable, independent piece of UI that defines how a specific part of the screen should appear to build web applications.

Functional vs. Class?

Functional components are simple, modern JavaScript functions. Class components use ES6 classes and follow an older, more verbose approach.

Why are they important?

They bring order to complexity, making code reusable, highly organized, and significantly easier to maintain.

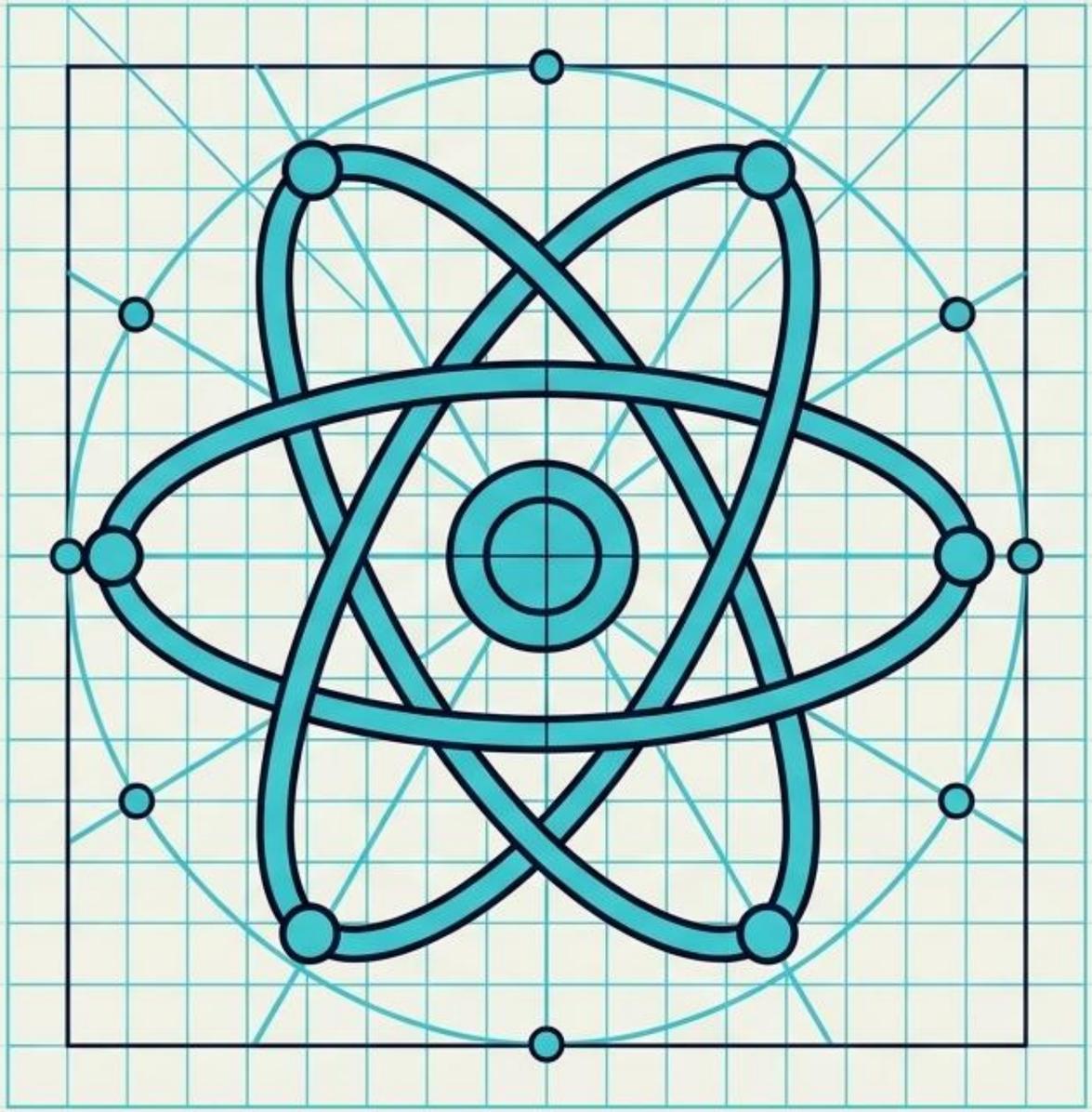
Can we reuse them?

Yes! Components are fundamentally designed to be written once and reused multiple times across an application.

From Complexity to Clarity

7Scribes

React components form the foundation of every React application. By starting with functional components and practicing consistent modularity, the art of building user interfaces becomes deeply intuitive.



Stop writing code.
Start building systems.

@7scribes contact@7scribes.com 03418949391