🧩 React Component

→ React এ Component হলো একটা ছোট, reusable UI অংশ যেটা পুরো Application বানানোর জন্য ব্যবহৃত হয়।

🧩 Component Creation in React

১. Class Component (ক্লাস কম্পোনেন্ট) 🧱

```
import React, { Component } from 'react';
      //React.Component
          👇
class ComponentName extends Component {
 // render() method (আবশ্যক)
 render() {
   return (
    // JSX Markup
    <><h1>Hello World</h1></>
   );
 }
}
export default ComponentName;
```

- Class এক্সটেনশন: ক্লাসটিকে অবশ্যই `React.Component` ক্লাস থেকে extend করতে হয়, যাতে এটি React-এর Component-এর সব বৈশিষ্ট্য (যেমন lifecycle methods, `setState`) পেতে পারে।
- `render()` মেথড: একটি Class Component-এ `render()` মেথড থাকা আবশ্যক। React এই মেথডটিকে কল করে এবং এই মেথডের রিটার্ন করা JSX থেকে UI তৈরি হয়।

২. **Functional Component** (ফাংশনাল কম্পোনেন্ট) 🚀

```
import React from 'react';

const ComponentName = () => {
  return (
    // JSX Markup
    <><h1>Hello World</h1></>
  );
};

export default ComponentName;


// অথবা

function ComponentName() {
  return (
    // JSX Markup
     <><h1>Hello World</h1></>
  );
}

export default ComponentName;
```

- জাভাস্ক্রিপ্ট ফাংশন: এটি তৈরি হয় একটি রেগুলার বা Arrow JavaScript Function হিসেবে।
- JSX রিটার্ন: ফাংশনটি সরাসরি JSX রিটার্ন করে, যা React ব্যবহার করে স্ক্রিনে UI হিসেবে রেন্ডার করে।

❓ কিছু কিছু প্রশ্ন:

🤔 ক) কেন **React** এ **Component** ব্যবহার করবো?

→ Component code কে ছোট ছোট অংশে ভাগ করে, যাতে code reusable, readable, maintainable হয়।

❌ খ) **Component** ব্যবহার না করলে কী হবে?

→ সব code এক file এ লিখতে হবে → code অনেক বড় হবে → বুঝতে কষ্ট হবে → bug fixed বা update করতে সময় লাগবে।

🚀 গ) কেন **Functional component** বেশি জনপ্রিয়? এবং **Class Component** কেন ব্যবহার করা হয় না?

→ Functional Component ছোট, সহজে, readable এবং hooks থাকার কারণে সব কাজে এখন করতে পারে। কিন্তু Class component লম্বা code, `this` binding এবং কাঠামো আর কম readable বলে খুব কম ব্যবহার করা হয়।

🔄 ঘ) **Component re-render** মানে কী এবং কেন **re-render** হয়?

→ যখন Component এ কোনো State পরিবর্তন হয়, তখন সেই Component এবং তার Child Component গুলি re-render হয়। React Component re-render হওয়ার পরে virtual DOM এর মাধ্যমে DOM এ update করে। → কেন re-render হয়? i) State change হলে। ii) Props change হলে। iii) App-এর লজিক দ্বারা (Scroll বা click এ পরিবর্তন হলে)। iv) FPS কমলে। v) Rendering এ বাধা পেলে। vi) CPU use বেড়ে গেলে। vii) Batch data update করার কারণে।

⚡ ঙ) **Component re-render** হলে কীভাবে **optimize** করা যায়?

→ Re-render optimize করা দরকার কারণ re-render বেশি হলে application slow, user experience খারাপ হবে এবং performance কমবে। → Component re-render optimize করলে app fast, smooth এবং responsive হয়। → Re-render optimize করার জন্য `React.memo()`, `useCallback()`, `useMemo()` ব্যবহার করতে হয়।

⚠️ চ) **Error Boundary** কী এবং কেন ব্যবহার করা হয়?

→ Error Boundary হলো এক ধরনের React Component.
→ Error Boundary ব্যবহার করার প্রধান কারণ হলো এই পরিস্থিতি সামাল দেওয়া।

- পুরো UI ক্র্যাশ হওয়া থেকে রক্ষা: এটি নিশ্চিত করে যে একটি অংশের Error-এর জন্য পুরো অ্যাপ্লিকেশন বন্ধ না হয়ে যায়। এটি ত্রুটিযুক্ত অংশটিকে আলাদা করে সেখানে একটি বন্ধুত্বপূর্ণ ত্রুটি বার্তা (Fallback UI) দেখায়।
- ভালো ব্যবহারকারীর অভিজ্ঞতা (UX): ব্যবহারকারীরা একটি খালি স্ক্রিনের পরিবর্তে একটি ত্রুটি বার্তা বা লোডিং স্ক্রিন দেখতে পায়, যা তাদের বুঝতে সাহায্য করে যে কিছু ভুল হয়েছে।