import React, { useState } from 'react';

const messages = [

  'Learn React ⚛️',

  'Apply for jobs 💼',

  'Invest your new income 🤑',

];

const App = () =>

{

  const [step, setStep] = useState(1);

  const [isOpen, setIsOpen] = useState(true);

  const handlePrevious = ()=>

  {

    if(step > 1)

    {

      setStep((step)=> step-1);

    }

  }

  const handleNext = ()=>

  {

    if(step < 3)

    {

      setStep((step)=> step+1);

    }

  }

  return (

    <div>

       <button className='close' onClick={()=> setIsOpen((isOpen)=> !isOpen)}>&times;</button>

  { isOpen && (  <div className="steps">

    <div className="numbers">

      <div className={`${ step >= 1 ? "active" : "" }`}>1</div>

      <div className={`${ step >= 2 ? "active" : "" }`}>2</div>

      <div className={`${ step >= 3 ? "active" : "" }`}>3</div>

    </div>

    <p className="message"> Step {step}: {messages[step-1]}</p>

    <div className="buttons">

      <button className="btn" onClick={handlePrevious}>Previous</button>

      <button className="btn" onClick={handleNext}>Next</button>

    </div>

  </div>)}

    </div>

  );

}

export default App;

**Introduction to State**

State is crucial in React for making components interactive. It is data that a component can hold and remember over time.

**Characteristics of State**

* Can be updated by user actions.
* Helps the component remember information throughout its lifecycle.
* State is preserved through re-renders unless a component unmounts.

**Re-rendering**

* Re-rendering means React calls the component function again, conceptually removing and replacing the view with a new one each time.
* State is preserved throughout re-renders unless the component unmounts.

**Importance of State**

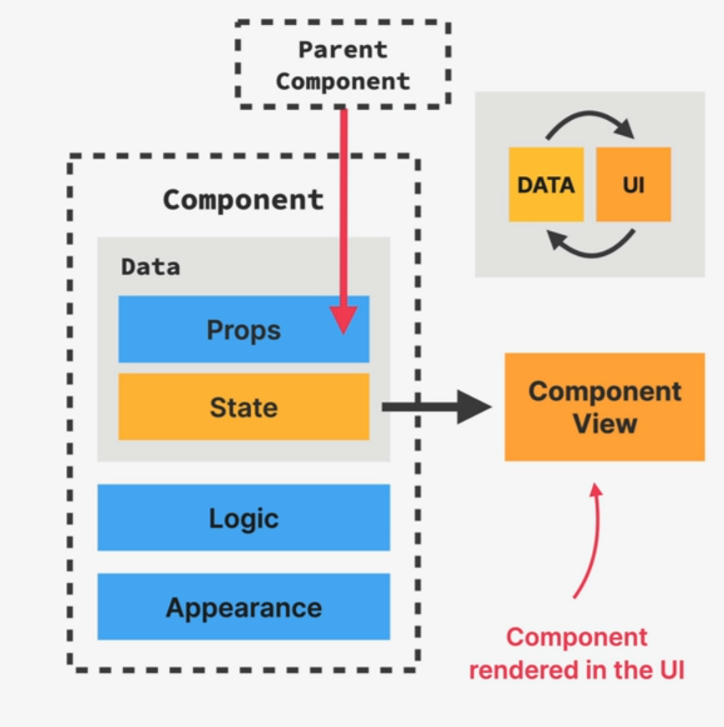
* Updating state triggers React to re-render the component, creating an updated view.
* State updates trigger automatic re-rendering of the component.
* State ensures the user interface (UI) stays in sync with data changes.

**State vs. Props**

* Props are data passed into a component from outside.
* State is internal to the component and can change over time.

**Using State**

* Managed using hooks like useState or useReducer.
* Can also be managed with Context API or external tools like Redux.



**Component-Specific State**

* Each component has and manages its own state.
* State in one component doesn't affect the state in other instances of the same component.

**UI as a Function of State**

* The entire application view is a reflection of the current state in all components.
* React applications are about changing state over time and displaying it correctly.

**Practical Guidelines for Using State**

* Create state variables for data that a component needs to track over time.
* Use state for variables that need to change in the future or make a component dynamic.
* Avoid using state for variables that don't need to trigger re-renders to prevent unnecessary re-renders and potential performance issues.