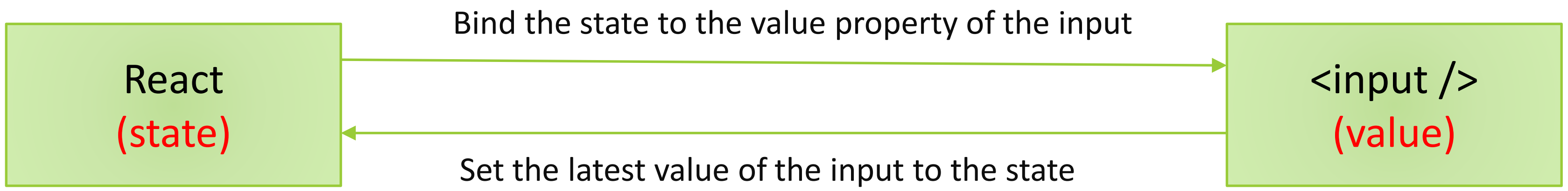
Components, useRef, CSS

RUJUAN XING

Controlled Components

Controlled Components

1. State-Based: Controlled components are tightly coupled with React's state management. They use React state to store and update the value of form elements.
2. Controlled by React: The value of the form element is controlled by React. This means that the value is set and updated through React's state management, and React maintains full control over the form element's value.
3. Explicit Updates: When the user interacts with a controlled form element, an event handler is used to update the state with the new value, causing the component to re-render and reflect the updated value.


Bind the state to the value property of the input
React
(state)
<input />
(value)
Set the latest value of the input to the state

Controlled Components Demo

```
interface FormData {
  username: string;
  password: string;
}

function App() {

  const [formState, setFormState] = useState<FormData>({username: '', password: ''});

  const handleInputChange = (e: ChangeEvent<HTMLInputElement>) => {
    const {name, value} = e.target;
    setFormState({...formState, [name]: value});
  }

  const handleSubmit = (e: FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    console.log(formState);
  }

  return (
    <form onSubmit={handleSubmit}>
      Username: <input name="username" value={formState.username} onChange={handleInputChange}/>
      Password: <input name="password" value={formState.password} onChange={handleInputChange}/>
      <button>Login</button>
    </form>
  );
}
```

DOM, useRef

Access DOM in React

In general, we don't directly manipulate the DOM. However, in some situations, how do we access the DOM in React?

Two Steps:

1. Create a `ref` object using `useRef` hook and bind it with `JSX`.
2. Access the DOM object through `inputRef.current`.

```
  const inputRef = useRef<HTMLInputElement>(null);

  const getDom = (e: SyntheticEvent) => {
    console.dir(inputRef.current?.value);
  }

  return (
    <>
      <input ref={inputRef} />
      <button onClick={getDom}>Get DOM</button>
    </>
  );
}
```

Ref Hook - `useRef`

`useRef` is a React Hook that lets you reference a value that’s not needed for rendering.

```
const ref = useRef(initialValue)
```

- `initialValue`: The value you want the `ref` object’s current property to be initially. It can be a value of any type. This argument is ignored after the initial render.
- `useRef` returns an object with a single property:
  - `current`: Initially, it’s set to the `initialValue` you have passed. You can later set it to something else. If you pass the `ref` object to React as a `ref` attribute to a JSX node, React will set its `current` property.
  - On the next renders, `useRef` will return the same object.

When to use Refs

Typically, you will use a ref when your component needs to “step outside” React and communicate with external APIs—often a browser API that won’t impact the appearance of the component. Here are a few of these rare situations:

1. Storing timeout IDs
2. Storing and manipulating DOM elements, which we cover on the next page
3. Storing other objects that aren’t necessary to calculate the JSX.

If your component needs to store some value, but it doesn’t impact the rendering logic, choose refs.

useRef & TypeScript

When retrieve DOM, you can directly pass the type of the DOM element you want to retrieve as a generic parameter to `useRef`, which allows the type of the `.current` property to be inferred.

```
  const inputRef = useRef<HTMLInputElement>(null);


  return (
    <>
      <input ref={inputRef} />
    </>
  );
}
```

Uncontrolled Components

The uncontrolled components where form data is handled by the DOM itself. So in order to access any value that has been entered we take the help of refs.

```
function App() {
  const usernameRef = useRef<HTMLInputElement>(null);
  const passwordRef = useRef<HTMLInputElement>(null);

  const handleSubmit = (e: FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    console.log(usernameRef.current!.value, passwordRef.current!.value);
  }

  return (
    <form onSubmit={handleSubmit}>
      Username: <input name="username" ref={usernameRef}/>
      Password: <input name="password" ref={passwordRef}/>
      <button>Login</button>
    </form>
  );
}
```

CSS

REACT

CSS

Two ways to add CSS in React:

- Inline Style

```
<button style={{color: 'red'}}>Change Name</button>
```

- Use className

```
import './App.css';

<p className='foo'>use className</p>
```

```
.foo {
  color: pink
}
```

Modularize CSS in React

To support CSS Modules alongside regular stylesheets using the [name].module.css file naming convention.

```
import hello from './Hello.module.css';
export default function Hello(){
    return (
        <div>
            <h1 className={hello.title}>Hello Component!!</h1>
        </div>
    )
}
```

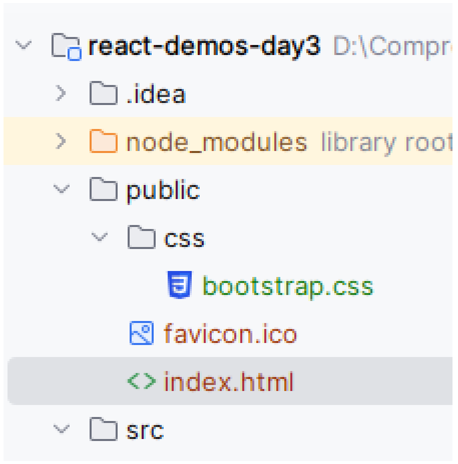
Alternatives:

- Use Less, Sass
- styled-components

Bootstrap

1. Download bootstrap.css
2. Place in `public` -> `css` folder
3. Link in `index.html`

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link href="css/bootstrap.css" type="text/css" rel="stylesheet" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```



```
react-demos-day3 D:\Compr
  .idea
  node_modules library root
  public
    css
      bootstrap.css
    favicon.ico
    index.html
  src
```

Main Point

React Hooks, introduced in React 16.8, are a set of functions that enable developers to add state and side-effect handling to functional components. This innovation allows for the management of component logic without the need for class components. By using hooks like useState and useEffect, developers can achieve cleaner, more readable code and better component reusability. Hooks have become a cornerstone of modern React development, simplifying state management and making it more approachable for both new and experienced developers.

The existence of a unified field of all the laws of nature, from which everything in the universe originates. It is described as a field of pure consciousness and the source of all knowledge and creativity.