

A decorative wavy line in orange and white, running vertically along the left side of the slide.

React **Component**

Props, Refs & State

Content

1. Component?
2. Why Component?
3. Creating Components
4. Props & State?
5. Working with Props
6. Refs & the DOM
7. Component Lifecycle
8. Working with State

1. Component?

- HTML, CSS & JavaScript are about building user interfaces as well. React makes building complex, interactive and reactive user interfaces simpler.
- React is about “Components”
- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Components are like TypeScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

Add New Expense

Filter by year

2020



Jan



Feb



Mar



Apr



May



Jun



Jul



Aug



Sep



Oct



Nov



Dec

December
2020

16

Some Books

\$ 39

August
2020

14

Toilet Paper

\$ 94.12

2. Why Component?



Reusability

Don't repeat yourself



Separation of Concerns

Don't do too many things in one and the same place (function)



Split big chunks of code into multiple smaller functions



3. Creating Components

- 3.1. Creating Login component
- 3.2. Creating reusable component

3.1. Creating Login component

- Create a new file named “Login.tsx” in “src” folder

```
import React from "react";
class Login extends React.Component {
  render() {
    return (
      <>
      </>
    );
  }
}

export default Login;
```

- Copy all JSX codes from `App.tsx` to `Login.tsx`

```
import React from "react";  
class Login extends React.Component {  
  render() {  
    return (  
      <div className="container h-100"> ...  
    </div>  
    );  
  }  
}
```

- Modify `App.tsx` file

```
import "./App.css";  
import Login from "./Login";  
function App() {  
  return <Login />;  
}  
  
export default App;
```


3.2. Creating Reusable Component

 Login

User name

Password

Sign in

```
<div className="row mb-3">
  <label htmlFor="txtUsername" className="col-sm-3 col-form-label">User name</label>
  <div className="col-sm-9">
    <input type="text" className="form-control" id="txtUsername" />
  </div>
</div>
```

```
<div className="row mb-3">
  <label htmlFor="txtPassword" className="col-sm-3 col-form-label">Password</label>
  <div className="col-sm-9">
    <input type="password" className="form-control" id="txtPassword" />
  </div>
</div>
```

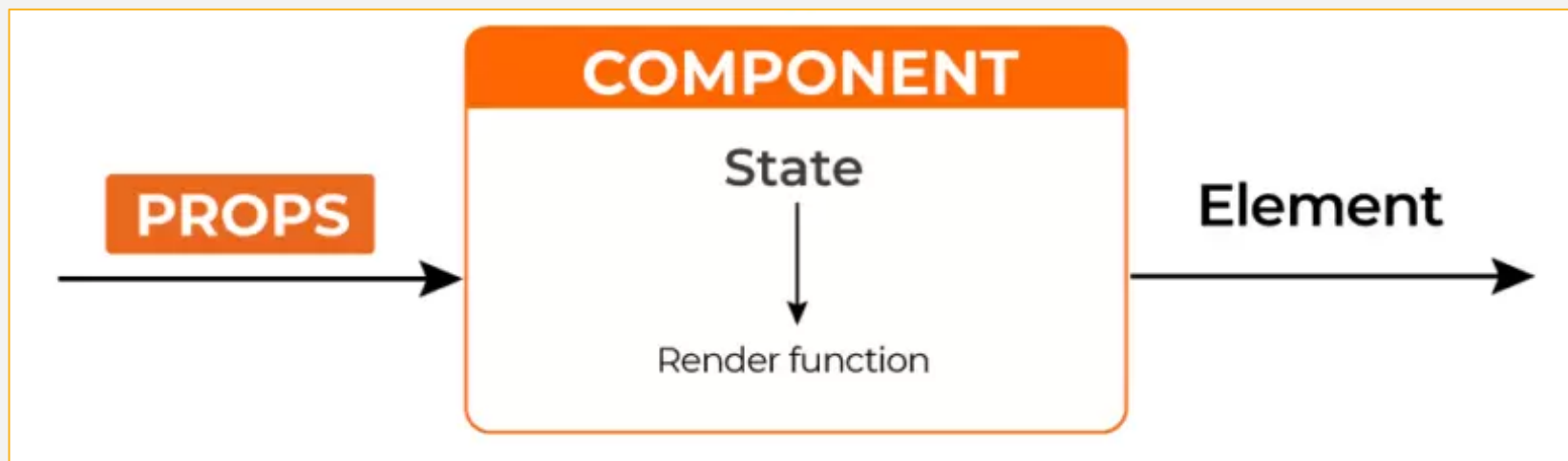
Creating **Input** Component

- Create a new folder named “**components**”
- In “**components**” folder, create a new file named “**Input.tsx**”
- Create a new class component named “**Input**” as below

```
import React from "react";
class Input extends React.Component {
  render() {
    return (
      <div className="row mb-3">
        <label htmlFor="inputEmail3" className="col-sm-3 col-form-label">
          User name
        </label>
        <div className="col-sm">
          <input type="text" className="form-control" id="inputEmail3" />
        </div>
      </div>
    );
  }
}
export default Input;
```

4. Props & State?

- “**props**” is short for “properties”.
- props are passed into the component (like parameters)
- **state** is **managed** within the component (similar to variables)
- Both **props** and **state** are plain JS objects
- Both **props** and **state** changes **trigger** a **render** update
- Changing **props** and **state**



Props vs. State

- ★ Immutable
- ★ Has better performance
- ★ Can be passed to child components



- ★ Owned by its component
- ★ Locally scoped
- ★ Writeable / Mutable
- ★ Has setState() method to modify properties
- ★ Changes to state can be asynchronous
- ★ Can only be passed as props



5. Working with props

```
import React, { DetailedHTMLProps, InputHTMLAttributes } from "react";

type InputAttributes = InputHTMLAttributes<HTMLInputElement>;

interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> {
  id: string;
  label: string;
}

> class Input extends React.Component<Props> { ...
}

export default Input;
```

```
> interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> { ...  
}
```

```
class Input extends React.Component<Props> {  
  render() {  
    const { id, label, className, ...others } = this.props;  
    const inputClass = `form-control ${className ? className : ""}`;  
    return (  
      <div className="row mb-3">  
        <label htmlFor={id} className="col-sm-3 col-form-label">  
          {label}  
        </label>  
        <div className="col-sm">  
          <input id={id} {...others} className={inputClass} />  
        </div>  
      </div>  
    );  
  }  
}
```

```
export default Input;
```

Passing Props in Login Component

```
<form>
  <Input
    id="txtUsername" label="User name"
    type="text" maxLength="50"
    autoComplete="off" placeholder="Enter user name"
  />
  <Input
    id="txtPassword" label="Password"
    type="password" maxLength="100"
    placeholder="Enter password"
  />
  <div className="row">
    <div className="offset-sm-3 col-auto">
      <button type="submit" className="btn btn-primary">
        Sign in
      </button>
    </div>
  </div>
</form>
```

Controlling Label Width

```
interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> {
  id: string;
  label: string;
  labelSize?: number;
}

class Input extends React.Component<Props> {
  render() {
    const { id, label, labelSize = 3, className = "", ...others } = this.props;
    const labelClass = `col-sm-${labelSize} col-form-label`;
    const inputClass = `form-control ${className}`;
    return (
      <div className="row mb-3">
        <label htmlFor={id} className={labelClass}>
          {label}
        </label>
        <div className="col-sm">
          <input id={id} {...others} className={inputClass} />
        </div>
      </div>
    );
  }
}
```


Supporting Multiple Rows

```
import React, {
  DetailedHTMLProps,
  InputHTMLAttributes,
  TextareaHTMLAttributes } from "react";

type InputAttributes = InputHTMLAttributes<HTMLInputElement>;

interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> {
  id: string;
  label: string;
  labelSize?: number;
  rows?: number;
}

> class Input extends React.Component<Props> { ...
}

export default Input;
```

```
class Input extends React.Component<Props> {
  render() {
    const { id, label, labelSize = 3, rows = 1, className = "", ...others } = this.props;
    const labelClass = `col-sm-${labelSize} col-form-label`;
    const inputClass = `form-control ${className}`;
    return (
      <div className="row mb-3">
        <label htmlFor={id} className={labelClass}>{label}</label>
        <div className="col-sm">
          {rows > 1 ? (
            <textarea id={id} rows={rows}
              {...(others as TextareaHTMLAttributes<HTMLTextAreaElement>)}
              className={inputClass}
            ></textarea>
          ) : ( <input id={id} {...others} className={inputClass} /> )}
        </div>
      </div>
    );
  }
}

export default Input;
```

6. Refs & the DOM

- Refs provide a way to access DOM nodes or React elements created in the render method.
- There are a few good use cases for refs:
 - Managing focus, text selection, or media playback.
 - Triggering imperative animations.
 - Integrating with third-party DOM libraries.
- Don't Overuse Refs

Your first inclination may be to use refs to “make things happen” in your app. If this is the case, take a moment and think more critically about where state should be owned in the component hierarchy.

Working with DOM

- In “Login.tsx” file, modify following codes

```
class Login extends React.Component {  
  usernameRef = React.createRef<any>();  
  passwordRef = React.createRef<any>();  
  
>  render() { ...  
    }  
}
```

```
<Input  
  inputRef={this.usernameRef}  
  id="txtUsername" label="User name"  
  type="text" autoComplete="off" maxLength={50} />  
<Input  
  inputRef={this.passwordRef}  
  id="txtPassword" label="Password" type="password" />
```

- In “Input.tsx”, add “inputRef” property

```
import React, {
  DetailedHTMLProps,
  InputHTMLAttributes,
  TextareaHTMLAttributes } from "react";

type InputAttributes = InputHTMLAttributes<HTMLInputElement>;

interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> {
  id: string;
  inputRef?: any;
  label: string;
  labelSize?: number;
  rows?: number;
}

> class Input extends React.Component<Props> { ...
}

export default Input;
```

```
> interface Props extends DetailedHTMLProps<InputAttributes, HTMLInputElement> { ...
}

class Input extends React.Component<Props> {
  render() {
    const { inputRef, id, label,
      labelSize = 3, rows = 1, className = "",
      ...others } = this.props;
    const labelClass = `col-sm-${labelSize} col-form-label`;
    const inputClass = `form-control ${className}`;
    > return ( ...
      );
    }
  }

  export default Input;
```

```
const inputClass = `form-control ${className}`;
return (
  <div className="row mb-3">
    <label htmlFor={id} className={labelClass}>
      {label}
    </label>
    <div className="col-sm">
      {rows > 1 ? (
        <textarea ref={inputRef} id={id} rows={rows}
          {...(others as TextareaHTMLAttributes<HTMLTextAreaElement>)}
          className={inputClass}
        ></textarea>
      ) : (
        <input ref={inputRef} id={id} {...others} className={inputClass} />
      )}
    </div>
  </div>
);
```

Handling Form Submission

```
class Login extends React.Component {  
  usernameRef = React.createRef<any>();  
  passwordRef = React.createRef<any>();  
  
  handleSubmitHandler = (e: React.FormEvent<HTMLFormElement>) => {  
    e.preventDefault();  
    const username = this.usernameRef.current?.value;  
    const password = this.passwordRef.current?.value;  
  
    console.log(username, password);  
  };  
  
  > render() { ...  
    }  
}  
  
export default Login;
```



```
<div className="card-body bg-white rounded-bottom">
  <form onSubmit={this.formSubmitHandler}>
    <Input
      inputRef={this.usernameRef} ...
      maxLength={50}
    />
    <Input inputRef={this.passwordRef} ...
    <div className="row">
      <div className="offset-sm-3 col-auto">
        <button type="submit" className="btn btn-primary">
          Sign in
        </button>
      </div>
    </div>
  </form>
</div>
```

Testing

localhost:3000

Login

User name

admin

Password

.....

Sign in

Inspector

Console

Debugger

Network

Style Editor

Filter Output

Errors

Warnings

Logs

Info

Debug

CSS

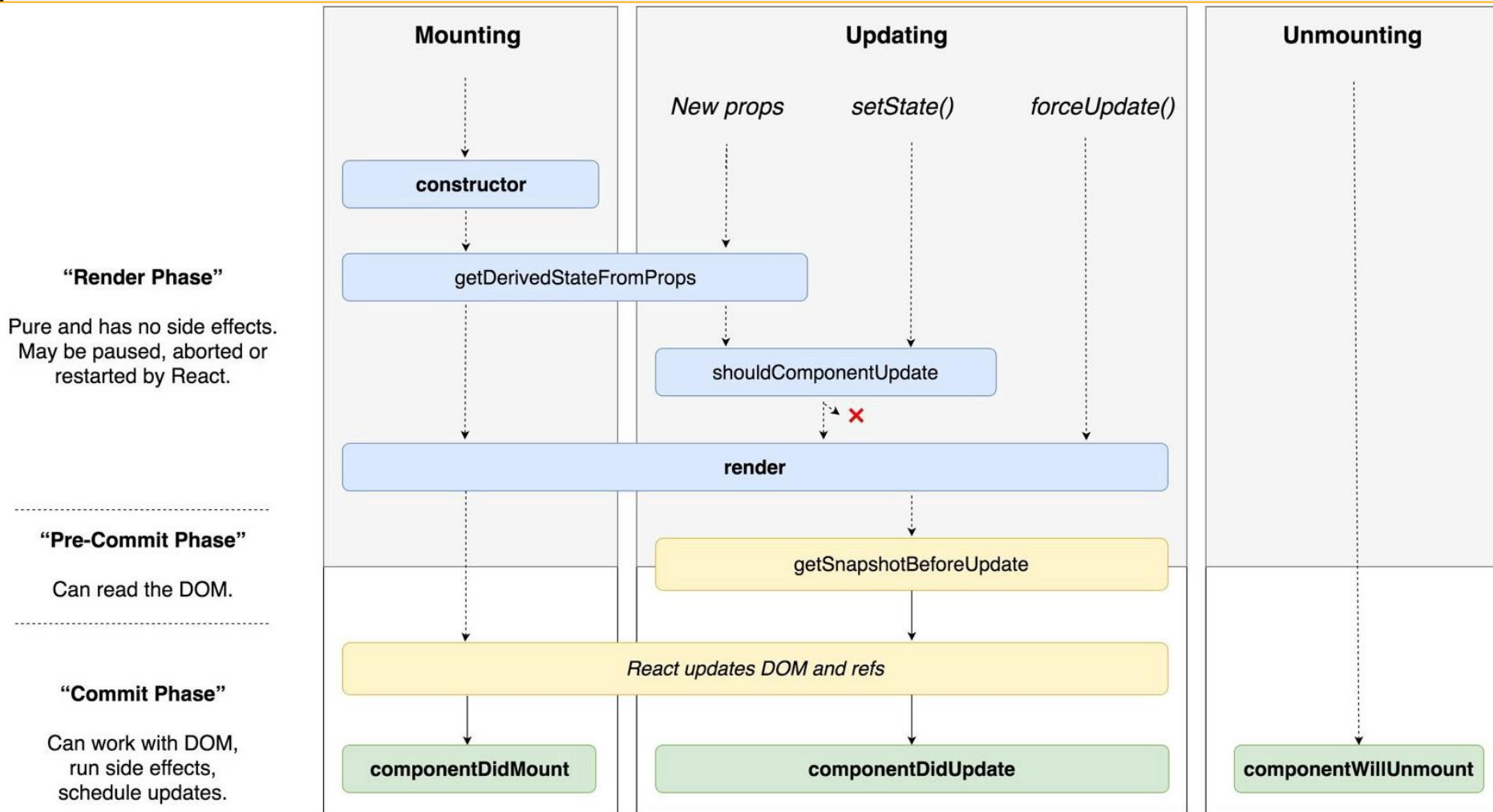
XHR

Requests

Download the React DevTools for a better development experience: [react-dom.development.js:29742](https://reactjs.org/link/react-devtools)
<https://reactjs.org/link/react-devtools>

admin 123456 Login.js:10

7. Component Lifecycle



What to do/not to do with lifecycle

■ constructor

- Do
 - set initial state
 - if not using class properties syntax — prepare all class fields and bind functions that will be passed as callbacks
- Don't
 - cause any side effects (AJAX calls etc.)

■ componentDidMount

- Do
 - Perfect place to call AJAX.

- **componentDidUpdate(prevProps, prevState, snapshot)**

- Do

- cause side effects (AJAX calls etc.)

- **componentWillUnmount**

- Do

- remove any timers or listeners created in lifespan of the component

- Don't

- call **this.setState**, start new listeners or timers

Using `componentDidMount` Event


```
class Login extends React.Component {  
  usernameRef = React.createRef<any>();  
  passwordRef = React.createRef<any>();  
  
>  handleSubmitHandler = (e: React.FormEvent<HTMLFormElement>) => { ...  
    };  
  
  componentDidMount(): void {  
    |   this.usernameRef.current?.focus();  
  }  
  
>  render() { ...  
    }  
}  
  
export default Login;
```

8. Working with State

```
type LoginStates = {  
  message: string;  
};
```

```
class Login extends React.Component<{}, LoginStates> {  
  usernameRef = React.createRef<any>();  
  passwordRef = React.createRef<any>();  
  
  formSubmitHandler = (e: React.FormEvent<HTMLFormElement>) => {  
    e.preventDefault();  
    const username = this.usernameRef.current?.value;  
    const password = this.passwordRef.current?.value;  
    if (username === "admin" && password === "123456") {  
      this.setState({ message: "Good!" });  
    } else {  
      this.setState({ message: "Bad!" });  
    }  
  };  
};
```


Testing

 Login

Good!

User name

Password



THE END