React - UI, Forms, Events, Animation

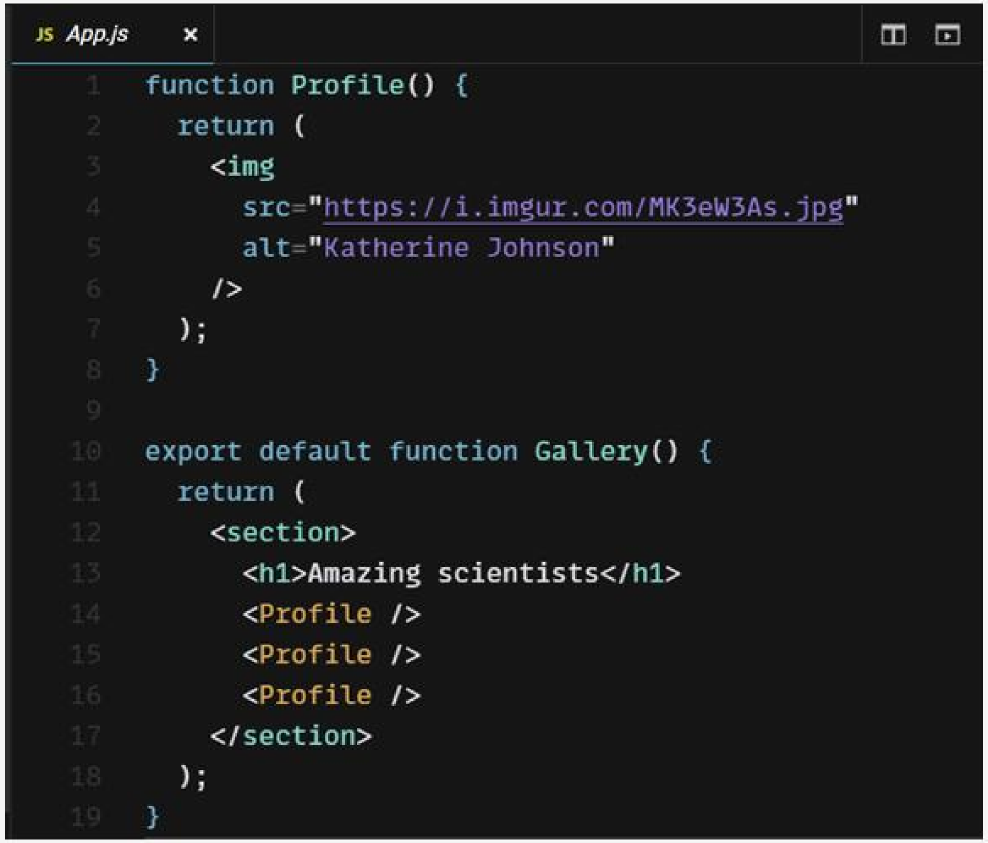
Vijesh M. Nair
Assistant Professor
Dept. of CSE (AI-ML)

Describing the UI

- React is a JavaScript library for rendering user interfaces (UI).
- UI is built from small units like buttons, text, and images.
- React lets you combine them into reusable, nestable *components*.
- From web sites to phone apps, everything on the screen can be broken down into components.
- React applications are built from isolated pieces of UI called *components*.
- A React component is a JavaScript function that you can sprinkle with markup.
- Components can be as small as a button, or as large as an entire page.

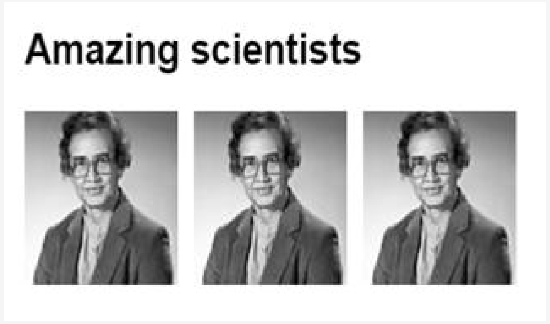
Describing the UI - Example

Here is a `Gallery` component rendering three `Profile` components:



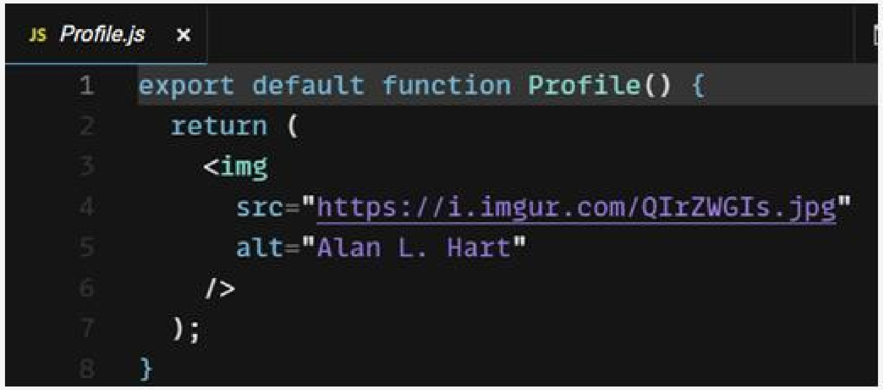
App.js

```
function Profile() {
  return (
    <img
      src="https://i.imgur.com/MK3eW3As.jpg"
      alt="Katherine Johnson"
    />
  );
}

export default function Gallery() {
  return (
    <section>
      <h1>Amazing scientists</h1>
      <Profile />
      <Profile />
      <Profile />
    </section>
  );
}
```



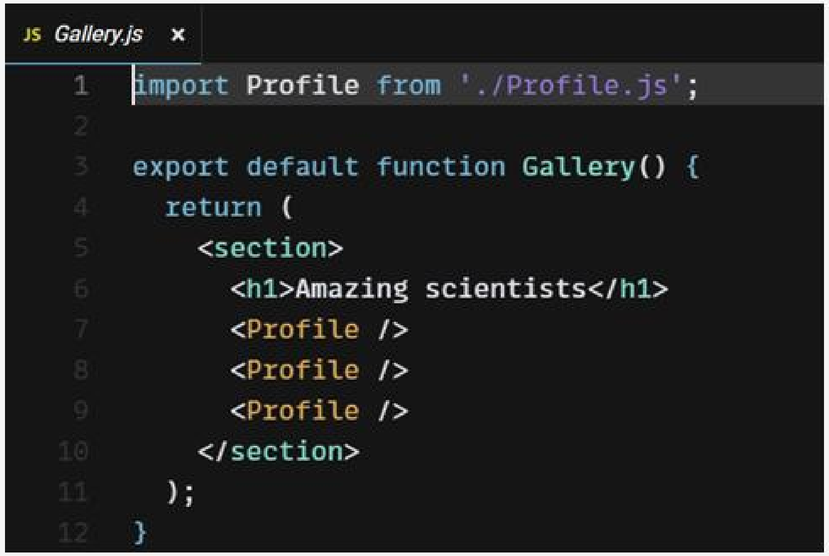
Importing and exporting components

- We can declare many components in one file, but large files can get difficult to navigate.
- To solve this, you can export a component into its own file, and then import that component from another file:



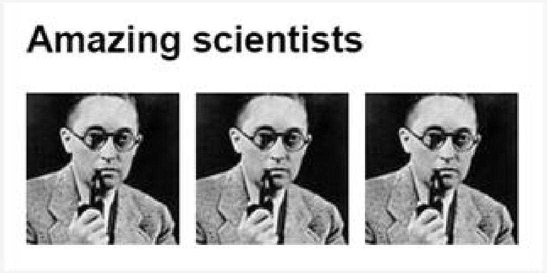
Profile.js

```
export default function Profile() {
  return (
    <img
      src="https://i.imgur.com/QIrZWGIs.jpg"
      alt="Alan L. Hart"
    />
  );
}
```



Gallery.js

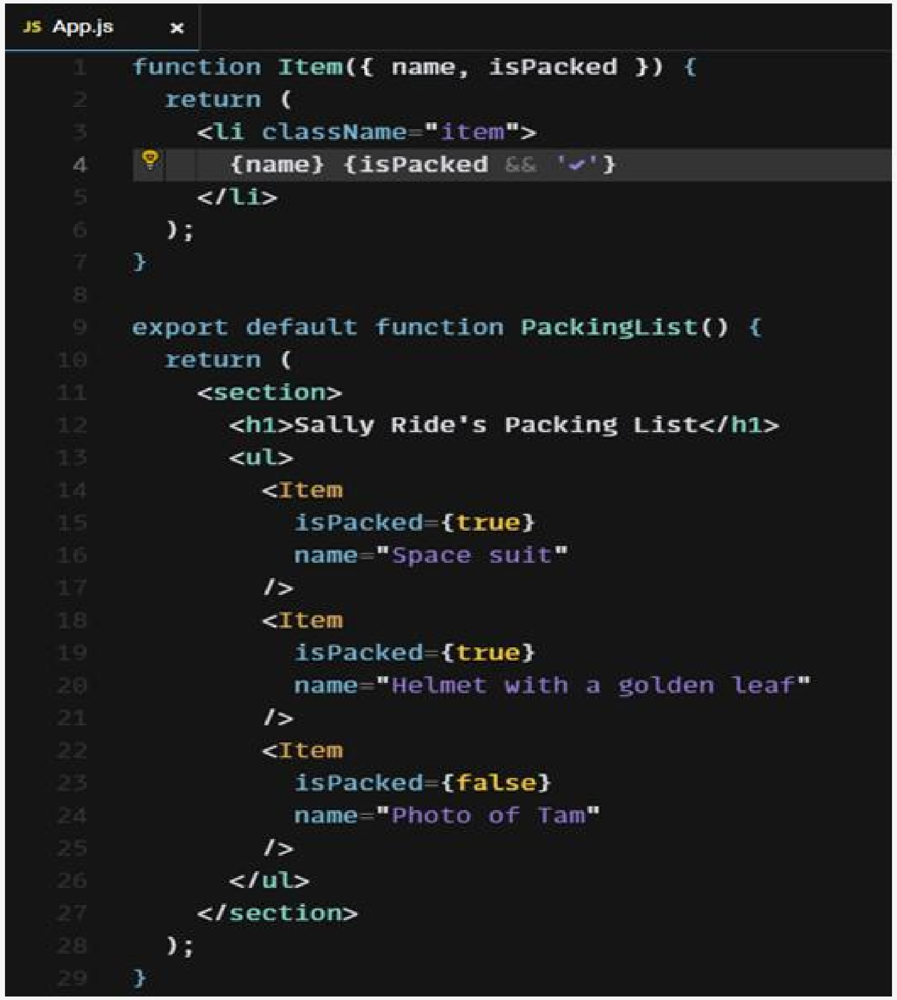
```
import Profile from './Profile.js';

export default function Gallery() {
  return (
    <section>
      <h1>Amazing scientists</h1>
      <Profile />
      <Profile />
      <Profile />
    </section>
  );
}
```



Conditional rendering

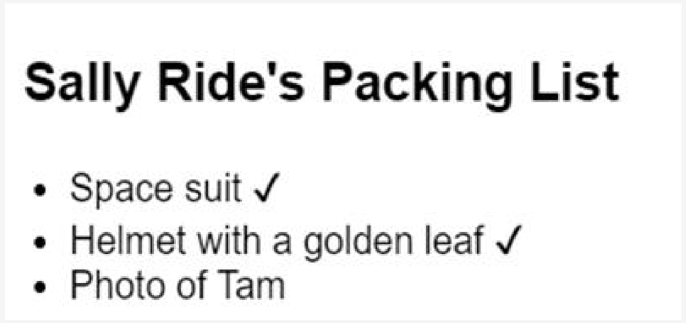
- Our components will often need to display different things depending on different conditions.
- In React, you can conditionally render JSX using JavaScript syntax like `if` statements, `&&`, and `? :` operators.
- In this example, the JavaScript `&& operator` is used to conditionally render a checkmark:

Conditional rendering



App.js

```
function Item({ name, isPacked }) {
  return (
    <li className="item">
      {name} {isPacked && '✓'}
    </li>
  );
}

export default function PackingList() {
  return (
    <section>
      <h1>Sally Ride's Packing List</h1>
      <ul>
        <Item
          isPacked={true}
          name="Space suit"
        />
        <Item
          isPacked={true}
          name="Helmet with a golden leaf"
        />
        <Item
          isPacked={false}
          name="Photo of Tam"
        />
      </ul>
    </section>
  );
}
```



Sally Ride's Packing List

- Space suit ✓
- Helmet with a golden leaf ✓
- Photo of Tam

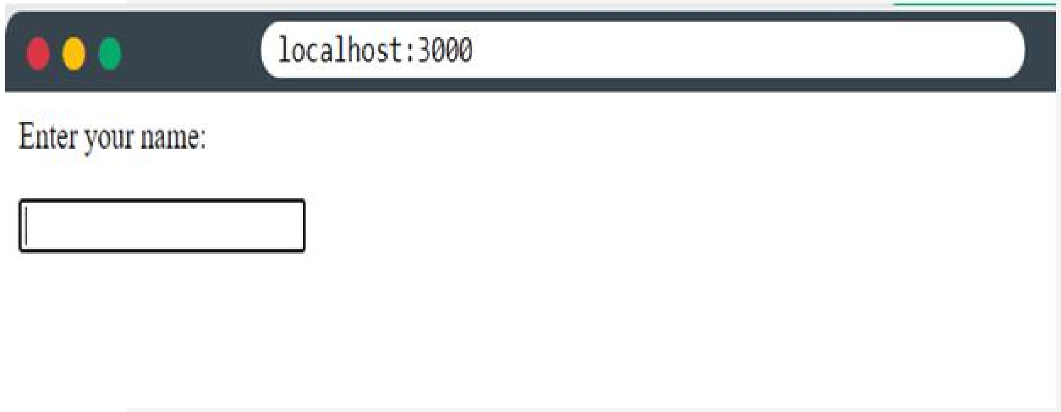
React - Forms

Handling Forms

- Handling forms is about how you handle the data when it changes value or gets submitted.
- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
- When the data is handled by the components, all the data is stored in the component state.
- We can control changes by adding event handlers in the onChange attribute.
- We can use the useState Hook to keep track of each inputs value and provide a "single source of truth" for the entire application.

Handling Forms

```
import { useState } from "react";
import ReactDOM from 'react-dom/client';

function MyForm() {
  const [name, setName] = useState("");

  return (
    <form>
      <label>Enter your name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```



Enter your name:

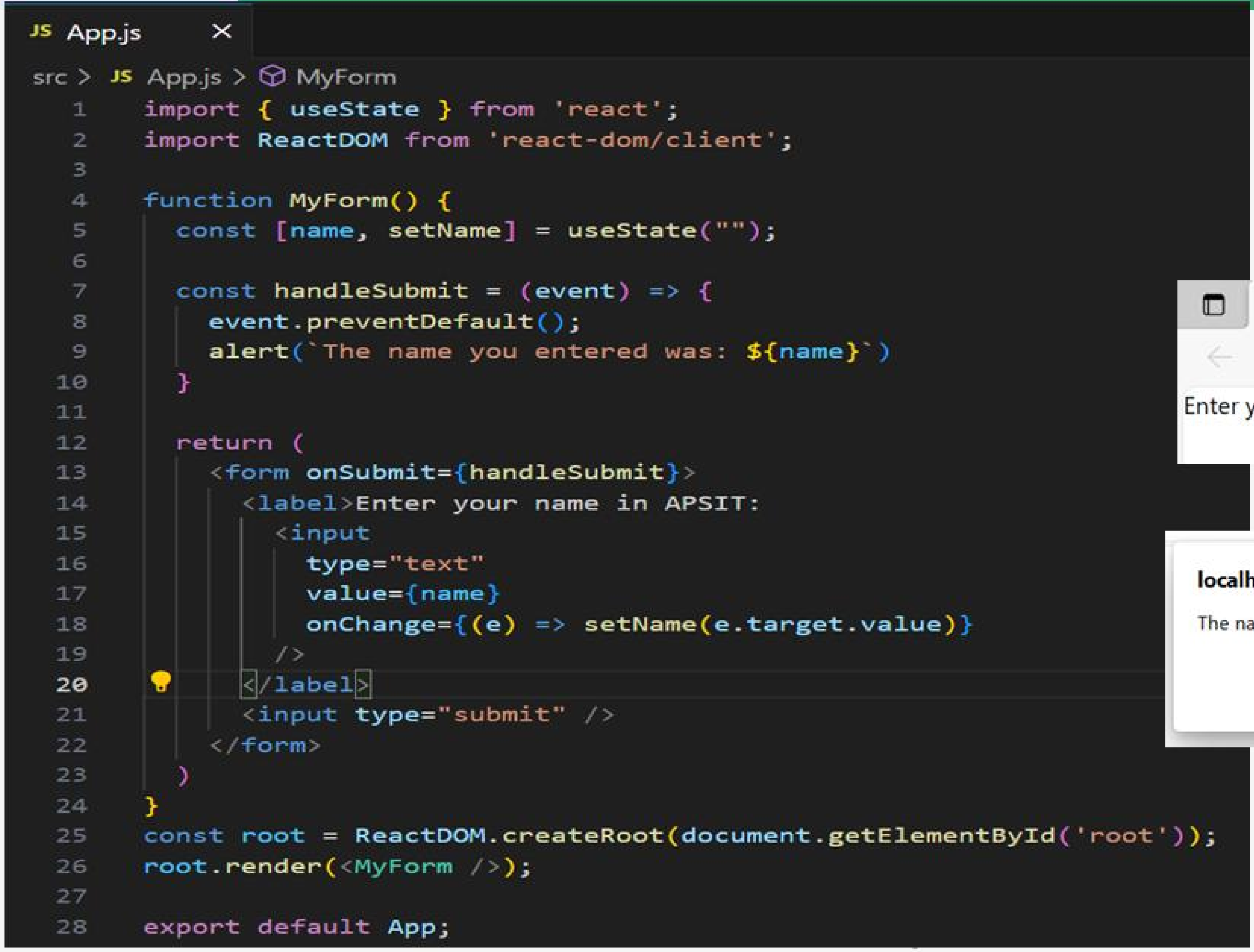
Submitting Forms

You can control the submit action by adding an event handler in the `onSubmit` attribute for the `<form>`:

Example:

Add a submit button and an event handler in the `onSubmit` attribute:

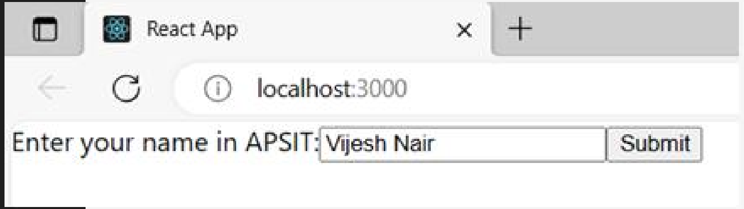
Submitting Forms



App.js

src > App.js > MyForm

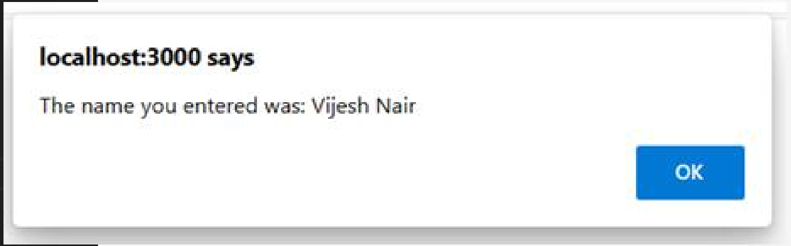
```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function MyForm() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`The name you entered was: ${name}`)
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name in APSIT:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);

export default App;
```



React App

localhost:3000

Enter your name in APSIT: Vijesh Nair Submit

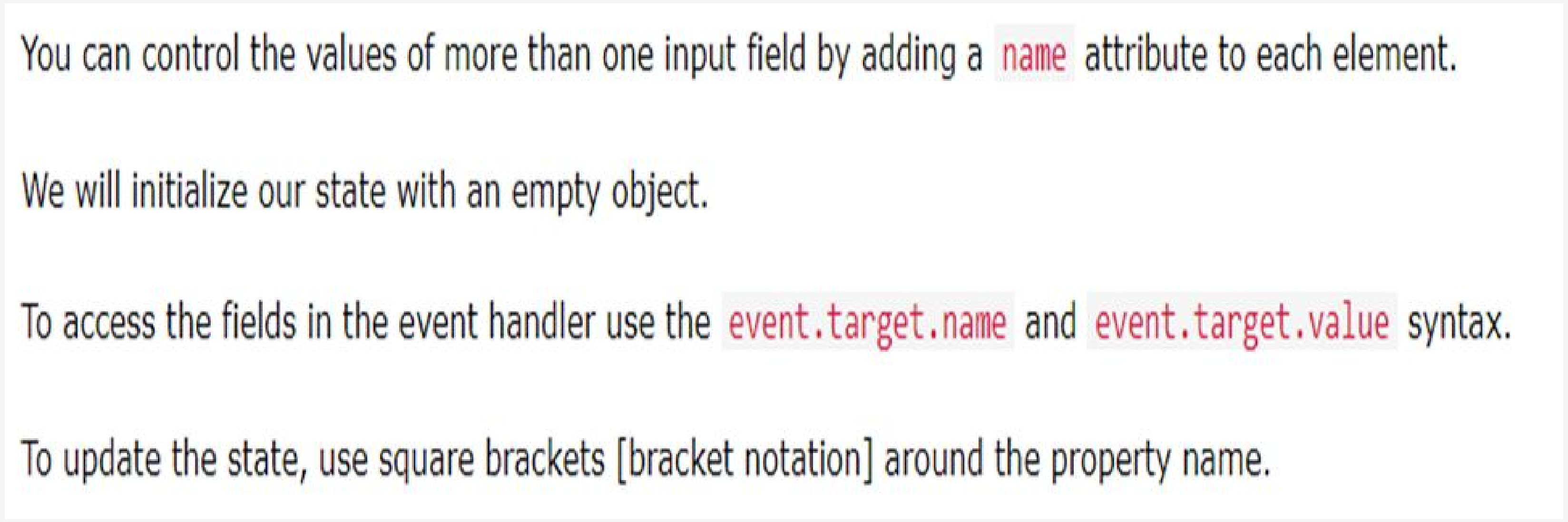


localhost:3000 says

The name you entered was: Vijesh Nair

OK

Multiple Input Fields



You can control the values of more than one input field by adding a `name` attribute to each element.

We will initialize our state with an empty object.

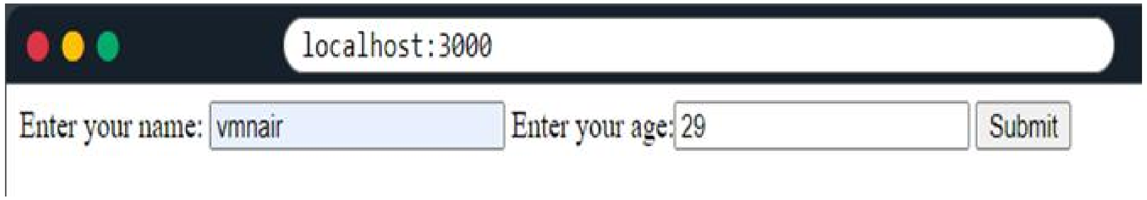
To access the fields in the event handler use the `event.target.name` and `event.target.value` syntax.

To update the state, use square brackets [bracket notation] around the property name.

Multiple Input Fields

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function MyForm() {
  const [inputs, setInputs] = useState({});

  const handleChange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setInputs(values => ({...values, [name]: value}))
  }

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log(inputs);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:
      <input
        type="text"
        name="username"
        value={inputs.username || ""}
        onChange={handleChange}
      />
      </label>
      <label>Enter your age:
```

```
        <input
          type="number"
          name="age"
          value={inputs.age || ""}
          onChange={handleChange}
        />
        </label>
        <input type="submit" />
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```



Enter your name: vmnair Enter your age: 29 Submit

Vijesh Nair

Textarea

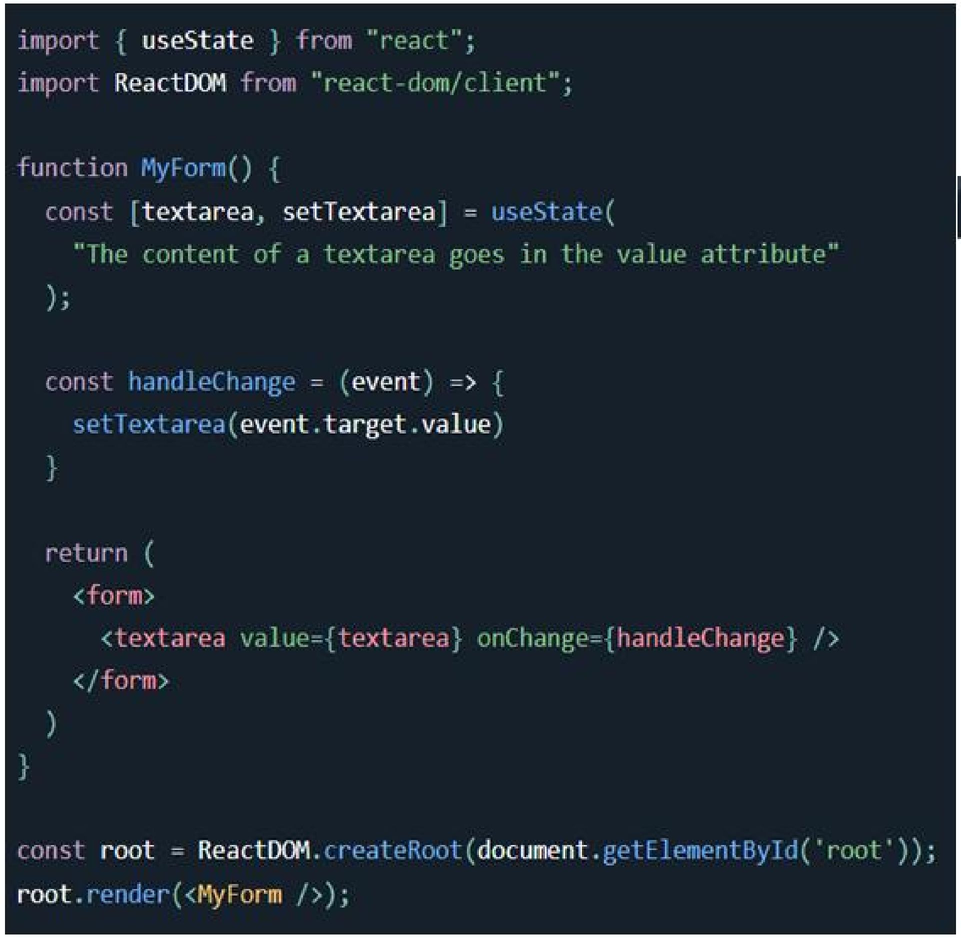
The textarea element in React is slightly different from ordinary HTML.

In HTML the value of a textarea was the text between the start tag `<textarea>` and the end tag `</textarea>`.

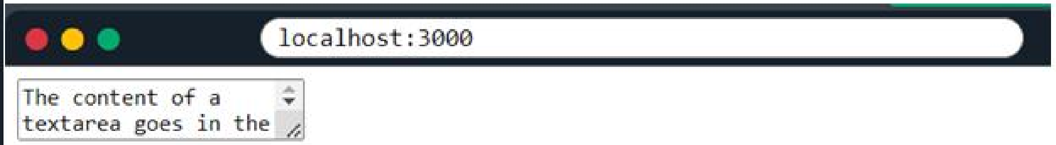
```
<textarea>
  Content of the textarea.
</textarea>
```

In React the value of a textarea is placed in a value attribute. We'll use the `useState` Hook to manage the value of the textarea:

Textarea



```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function MyForm() {
  const [textarea, setTextarea] = useState(
    "The content of a textarea goes in the value attribute"
  );

  const handleChange = (event) => {
    setTextarea(event.target.value)
  }

  return (
    <form>
      <textarea value={textarea} onChange={handleChange} />
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```



```
The content of a
textarea goes in the
```

Select

A drop down list, or a select box, in React is also a bit different from HTML.

in HTML, the selected value in the drop down list was defined with the `selected` attribute:

HTML:

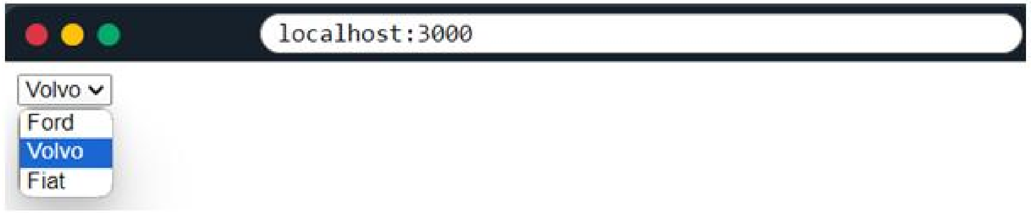
```
<select>
  <option value="Ford">Ford</option>
  <option value="Volvo" selected>Volvo</option>
  <option value="Fiat">Fiat</option>
</select>
```

In React, the selected value is defined with a `value` attribute on the `select` tag:

Select

Example:

A simple select box, where the selected value "Volvo" is initialized in the constructor:



```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function MyForm() {
  const [myCar, setMyCar] = useState("Volvo");

  const handleChange = (event) => {
    setMyCar(event.target.value)
  }

  return (
    <form>
      <select value={myCar} onChange={handleChange}>
        <option value="Ford">Ford</option>
        <option value="Volvo">Volvo</option>
        <option value="Fiat">Fiat</option>
      </select>
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

Vijesh Nair

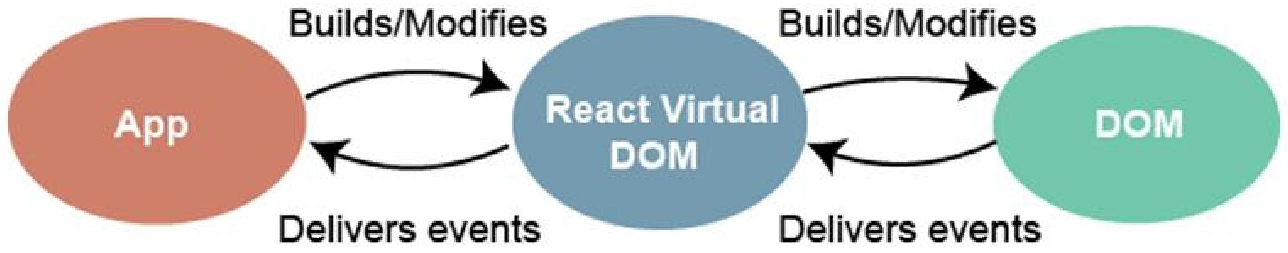
React - Events

Events

- An event is an action that could be triggered as a result of the user action or system generated event.
- For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.
- React has its own event handling system which is very similar to handling events on DOM elements.
- The react event handling system is known as Synthetic Events.
- The synthetic event is a cross-browser wrapper of the browser's native event.

Events

Events Handler



- Handling events with react have some syntactic differences from handling events on DOM. These are:

1. React events are named as **camelCase** instead of **lowercase**.
2. With JSX, a function is passed as the **event handler** instead of a **string**. For example:

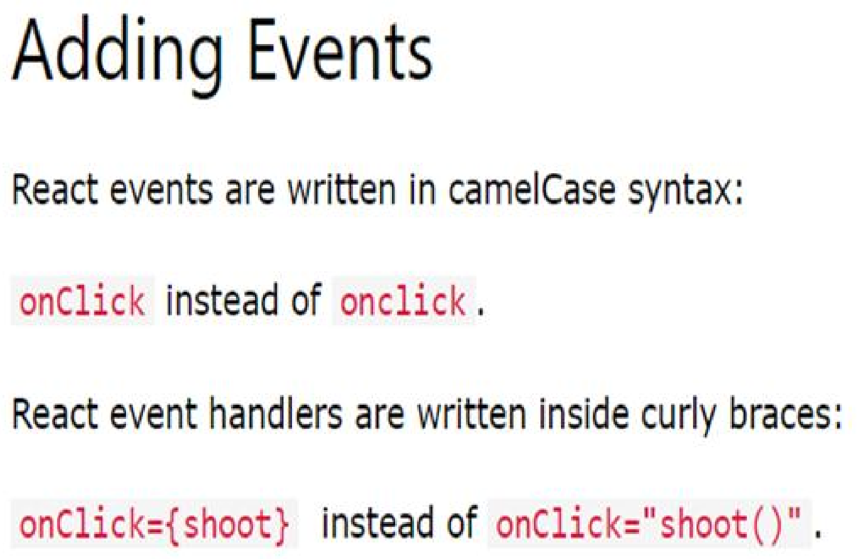
Events

Event declaration in plain HTML:

```
<button onclick="showMessage()">
    Hello APSIT..!
</button>
```

Event declaration in React:

```
<button onClick={showMessage}>
    Hello APSIT..!
</button>
```



Adding Events

React events are written in camelCase syntax:

`onClick` instead of `onclick`.

React event handlers are written inside curly braces:

`onClick={shoot}` instead of `onClick="shoot()"`.

Events

3. In react, we cannot return **false** to prevent the **default** behavior. We must call **preventDefault** event explicitly to prevent the default behavior. For example:

In plain HTML, to prevent the default link behavior of opening a new page, we can write:

```
<a href="#" onclick="console.log('You had clicked a Link.'); return false">
  Click_Me
</a>
```

Events

In React, we can write it as:

```
function ActionLink() {
   function handleClick(e) {
      e.preventDefault();
      console.log('You had clicked a Link.');
   }
   return (
      <a href="#" onClick={handleClick}>
          Click_Me
      </a>
   );
}
```

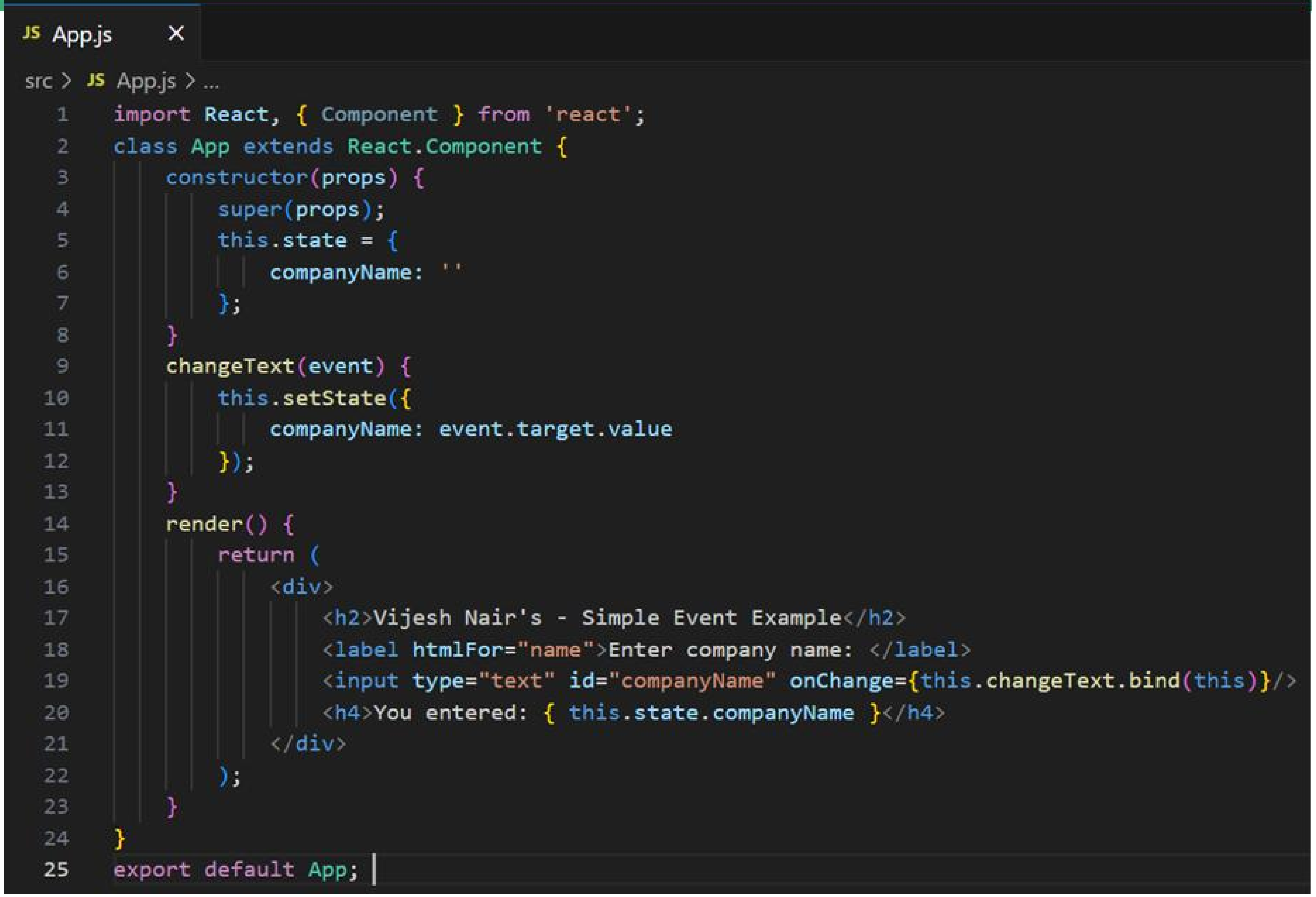
In the above example, e is a **Synthetic Event** which defines according to the **W3C** spec.

Event - Example 1

- ❑ In the below example, we have used only one component and adding an onChange event.

- ❑ This event will trigger the **changeText** function, which returns the company name.

Vijesh Nair

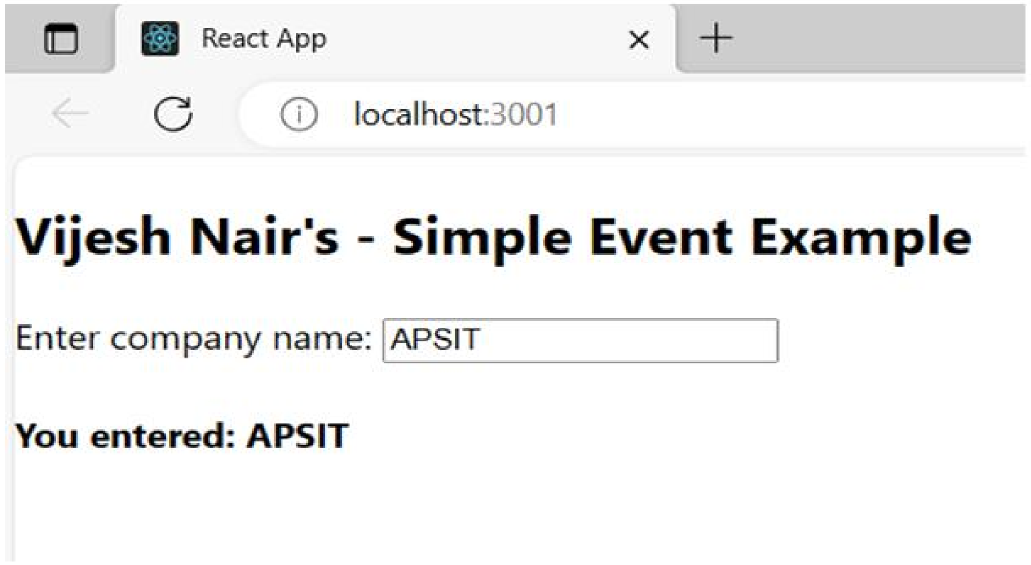


App.js

src > App.js > ...

```
import React, { Component } from 'react';
class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            companyName: ''
        };
    }
    changeText(event) {
        this.setState({
            companyName: event.target.value
        });
    }
    render() {
        return (
            <div>
                <h2>Vijesh Nair's - Simple Event Example</h2>
                <label htmlFor="name">Enter company name: </label>
                <input type="text" id="companyName" onChange={this.changeText.bind(this)}/>
                <h4>You entered: { this.state.companyName }</h4>
            </div>
        );
    }
}
export default App;
```

Event - Example 1

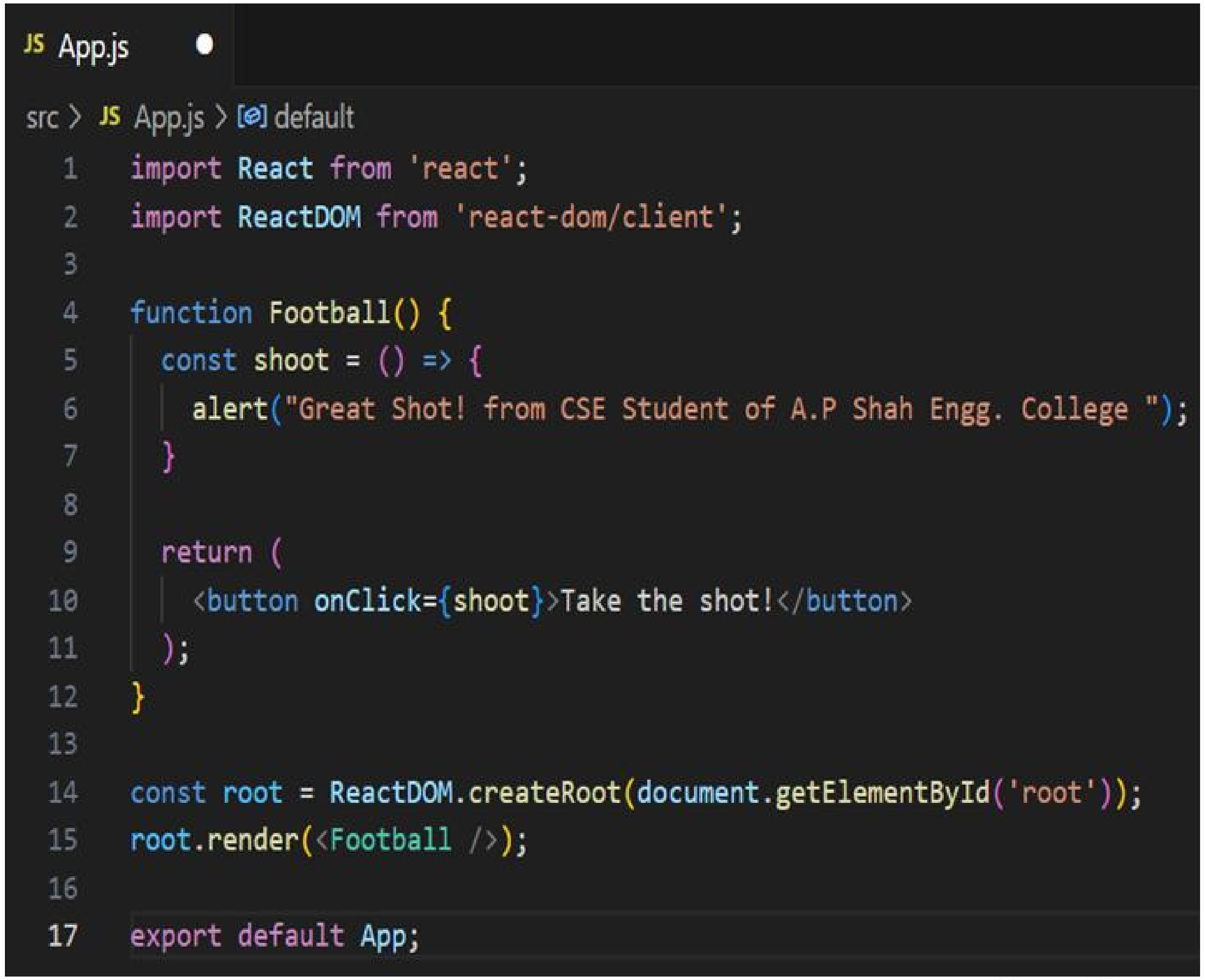


Vijesh Nair's - Simple Event Example

Enter company name: APSIT

You entered: APSIT

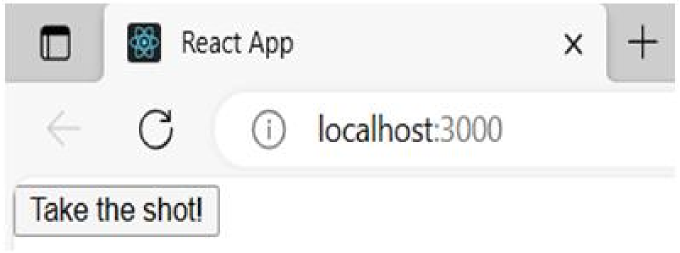
Event - Example 2

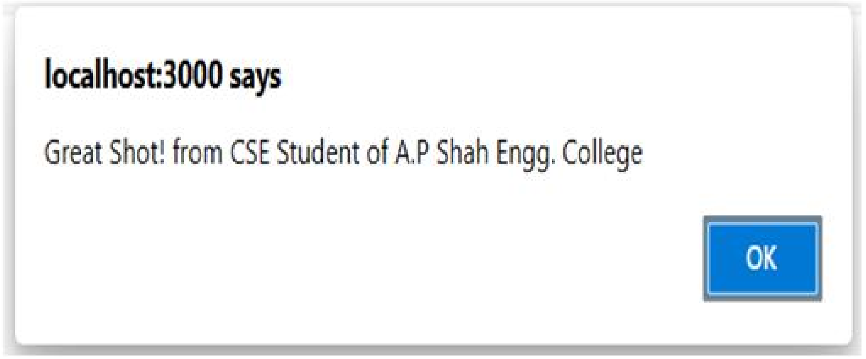


JS App.js ●

src > JS App.js > [⊘] default

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = () => {
    alert("Great Shot! from CSE Student of A.P Shah Engg. College ");
  }

  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);

export default App;
```





localhost:3000 says

Great Shot! from CSE Student of A.P Shah Engg. College

Vijesh Nair

Event - Example 3

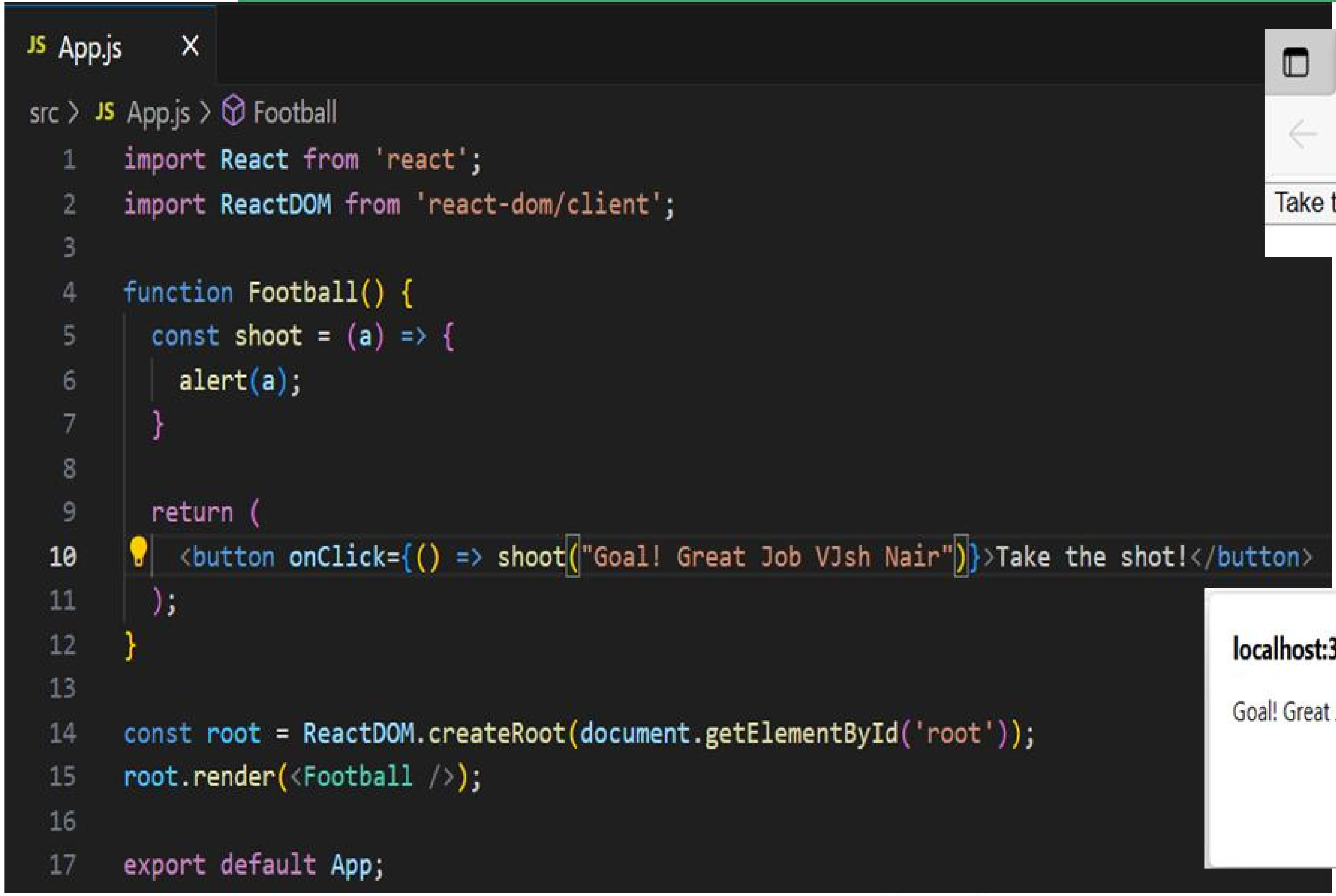
Passing Arguments

To pass an argument to an event handler, use an arrow function.

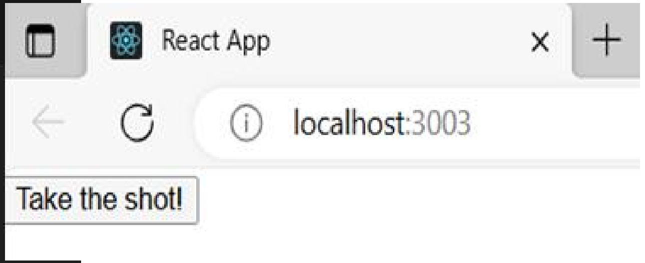
Example:

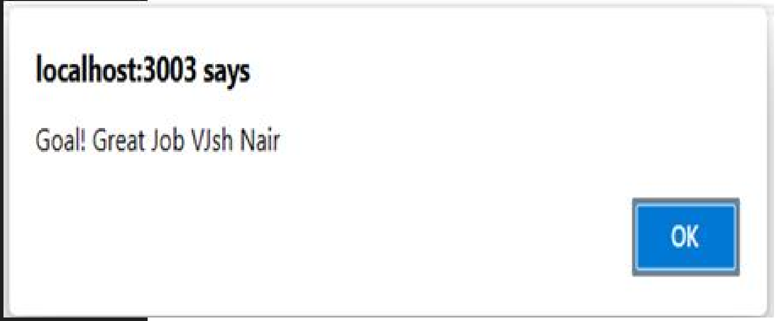
Send "Goal!" as a parameter to the `shoot` function, using arrow function:

Event - Example 3



```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = (a) => {
    alert(a);
  }

  return (
    <button onClick={() => shoot("Goal! Great Job VJsh Nair")}>Take the shot!</button>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);

export default App;
```





localhost:3003 says

Goal! Great Job VJsh Nair

Vijesh Nair

React - Animation

React Animation

- The animation is a technique in which images are manipulated to appear as moving images.
- It is one of the most used technique to make an interactive web application.
- In React, we can add animation using an explicit group of components known as the *React Transition Group*.
- React Transition Group is an add-on component for managing component states and useful for defining **entering** and **exiting** transitions.
- It is not able to animate styles by itself. Instead, it exposes transition states, manages classes and group elements, and manipulates the DOM in useful ways.
- It makes the implementation of visual transitions much easier.

Installation of react-transition-group

React Transition group has mainly **two APIs** to create transitions. These are:

1. **ReactTransitionGroup:** It uses as a low-level API for animation.
2. **ReactCSSTransitionGroup:** It uses as a high-level API for implementing basic CSS transitions and animations.

Installation

```
$ npm install react-transition-group --save
```

React Transition Group Components

React Transition Group API provides **three** main components. These are:

1. Transition
2. CSSTransition
3. Transition Group

Transition

It has a simple component API to describe a transition from one component state to another over time. It is mainly used to animate the **mounting** and **unmounting** of a component. It can also be used for in-place transition states as well.

We can access the Transition component into four states:

React Transition Group Components

- entering
- entered
- exiting
- exited

CSSTransition

The CSSTransition component uses CSS stylesheet classes to write the transition and create animations. It is inspired by the **ng-animate** library. It can also inherit all the props of the transition component. We can divide the "CSSTransition" into **three** states. These are:

React Transition Group Components

- Appear
- Enter
- Exit

CSSTransition component must be applied in a pair of class names to the child components. The first class is in the form of **name-stage** and the second class is in the **name-stage-active**. For example, you provide the name fade, and when it applies to the 'enter' stage, the two classes will be **fade-enter** and **fade-enter-active**. It may also take a prop as Timeout which defines the maximum time to animate.

React Transition Group Components

TransitionGroup

This component is used to manage a set of transition components (Transition and CSSTransition) in a list. It is a state machine that controls the **mounting** and **unmounting** of components over time. The Transition component does not define any animation directly. Here, how 'list' item animates is based on the individual transition component. It means, the "TransitionGroup" component can have different animation within a component.

Thank You!