logic such as authentication of the user.

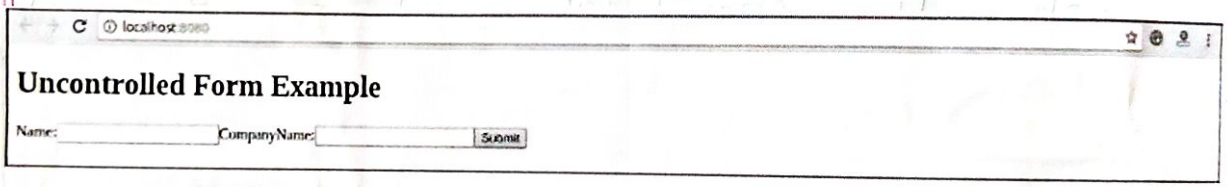
Creating form :-
React offers a stateful, reactive approch to build a form. The component rather than Dom usually handles the React form.

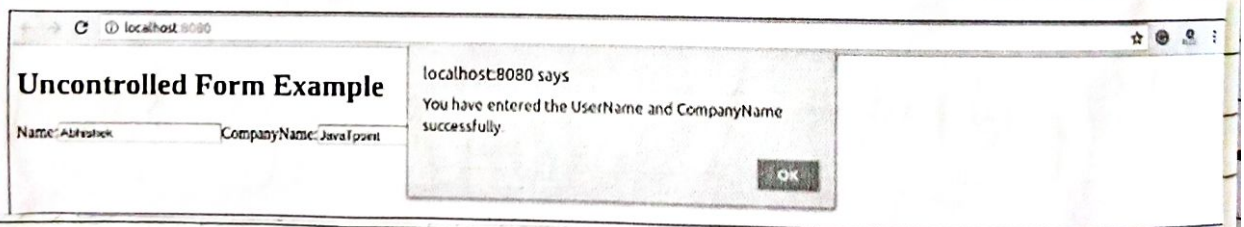
① Uncontrolled component.
② Controlled component.

① Uncontrolled Component :-
The uncontrolled component input is similar to the traditional HTML form input. The Dom itself handles the form data. The HTML element maintain their own state that will be updated when the input values changes. To write an uncontrolled component.

```
impoit React, {component} from 'react';
class App extends React.Component {
    constructor (props)
        super(props);
    this.updateSubmit = this.updateSubmit.
                        bind (this);
    this.input = React.createRef ();
    }
updateSubmit (event) {
 alert ("You have entered the username and
  CompanyName successfully");
  event.preventDefault();
 }
render () {
```

```
return(
 <form onSubmit={this.updateSubmit}>
   <h1> Uncontrolled Form </h1>
   <label> Name:
     <input type = "text" ref = {this.input
   </label>
   <label>
   Company Name:
   <input type = "text" ref={this.input1/>
   </label>
   </form>
 );
}}
export default App;
```





Controlled Component.

Controlled component have functions that govern the data passing into them on every onChangeevent; rather than grabbing the data only ones.

```
impost React {Component } from 'react';
class App extends React.Component {
    constructor (props) {
      super (props);
     this.state = {value :" "};
     this.handleChange = this.handleChange.bind
                                    (this);
    this.handleSubmit= this.handleSubmit.bind (this);
}
  handleChange (event) {
    this.setState ({value: event.target.value})}
  }
  handleSubmit (event) {
    alert ("You have submitted the input Success-
        fully:' + this.state.value);
     event.preventDefault ();
  }
render () {
  return (
  <form onSubmit = {this.handleSubmit} >
   <h1> Controlled Form </h1>
  <label>
  Name:
<input type ="text" value = {this.state.value}
     onChange = {this.handlechange} />
   </label>
  <input type = "submit" value ="Submit" />
  </form>
  );
  }
}
export Default App;
```

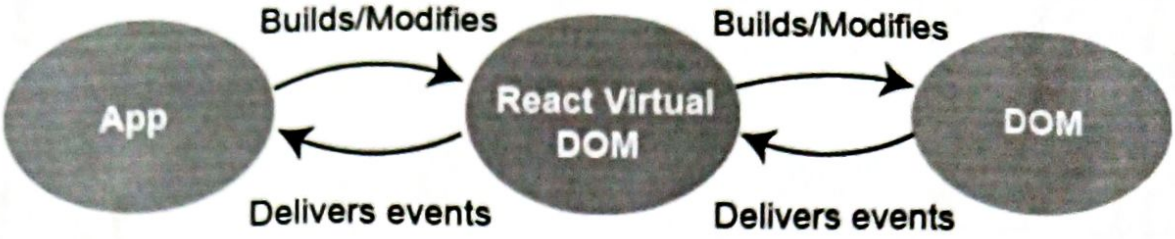
Controlled Componant Vs Uncontrolled Component.

Controlled	Uncontrolled.
It does not maintain internal state	It maintains internal state.
Data is controlled by parent component	Data is controlled by a Dom itself
It accept the current value as a prop	It uses a ref for their current values.
It allows validation control	It does not allow validation control.
It has better control over the form elements and data	It has limited control over the form and elements.

React 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 events is a a cross browser wrapper of the browser native event.

① React events are named as camelCase insted of lowercase.



② A function is passed as a event handler instead of a string.

For declaration in plain HTML :-

```
<button onclick = "showMessage()">
    Hellow JavaTpoint
</button>
```

Event declaration in React :-

```
<button onClick = {showMessage}>
    Hellow JavaTpoint
</button>
```

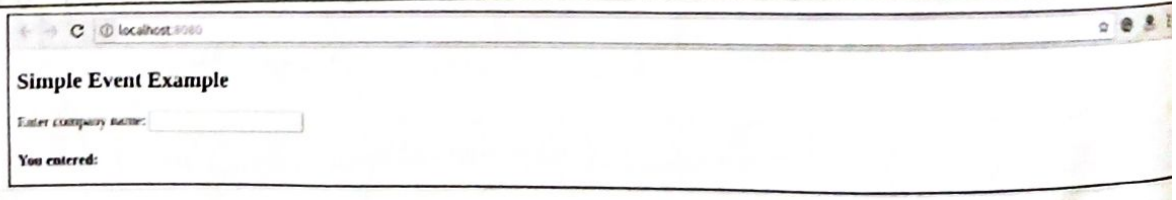
③ In react, we cannot return false to prevent the default behaviour we must call preventDefault event explicitly to prevent the

Example :-

```
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> Simple Event </h2>
   <label html For='name"> Enter Company
              Name </label>
  <input type="text" id="companyName"
     onchange = {this.changeText bind (this
   <h4> You entered ; {this.state.company
    Name } </h4>
   </div>
  );
  }
 }

 export default App;
```



React Conditional Rendering:-

There is more than one way to do conditional rendering in React.

- if

It is the easiest way to have conditional rendering in React in the render method.

```
function UserLogin (props){
    return <h1> Welcome Back </h1>;
}
function GuestLogin(Props ) {
    return <h1> Please Sign Up </h1>
}
function SignUp (props){
  const isLoggedIn = props. isLoggedIn;
  if (isLoggedIn ){
      return <User Login />;
  }
 return <GuestLogin />;
  }
ReactDom. render {
  <signUp isLoggedIn = {false} />
   document. getElementById ('root')
);
```

Logical && operator.

The operator is used to checking the condition.

```
import React from 'react';
import ReactDOM from 'react-Dom';
function Example ()
```

```
return (<div>
 { (10>5) && alert ('This alert will be
        Shown!")
 }
 </div>
);
}
```

Ternary Operator :-

```
render ( ) {
 const isLoggedIn = this.state.isLoggedIn;
 return {
  <div>
Welcome {isLogged In ? 'Back ; Please login}
 </div>
 };
}
```

Switch Case Operator

```
functional NotificationMsg ({text }) {
  Switch (text) {
   case 'Hi all';
       return <Message: text ={text }/>;
   case "Hellow Java Tpoint ;
       return <Message text ={text}/>;
   default
    return null;
  }
 }
```

Conditional Rendering with enums

An enum is a great way to have a multiple Conditional rendering. It is more readable as compared to switch case operator.

```
function NotificationMsg({text, state}) {
  return(
    <div>
    {{
      info: <Message text={text} />
          warning : <Message text={text} />,
      } [state]}
    </div>
  );
}
```

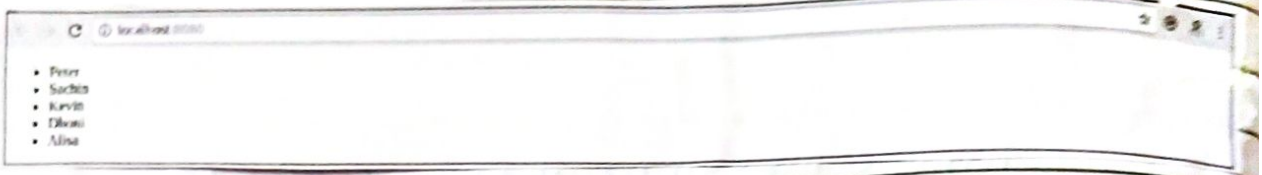
React Lists:-

Lists are used to display in an ordered forma and mainly used to display menus on websit The map() function is used to traversing the list. Include the new list <ul> elements and render it to dom.

```
import React from 'react';
import ReactDOM from 'react-dom';
const myList = ['Peter', 'sachin', 'kelvin', 'Dhoni
  'Alisa'];
Const myListItems = myList.map((myList)=> {
    return <li> {mylist} </li>
  });
ReactDOM.render (
    <ul> {listItems} </ul>
```

```
document.getElementById('app')
);
export default App;
```

Output:



React Keys :-

A key is unique identifier. In React, It is used to identify which items have changed, updated or deleted from the lists. It is useful when we dynamically created component or when the user alert the Lists. It also helps to determine which component in a collection need to be rendered instead of re-rendering the entire set of components every time.

```
const stringList = ['Peter', 'Sachin', 'Kevin',
        'Dhoni', 'Alisa'];
const updatedLists = stringLists.map((strList
    <li key={strList.id}> {strList} </li>
  });
```

If there are no stable IDs for rendered items, you can assign the item index as a key of the lists. It can be shown in the below,

```
const stringLists = ['Peter', 'Sachin', 'Kevin', 'Dhoni',
    'Alisa'];
const updatedLists = stringLists.map((strList, index)=>
  {
    <li key={index} > {strList} </li>
  });
```

React Refs:-

Refs is the shorthand used for referances in React. It is an attribute which makes it possible to store a referance to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it.

When to Use Refs:-

- When we need DOM measurements such as managing focus, text selection, or media playback.
- It is used triggring imperative animations.
- When integrating with third-party DOM libraries.
- It can also use as in callbacks.

When to not use Refs:-

- It is use should be avoided for identify anything that can be done declaratively. insted of using open() and close() methods on a Dialog component, you need to pass an isOpen prop to it.
- You should have to avoid overuse of the Refs.

How to create Refs.

In React. Refs can be created by using React.createRef(). It can be assigned to React Element via the ref attribute. It is commonly assigned to an instance property when a component is created and them can be referenced throughout the component.

```
class MyComponent extends React.Component {
  constructor(props) {
  super(props);
  this.callRef = React.createRef();
  }
  render() {
    return <div ref={this.callRef} />
  } }
```

How to access Ref's

When a ref is passed to an element inside render method. a referance to the node can be accessed via the current attribute of the ref.

```
const node = this.callRef.current;
```

Ref current Properties

- When the ref attribute is used in HTML element, the ref created with React.createRef() receives, the underlying Dom element as its current property.

- If the ref attribute is used on a custom class component then ref object receives the mounted instance of the component as its current property.
- The ref attribute cannot be used to on function component because they dont have instances.

Callback Refs :-

There is another way to use refs that is called 'callback refs' and it gives more control when the refs are set and unset. Insted of creating refs by passing a callback function to the ref attribute of a component.

```
<input type ="text" ref ={element => this.call-
  RefInput =element}/ >
```

The callback function is used to store a referance to the Dom node in an instance property and can be accessed elsewhere.

```
this.callRefInput.value.
```

React with useRef ()

It is introduced in React 16.7 and above version It helps to get access the Dom node element and the we can interact with the Dom node or element such as focusing the element or accessing the input element value. It returns the ref object whose .current property initiallized to the passed argument.

```
const refContainer = useRef(initialValue);

function useRefExample (){
   const inputRef = useRef (null);
   const onButtonClick =()=>{
   inputRef.current.focus ();
  };
  return (
   <>
<input ref={inputRef} type ="text"/>
<button onClick ={onButtonClick}> Submit
                                  </button>

   </>
  );
}
```

React Fragments:-

The render method can return single elements or multiple element. The render method will only render a single node inside at a time.

```
class App extends React.Component {
   render (){
      return (
   <div>
   <h2> Hellow World </h2>
   <P> Welcome to the JavaTpoint </P>
   </div>
   );
  }
}
```

Syntax :-

```
<React Fragment>
  <h2> Child 1 </h2>
  <p> child 2 </p>
    .....
</React.Fragment>
```

Why we use Fragments?

① It makes the execution of code faster as compared to the div tag.

② It take less memory.

Fragments Short Syntax:-

There is also another shorthand exists for declaring fragments for the above method. It looks like empty tag in which we can use '<>' and insted of the 'React.Fragment'

```
Class Colums extend React.Component {
  render(){
    return (
    <>
   <h2> Hello World! </h2>
    <p> Welcome to the JavaTPoint </p>
    </>
    );
  }
}
```

Key Fragments:-

The shorthand syntax does not accept key