Subject: React JS      Assignment: 3

↳ Application Architecture:

↳ React applications typically follows a component-based architecture, where the UI is broken down into small, reusable components that manage their own state and behavior.

↳ This modular approach allows for efficient data handling and updationg in response to changes in application state, making it suitable for building dynamic, single-page applications.

ex: import react from 'react';
import ReactDom from 'react-dom';

ReactDom.render(<App/>);

↳ class components.

↳ class components are React components built using ES6 classes.
↳ They extend React.Component and have a render() method to define the UI.

ex: import React {component} from 'react';
class MyclassComponent extends component
{

constructor(props) {
super(props);

```
this.state = {count: 0};
}
increment = () =>
{
this.setState({ count: this.state.count + 1});
};
renderer()
{
 return
  (
   <div><p> count :{this.state.count}</p>
   </div>
```

↳ Functional components:

↳ Functional components are simpler and are just Javascript functions that return JSX.

↳ with the introduction of hooks, they can also manage state and side effects.

~~in~~ import {useState} from 'react';

```
function    count

{ p
    const [u , Setu] = useState(0);

~~return~~
  ~~(~~      const  co c = () =>
  ~~</>~~
            {
              Setu(u++)
              alert(u)

          });

return
(
 <>
  <div>
  <button   onclick ={co} > click </button>

  </div>

 </>

();

export  default  ~~co~~  count ;
```

↳ Nested components:

↳ React allows components to be nested within other components.
↳ Enabling a ~~bit~~ hierarchical UI structure.

ex:
```
import React from 'react';

function
        ChildComponent ()
    {
        return <p> child </p>
  }


function
        ParentComponent ()
    {

     return
     (
        <div>
          ~~<p>~~ <p> parent </p>
          </div>
     );
export default ParentComponent;
```

DOMS Page No.
Date / /

↳ conditional and Looping constructs.

↳ conditional renderring allows components to display UI based on conditions, while loops like map () allows rendering lists.

Ex:

```
import {ul} from 'list.js';

function List() {
return (
{/ <>

  {ul.map((item, index)=>

   {
    return
     (<h1> item.~~ha~~name </h1>)

  })};

}

export default List.
```

↳ State:

↳ State is an object in a components that holds data that affects the component's rendering.
↳ It is managed using useState in functional components or this.setState in class components.

↳ Ex:

```
function St()

{

const[mn, setmn] = useState("")

~~const = () =>~~

{  ~~const ()~~ const mp = () =>

          {

          setmn("Hello")
          console.log(mn)

          }

return
(
 <>
  <button onclick={mp}> click </button>
</>
);
export default St.
```

Page No.
Date / /

→ props:

→ ~~JH~~ props (short for properties") are used to pass data from a parent component to a child component: ~~ps~~
→ props are read-only.

(x) import react from 'react';

```
function
    Greeting ({name})

  {
      return <p> Hello, {name} !</p>;

}

    function parent ()

      {
          return <Greeting name ="Rufus" />;

      }

export default parent;
```

Scanned with OKEN Scanner