HAD-2

Practical 1:

Aim: Install create-react-app and create a new react project.

Theory:

Npx create-react-app command:

Create React App is a comfortable environment for learning React, and is the best way to start building a new single page-application in React.

Npm start command:

This runs an arbitrary command specified in the package's “start” property of its “scripts” object.

Steps:

- Type npx create-react-app hello-world in the visual studio console or the command prompt.
- After the npx create-app command execution change the directory to the newly created hello-world directory in the visual studio console or the command prompt.
- Then type npm start command in the visual studio console or the command prompt.
- Finish.

Code:

src > Table.js > Table_had > render

```
1  import React, { Component } from 'react';
2  class Table_had extends Component {
3      render() {
4          return (
5              <table>
6                  <thead>
7                      <tr>
8                          <th>Name</th>
9                          <th>Roll-no</th>
10                         <th>Class</th>
11                     </tr>
12                 </thead>
13                 <tbody>
14                     <tr>
15                         <td>satyam</td>
16                         <td>   86</td>
17                         <td>syit</td>
18                     </tr>
19                     <tr>
20                         <td>ramesh</td>
21                         <td>   75</td>
22                         <td>syit</td>
23                     </tr>
24                 </tbody>
25             </table>
26         )
27     }
28 }
29
30 const Table = () => {
31     return (
32         <h1>this is function</h1>
33     );
34 }
35
36 export default Table_had;
37 export {
38     Table
39 }
```

Output:

this is function

Name	Roll-no	Class
satyam	86	syit
ramesh	75	syit

Practical 2:

Aim: Create JSX expressions with different javascript expression, apply css via className and styles, use conditionals.

Theory:

- It is called JSX, and it is a syntax extension to JavaScript.
- We recommend using it with React to describe what the UI should look like.
- JSX may remind you of a template language, but it comes with the full power of JavaScript. JSX produces React "element".
- After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

Code:

src > JS index.js > ...

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
//import App from './App';
import reportWebVitals from './reportWebVitals';

class Dropdown extends React.Component {
  constructor(props) {
    super(props);
    this.state = { greetings: '' };
  }
  ChangeOnSelect = (event) => {
    this.setState({ greetings: event.target.value });
  }
  render() {
    return (
      <form>
        <h1>{this.state.greetings} World</h1>

        <select
          onChange={this.ChangeOnSelect}>
          <option value="Hello">Hello</option>
          <option value="Bonjour">Bonjour</option>
          <option value="Namaste">Namaste</option>
          <option value="Hola">Hola</option>
        </select>

      </form>
    );
  }
}

ReactDOM.render(<Dropdown />, document.getElementById('root'));

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

output:

Practical 3:

Aim: Create a stateful component and stateless component. Pass data from parent component to child component using props. Implement child to parent communication using callbacks.

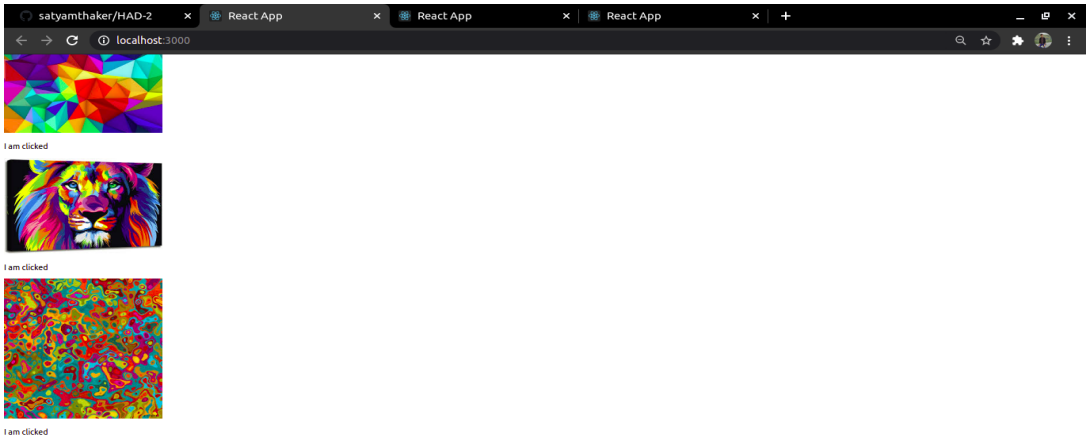
Theory:

- When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object "props".
- Components can refer to other components in their output.
- This lets us use the same component abstraction for any level of detail.
- A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

Code:

```
src > JS images.js > Images > render
1  import React , {Fragment} from 'react'
2
3  class Images extends React.Component{
4
5      constructor(props){
6          super(props);
7          this.state ={value : this.props.value[1],}
8      }
9
10
11     render() {
12         return (
13             <Fragment>
14                 <img src={this.props.value[0]}  width='300' onClick={() => this.setState({value: 'I am clicked'})} ></img>
15                 <p>{this.state.value}</p>
16             </Fragment>
17         );
18     }
19 }
20
21 class Card extends React.Component {
22     renderButton(value){
23         return (
24             <Images value={value} />
25         );
26     }
27
28     render() {
29         return(
30             <Fragment>
31                 {this.renderButton(["
                   2wCEAAkGBxITEhUSEhMWFRUWFRUXFRcVFRUXFxUVFxUWFxUVFxYYHSggGBolGxUVITEhJSkrLi4uFx8zODMtNygtLisBCgoKDg0OGxAQG i0lIC
                   S0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLS0tLf/AABEIAKgBKwMBEQACEQEDEQH/xAAbAAACAwEBAQAAAAAAAAAAAAAEBQI
                   EAD0QAAECAwUECAMGBgMBAAAAAAEAAgMEEQUSITEBUWFxkQYiMoGhscHRE0JSEWJyguHwECNDkqLxBzQv4y/EABsBAAEFAQEAAAAAAAAAAAAA
```

output:



Practical 4:

Aim: Create component which renders lists iteratively using map function of array.

Theory:

- With react, the approach is to use native JavaScript for iteration.
- So in React you can use the map () function to iterate over an array and create a list.
- We reach for the built-in map function in JavaScript.
- We can use the map function to map each item in the array to a react element.
- We pass an arrow function as an argument to the map function.

Code:

sorting.js | index.js

src > sorting.js > List_value > constructor > list_array

```
1  import React from 'react';
2
3  class List_value extends React.Component {
4
5      constructor(props) {
6          super(props);
7          this.state = {
8              list_array : ["D", "C", "A", "B"],
9          };
10     }
11
12
13
14
15     render() {
16         return (
17             <React.Fragment>
18                 <ul>
19                     {this.state.list_array.map((list_item) => <li key={list_item} value={list_item} > {list_item} </li>)}
20
21                 </ul>
22                 <button onClick={() =>  this.setState({list_array : this.state.list_array.sort()}) } >
23                     click to sort
24                 </button>
25
26             </React.Fragment>
27         );
28     }
29 }
30
31 export default List_value;
```

output:

- A
- B
- C
- D

click to sort

Practical 5:

Aim: Create a stateful component and implement lifecycle methods. Implement try catch mechanism using error boundaries.

Theory:

- Error boundaries are React components that catch Javascript errors anywhere in their child component tree, log those errors and display a Fallback UI, instead of the component tree that crashed.
- Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.
- Error boundaries can only catch errors in the components below them in the tree.
- An error boundary can't catch an error within itself.

Code:

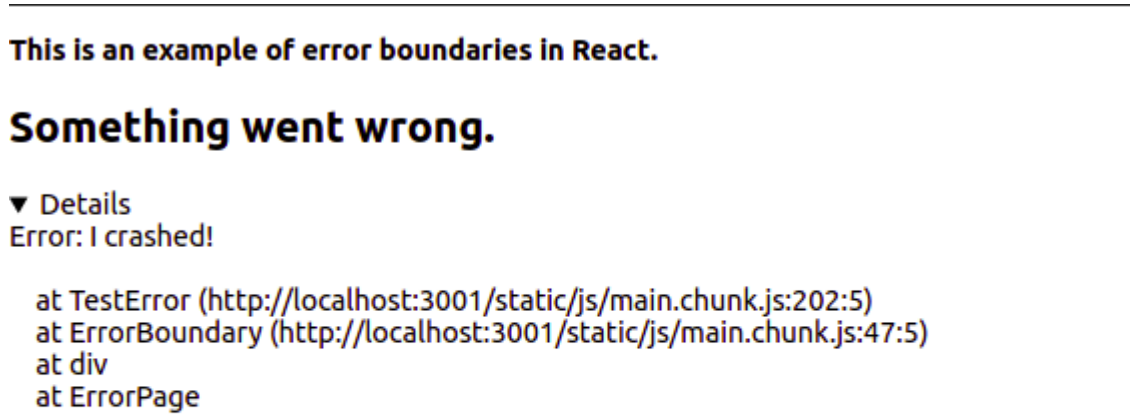
src > JS errorboundry.js > default

```
import React from 'react';

class errorboundry extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            is_error: false
        }
    }
    static getDerivedStateFromError() { return {is_error: true}}

    render() {
        if (this.state.is_error) {
            return <h1>Encountered an error</h1>;
        }
```

src > JS TestError.js > ...

```
import React from "react";
import ErrorBoundary from "./ErrorBoundary";

class TestError extends React.Component {
    constructor(props) {
        super(props);
        this.state = {shouldCrash: false};
    }

    makeItCrash() {
        this.setState({shouldCrash: true});
    }

    render() {
        if (this.state.shouldCrash) {
            // Simulate a JS error
            throw new Error('I crashed!');
        }
        return <button type="button" onClick={this.makeItCrash.bind(this)}>Crash it</button>
    }
}

function ErrorPage() {
    return (
        <div>
            <p>
                <b>
                    This is an example of error boundaries in React.
                </b>
            </p>
            <ErrorBoundary>
                <TestError/>
            </ErrorBoundary>
        </div>
    );
}


export default ErrorPage;
```

output:


This is an example of error boundaries in React.
Something went wrong.
▼ Details
Error: I crashed!
at TestError (http://localhost:3001/static/js/main.chunk.js:202:5)
at ErrorBoundary (http://localhost:3001/static/js/main.chunk.js:47:5)
at div
at ErrorPage

Practical 6:

Aim: Create a component that uses different form controls.

Theory:

- This form has the default HTML form behavior of browsing to a new page when the user submits the form.
- If you want this behavior in React, it just works.
- But in most cases, it's convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form.
- The standard way to achieve this is with a technique called "controlled components".
- React component that renders a form also controls what happens in that form on subsequent user input.
- An input form element whose value is controlled by React in this way is called a "controlled component".

Code:

```
src > JS Form.js > Form > state
  1  import React from 'react';
  2  import axios from 'axios';
  3
  4  class Form extends React.Component {
  5      state = {user: 'satyamthaker', response: '', showResponse: false}
  6
  7      async submission(event) {
  8          event.preventDefault();
  9          const apiResponse = await axios.get(`https://api.github.com/users/${this.state.user}`)
 10          this.setState({response: JSON.stringify(apiResponse.data), showResponse: true})
 11      }
 12
 13      render() {
 14          return (
 15              <div>
 16                  <form onSubmit={this.submission.bind(this)}>
 17                      <input type="text" placeholder="Enter your name" value={this.state.user}
 18                             onChange={event => this.setState({user: event.target.value})}
 19                      />
 20                      <button type="submit">Submit</button>
 21                  </form>
 22                  <h2>{this.state.showResponse ? 'Response data:' : ''}</h2>
 23                  <p>{this.state.response}</p>
 24              </div>
 25          );
 26      }
 27  }
 28
 29  export default Form;
 30
```

output:

satyamthaker Submit

Response data:

{"login":"satyamthaker","id":60962597,"node_id":"MDQ6VXNlcjYwOTYyNTk3","avatar_url":"https://avatars.githubusercontent.com/u/60962597?v=4","gravatar_id":"","url":"https://api.github.com/users/satyamthaker","html_url":"https://github.com/satyamthaker","followers_url":"https://api.github.com/users/satyamthaker/followers","following_url":"https://api.github.com/users/satyamthaker/following{/other_user}","gists_url":"https://api.github.com/users/satyamthaker/gists{/gist_id}","starred_url":"https://api.github.com/users/satyamthaker/starred{/owner}{/repo}","subscriptions_url":"https://api.github.com/users/satyamthaker/subscriptions","organizations_url":"https://api.github.com/users/satyamthaker/orgs","repos_url":"https://api.github.com/users/satyamthaker/repos","events_url":"https://api.github.com/users/satyamthaker/events{/privacy}","received_events_url":"https://api.github.com/users/satyamthaker/received_events","type":"User","site_admin":false,"name":null,"company":null,"blog":"","location":null,"email":null,"hireable":null,"bio":null,"twitter_username":null,"public_repos":30,"public_gists":0,"followers":0,"following":0,"created_at":"2020-02-12T08:58:04Z","updated_at":"2021-01-15T06:28:53Z"}

Practical 7:

Aim: Create components that get applied with multiple themed styles using context to store theme info globally and apply to all components.

Theory:

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application.
- Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

Code:

```
src > App.js > App
import logo from './logo.svg';
import './App.css';
import {ThemeContext} from './theme-context'
import React from 'react';

function App() {

  const {theme, toggle, dark } = React.useContext(ThemeContext)


  return (
    <div className="App">
      <header className="App-header" style = {{backgroundColor: theme.backgroundColor, color: theme.color}}>
      <button
      type = "button"
      onClick = {toggle}
      style = {{backgroundColor: theme.backgroundColor, color: theme.color}}
      >
        Click here to toggle to {!dark ? 'dark': 'light'} theme
        </button>


        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

output:

Practical 8:

Aim: Create a functional component that uses the ability of state and life cycle features

Theory:

- Lifecycle methods use a form of hooking that allows the execution of code at set points during a component's lifetime.
- ShouldComponentUpdate allows the developer to prevent unnecessary re-rendering of a component by returning false if a render is not required.
- ComponentDidMount is called once the component has "mounted" (the component has been created in the user interface, often by associating it with a DOM node). This is commonly used to trigger data loading from a remote source via an API.
- ComponentWillUnmount is called immediately before the component is torn down or "unmounted". This is commonly used to clear resource-demanding dependencies to the component that will not simply be removed with the unmounting of the component (e.g., removing any setInterval() instances that are related to the component, or an "eventListener" set on the "document" because of the presence of the component)
- render is the most important lifecycle method and the only required one in any component. It is usually called every time the component's state is updated, which should be reflected in the user interface.

Code:

.

```
src > lifecyc.js > LifeCycles > render > state.userdata.map() callback
import React, { Component } from 'react';
import axios from 'axios'

class LifeCycles extends Component {
    constructor(props){
        super(props);
        this.state = {
            userdata : []
        }
    }

    componentDidMount() {
        axios.get('https://jsonplaceholder.typicode.com/users')
            .then(res => {
                this.setState({
                    userdata: res.data
                });
                console.log(res.data)
            })
    };

    render() {
        return (
            <div>
                <ul>
                    {this.state.userdata.map((item) => (
                        <li>
                            <h2>{item.username}</h2>
                            <h3>{item.name}</h3>
                            <h3>{item.email}</h3>
                            <h2>Address: </h2>
                            <h3>{item.address.suit} {item.address.city} {item.address.zipcode}</h3>

                            <small> call : {item.phone} web : {item.website} </small>
                            <h2>Company Details</h2>
                            <h3>{item.company.name}</h3>
                            <i>{item.company.catchPhrase}</i><br/>
                            <strong>{item.company.bs}</strong>
                            <hr></hr>
                        </li>


                    ))}
                </ul>
            </div>
        );
    }
}

export default LifeCycles;
```

output:

- **Bret**

  Leanne Graham

  Sincere@april.biz

  Address:

  Gwenborough 92998-3874

  call : 1-770-736-8031 x56442 web : hildegard.org

  Company Details

  Romaguera-Crona

  Multi-layered client-server neural-net
  harness real-time e-markets

- **Antonette**

  Ervin Howell

  Shanna@melissa.tv

  Address:

  Wisokyburgh 90566-7771

  call : 010-692-6593 x09125 web : anastasia.net

  Company Details

  Deckow-Crist

  Proactive didactic contingency
  synergize scalable supply-chains

- **Samantha**

  Clementine Bauch