

```

1  /*
2      ===== React Class Component =====
3
4      Clas-based Components: An Alternative To Functions
5
6  */
7  // Function base Component
8  function Product(props) {
9      return <h2>A product!</h2>
10 }
11
12 /*
13     Components are regular JavaScript function which return
14     renderable results (typically JSX)
15 */
16
17 // Class Base Component
18
19 class Product extends Component {
20     render() {
21         return <h2>AProduct!</h2>
22     }
23 }
24
25
26 /*
27     Components can also be defined as JS classes where a render() method
28     defines the to-be-rendered output
29 */
30
31
32 // ***** convert function to to class base component *****
33
34 // function base component ***** START *****
35
36 import React, { useEffect } from 'react';
37 import Styles from './User.module.css';
38
39 const User = (props) => {
40
41     return (
42         <li className={Styles.user}>{props.name}</li>
43     )
44 }
45 export default User;
46
47 // function base component ***** END *****
48
49
50
51 // class base Component ***** START *****
52
53 import Styles from './User.module.css';
54
55 // we need to import component from react and
56 import React, { Component } from 'react';
57
58 // and extends that Component

```

```

58 // and extends that Component
59 class User extends Component {
60   constructor() {
61     // initialize something first
62   }
63
64   // React find the render to to display on screen by call React
65   render() {
66     // This render method return the JSX
67     return (
68       // props get the class base automatically coz we extends Component from React by (this) we can use
69       <li className={Styles.user}>{this.props.name}</li>
70     );
71   }
72 }
73
74 // class base Component ***** END *****
75
76
77 // ***** Another Convert function Base to Class base Component*****
78
79 // function base component ***** START *****
80
81 import { useState } from 'react';
82 import User from './User';
83
84 import classes from './Users.module.css';
85
86 const DUMMY_USERS = [
87   { id: 'u1', name: 'Max' },
88   { id: 'u2', name: 'Manuel' },
89   { id: 'u3', name: 'Julie' },
90 ];
91
92 const Users = () => {
93   const [showUsers, setShowUsers] = useState(true);
94
95   const toggleUsersHandler = () => {
96     setShowUsers((curState) => !curState);
97   };
98
99   const usersList = (
100     <ul>
101       {DUMMY_USERS.map((user) => (
102         <User key={user.id} name={user.name} />
103       ))}
104     </ul>
105   );
106
107   return (
108     <div className={classes.users}>
109       <button onClick={toggleUsersHandler}>
110         {showUsers ? 'Hide' : 'Show'} Users
111       </button>
112       {showUsers && usersList}
113     </div>
114   );
115 };
116

```

```

117 export default Users;
118
119
120
121 // function base component ***** END *****
122
123
124 // class base Component ***** START *****
125 import { Component } from 'react';
126 import User from './User';
127
128 import classes from './Users.module.css';
129 import { thisStringValue } from 'es-abstract/es2019';
130
131 const DUMMY_USERS = [
132   { id: 'u1', name: 'Max' },
133   { id: 'u2', name: 'Manuel' },
134   { id: 'u3', name: 'Julie' },
135 ];
136
137 class Users extends Component {
138
139   // Define *****state ***** inside the class
140   // two things is needed
141   // 1) Define and 2) update
142   constructor() {
143     super();
144     // in class base component state Always object {} and only one state in class base
145     // one important things is that when we update the State then the object
146     // will Merge on overwrite
147     this.state = {
148       showUsers: true,
149       // moreState: "test",
150       // nested: {},
151       // data: []
152       // All are working
153     };
154   }
155
156   toggleUsersHandler() {
157     // method define like inthis way outside the render method
158     // this.state.showUsers = false // NOT !
159     // but
160     // this.setState({showUsers: false}); work it another way by class functions
161     this.setState((curState) => {
162       return { showUsers: !curState.showUsers }
163     });
164   }
165
166   render() {
167
168     // add userList inside the Rander Method
169     const userList = (
170       <ul>
171         {DUMMY_USERS.map((user) => (
172           <User key={user.id} name={user.name} />
173         ))}
174       </ul>
175     );

```

```

175             
176         </ul>
177     );
178
179     return (
180         <div className={classes.users}>
181             <button onClick={this.toggleUsersHandler.bind(this)}>
182                 {this.state.showUsers ? 'Hide' : 'Show'} Users
183             </button>
184             {this.sate.showUsers && usersList}
185         </div>
186     );
187 }
188 }
189
190 // class Base Component ***** END *****
191
192
193 // ===== Class-based Component Lifecycle =====
194 /*
195     1) Side-effects in functional Components: useEffect()
196     2) Class-based Components can't use React Hooks!
197 */
198
199 /*
200     1) componentDidMount()
201         *** Called once component mounted (was evaluated & rendered)
202         *** useEffect(..., []); equivalent to without dependencies
203     2) componentDidUpdate()
204         **** Callded once component updated (was evaluated & rendered)
205         **** useEffect(..., [dependencies]);
206     3) componentWillUpdate()
207         **** Called right before component is unmounted (removed from DOM)
208         **** useEffect(()=> {return ()=> {...}}, []); equivalent
209
210 */
211
212
213 // Functions Base *****
214 DUMMY_USERS = [
215
216 ]
217 const UserFinder = () => {
218
219     useEffect(() => {
220         setFileredUsers(
221
222             DUMMY_USERS.filter((user) => user.name.includes(searchTerm))
223         );
224     }, [searchTerm]);
225
226
227     return (
228         <p>Mehedi</p>
229     );
230 }
231
232
233 // Class Base *****

```

```

234 import React, { Component } from 'react';
235
236 class UserFinder extends Component {
237     constructor() {
238         super();
239         this.state = {
240             filteredUsers: DUMMY_USERS,
241             searchTerm: "",
242         }
243     }
244
245     // with out chacking previous state it gose like loop when state update
246     // this function call again and again that is why we need
247     componentDidUpdate(prevProps, prevState) {
248
249         if (prevState.searchTerm !== this.state.searchTerm) {
250             this.setState({
251                 filteredUsers: DUMMY_USERS.filter((user) =>
252                     user.name.includes(searchTerm)
253                 )
254             });
255         }
256     }
257
258     searchChangerHandeler(event) {
259         this.setState({ searchTerm: event.target.value }); // marge that object
260     }
261
262     render() {
263         return (
264             <React.Fragment>
265                 <div className={Styles.finder}>
266                     <input type="search" onChange={this.searchChangeHandler.bind(this)} />
267                 </div>
268                 <Users users={this.state.filteredUsers} />
269
270             </React.Fragment>
271
272         );
273     }
274 }
275
276
277 // ===== context inside the classbase component =====
278
279 // 1) create context data file which we want to share
280
281 // ***** two way to sue context in class
282 /*
283     1) context provider and consumer BOTH class and function use this way
284     2) static contextType = useContext; like this way inside the class base component
285
286 */
287
288 // 1 one
289
290 return (
291     <UserSContext.provider value={usersContext}>
292         <UserFinder />
293     </UserSContext.provider>

```

```

292     </UserSContext.provider>
293   </UserSContext.provider>
294 )
295
296 return (
297   <UserSContext.Consumer>
298     .....
299     .....
300     .....
301   </UserSContext.Consumer>
302 );
303
304
305 // 2 way
306
307 class UserFinder extends Component {
308   static contextType = userContext;
309
310   // use like this
311   componentDidMount() {
312     this.setState({ filteredUser: this.context.users })
313   }
314 }
315
316
317
318 // ===== Error =====
319
320 // this is how we can generating error by won
321 // when this application run that time if no users in this list that time it show an error
322 componentDidMount(){
323   if (this.props.users.length === 0) {
324     throw new Error("No Users provided!");
325   }
326 }
327
328 // But we can not use error like this way...Another solution to handel the erron...=> Error Boundaries
329
330 try {
331   somCodeWhichMightFail();
332 } catch (err) {
333   // handele error
334 }
335
336 // it will work in javascript but not in JSX
337 // That's why we can use nother way...Error Boundaries
338
339
340 // ***** Error Boundary *****
341 import { Component } from 'react';
342
343 class ErrorBoundary extends Component {
344
345   componentDidMount() {
346     // logic here wich time error we want to handele
347     super();
348     this.state = {hasError : false};
349   }
350

```

```

351     componentDidCatch(error){
352
353         // we can send error to server and analyzing the error message from here
354
355         console.error(error);
356         this.setState({hasError : true});
357     }
358
359     render() {
360         if(this.state.hasError){
361             return <p>Something went wrong!</p>
362         }
363         return this.props.children;
364     }
365 }
366 export default ErrorBoundary;
367
368
369 // ***** which Component generating error that will be wrap *****8
370
371 class App extends Component {
372
373     render() {
374         return (
375             <React.Fragment>
376                 <p>Mehedi</p>
377
378                 <ErrorBoundary >
379                     // this child component will be generating errors wrap this...
380                     <Users users={this.state.filteredUsers} />
381                 </ErrorBoundary>
382
383             </React.Fragment>
384
385         );
386     }
387
388 }

```