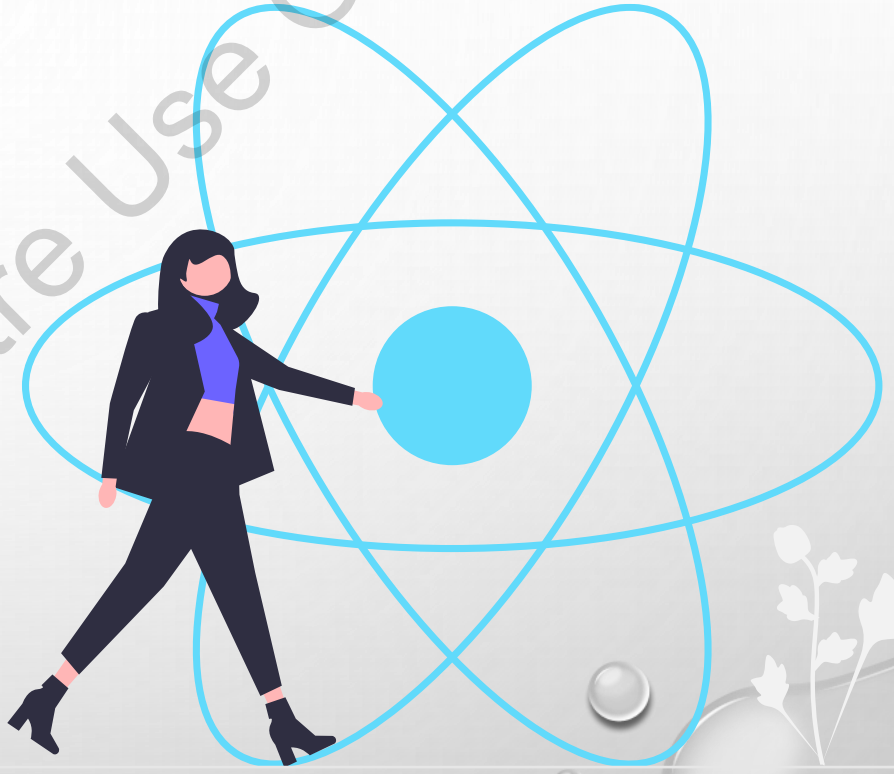
SESSION 01

INTRODUCTION TO REACTJS

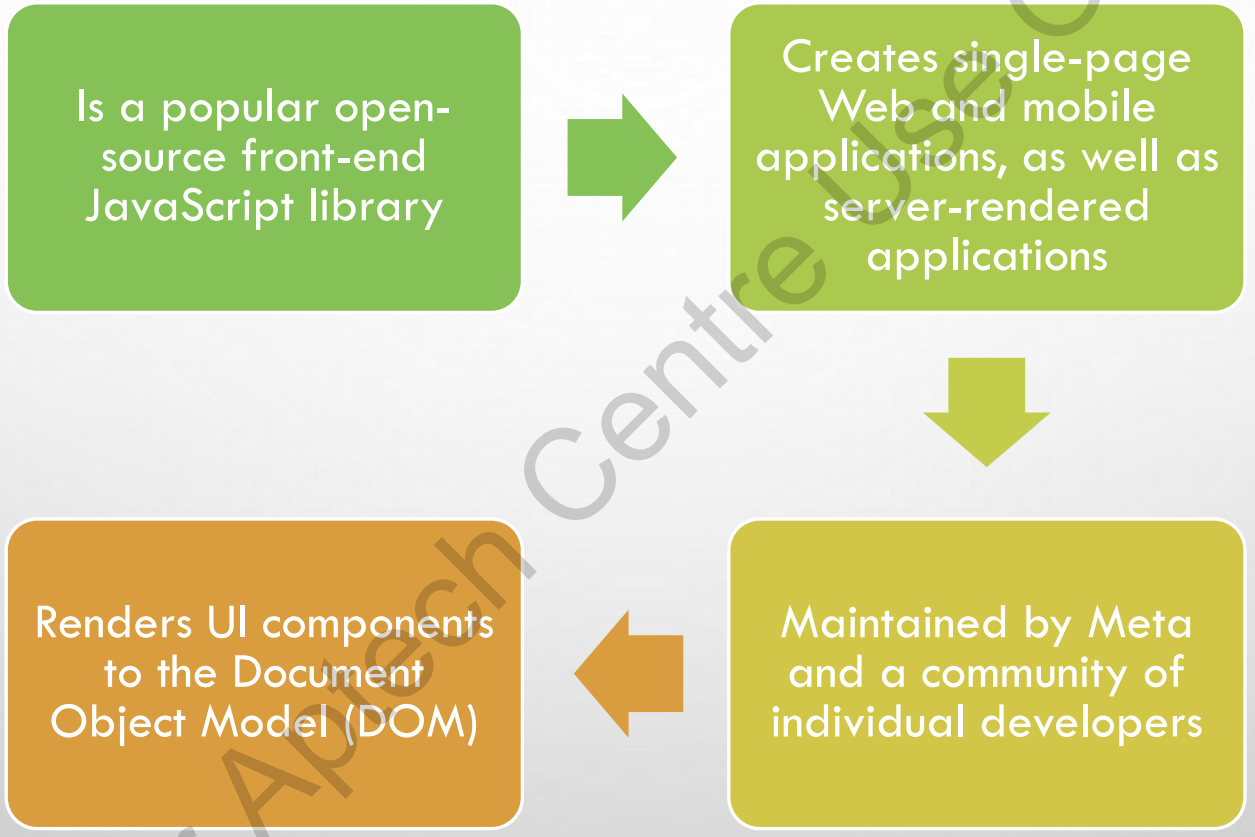


OBJECTIVES

In this session, the topics discussed are as follows:

- Explain ReactJS
- Explain the ReactJS ecosystem
- Explain the core concepts of ReactJS
- Illustrate the setup of the ReactJS development environment
- Illustrate installation of the create-react-app library

WHAT IS REACTJS?



WHY USE REACTJS?

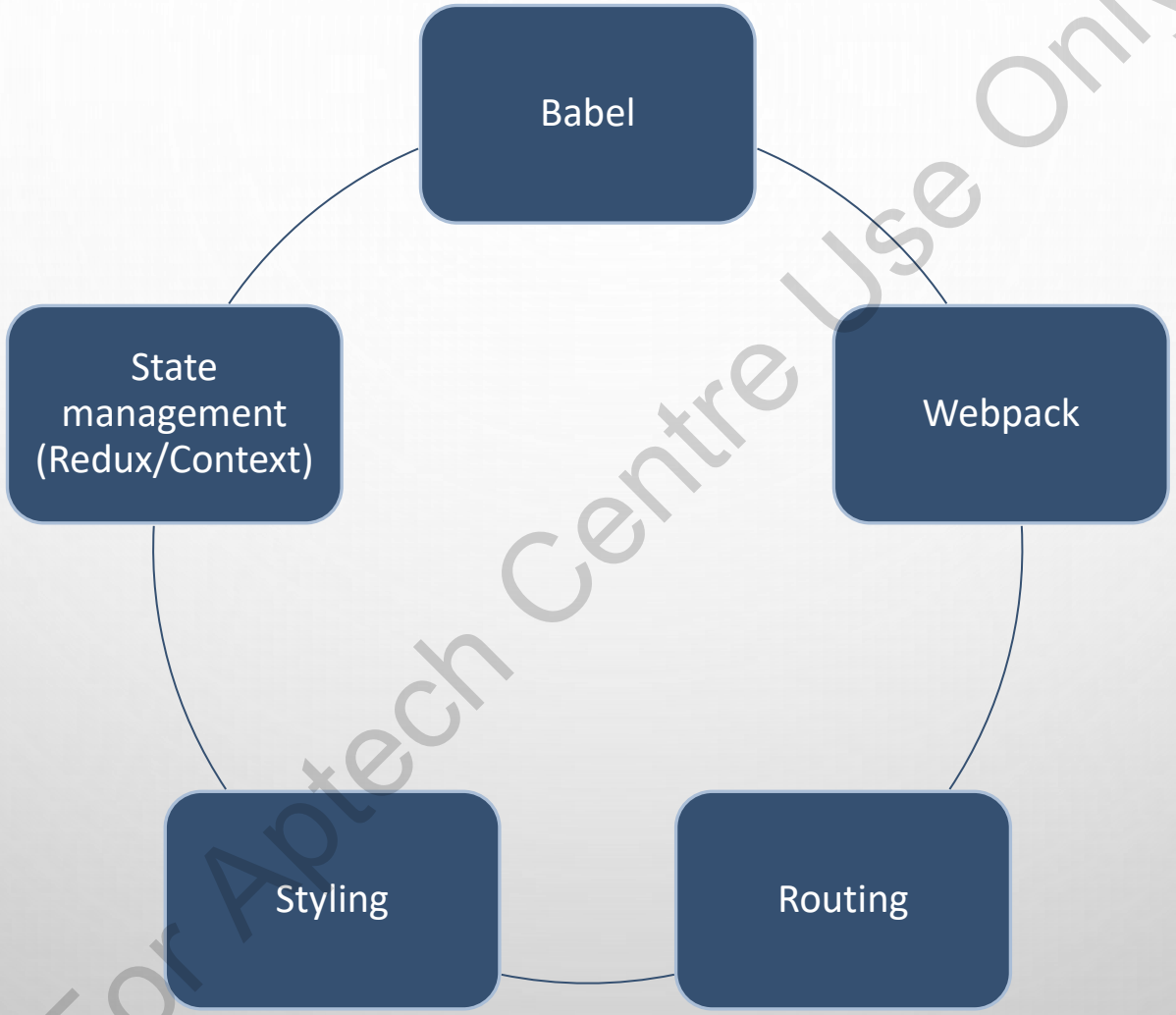
Declarative Views

- Predictable Code
- Easy to debug
- Designs simple views for each state
- Facilitates updating

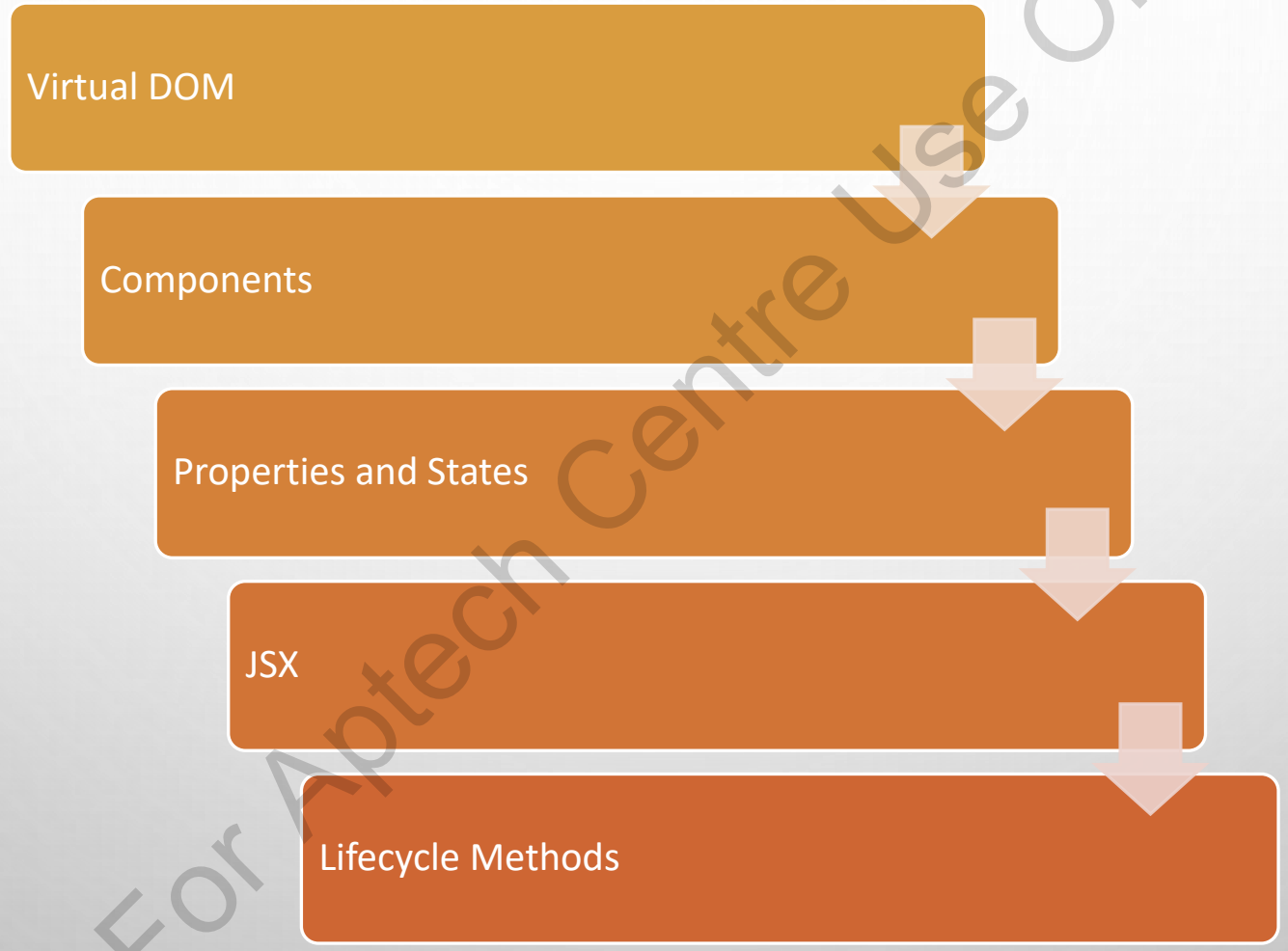
Component-Based UI Design

- Components manage their own states
- Allows designing interactive and complex UIs utilizing multiple components
- Helps passing rich data through application
- Allows keeping states out of DOM

REACTJS ECOSYSTEM



CORE CONCEPTS OF REACTJS [1-4]



CORE CONCEPTS OF REACTJS [2-4]

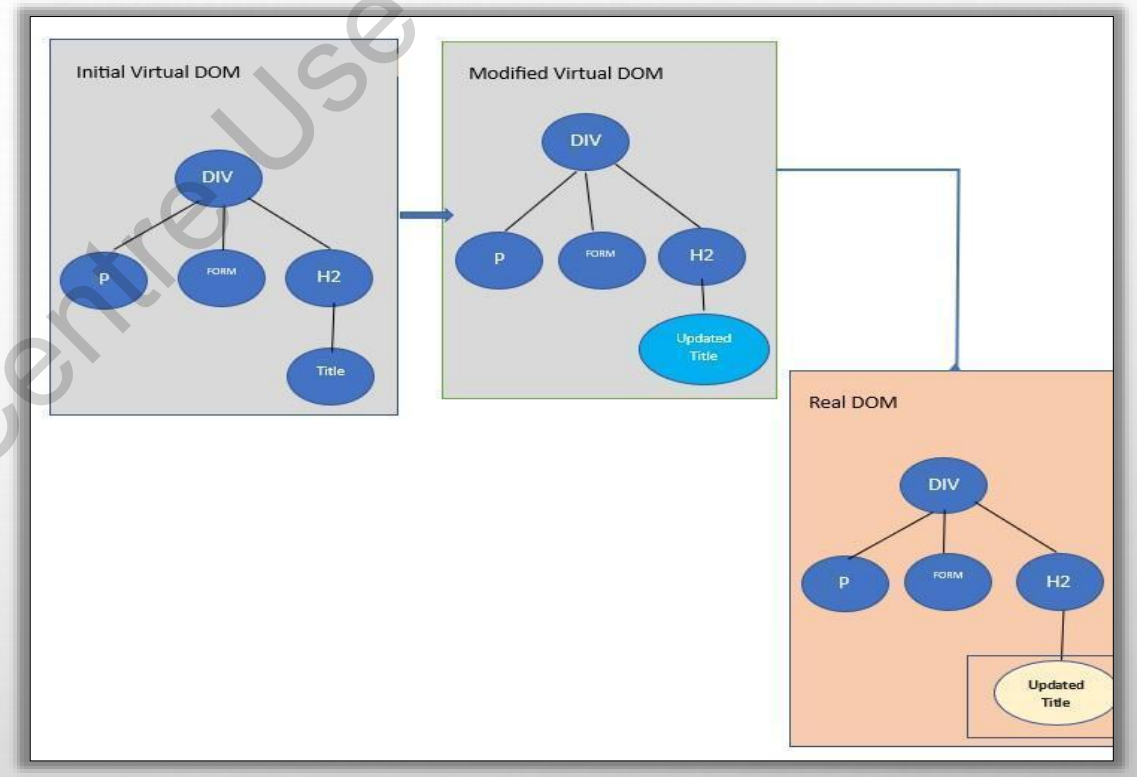
Virtual DOM

Copy, clone, or virtual image of DOM

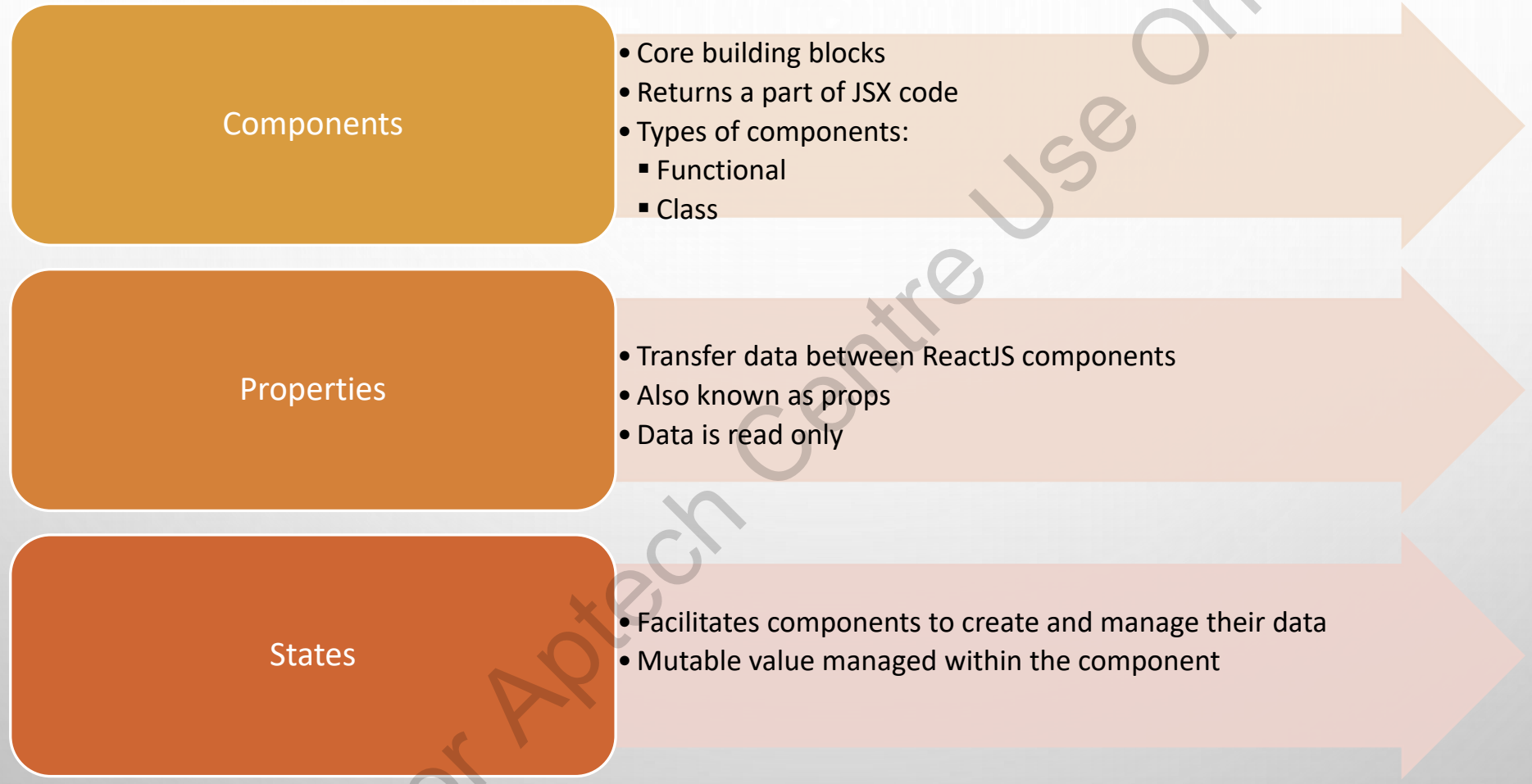
Allows faster access and manipulation

ReactJS has two Virtual DOM trees:

- Updated Virtual DOM
- Pre-Update version of the Virtual DOM



CORE CONCEPTS OF REACTJS [3-4]

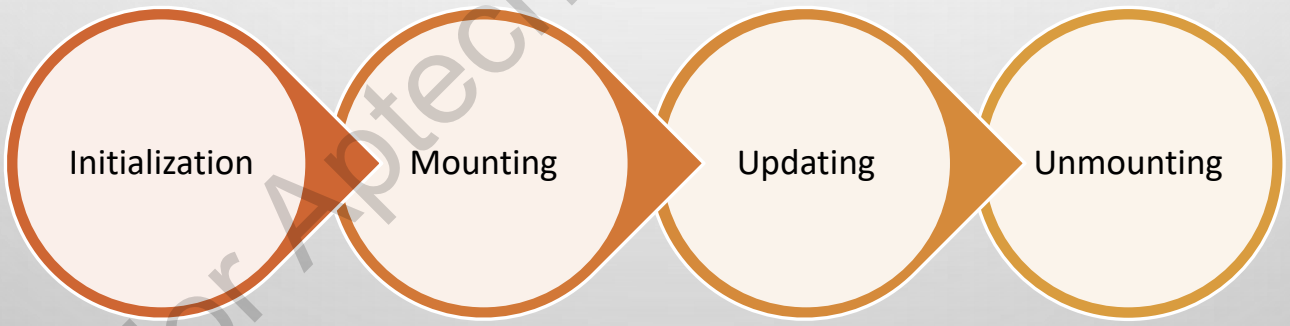


CORE CONCEPTS OF REACTJS [4-4]

JavaScript XML

- Extension of the JavaScript language syntax
- Allows writing JavaScript codes with an HTML/XML-like syntax
- Co-existence of JavaScript/ ReactJS code
- Faster processing
- Easy template creation

Lifecycle Methods



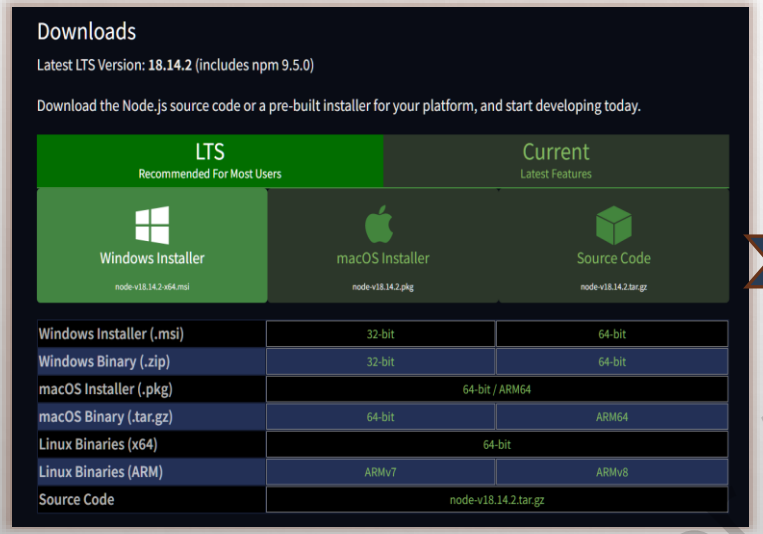
SETTING UP THE REACTJS DEVELOPMENT ENVIRONMENT [1-2]

JavaScript Runtime

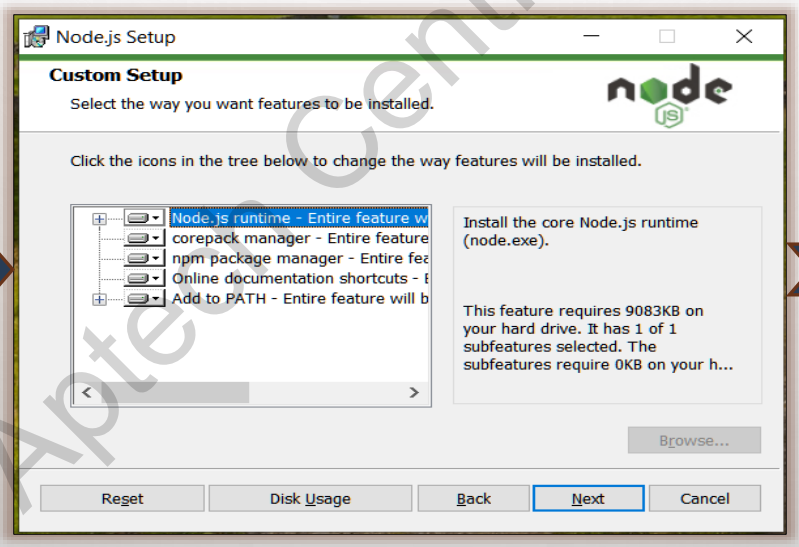
Integrated Development Environment (IDE)

Create-React-app library

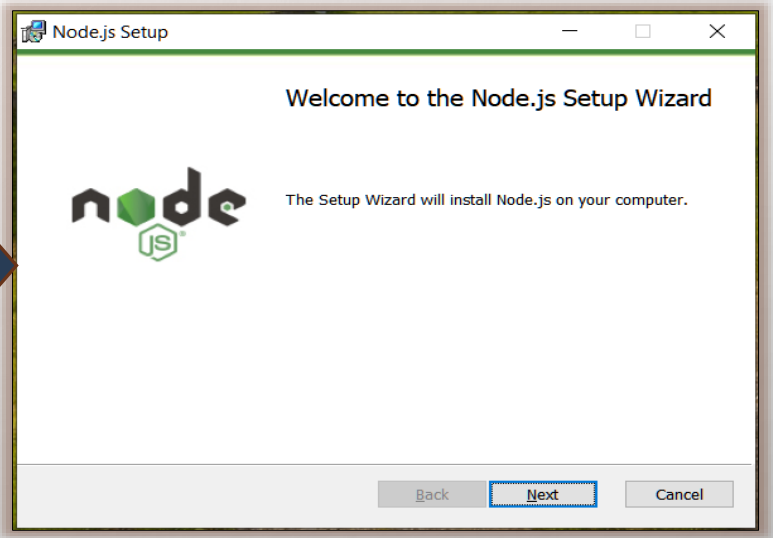
Installing NodeJS



Installing NodeJS



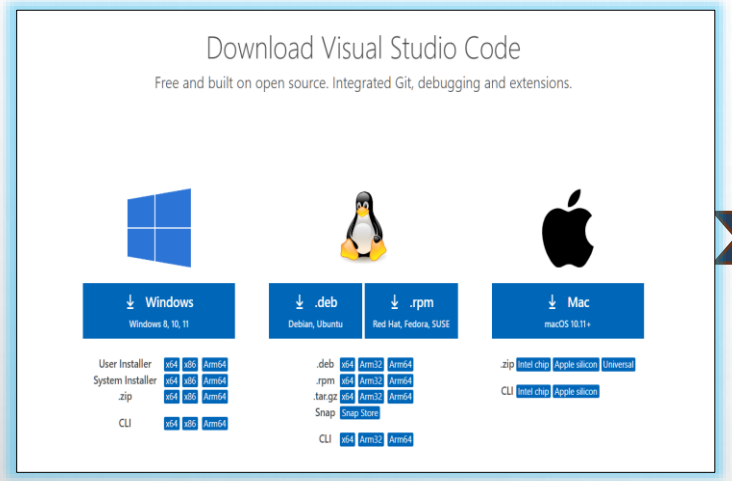
Custom Setup



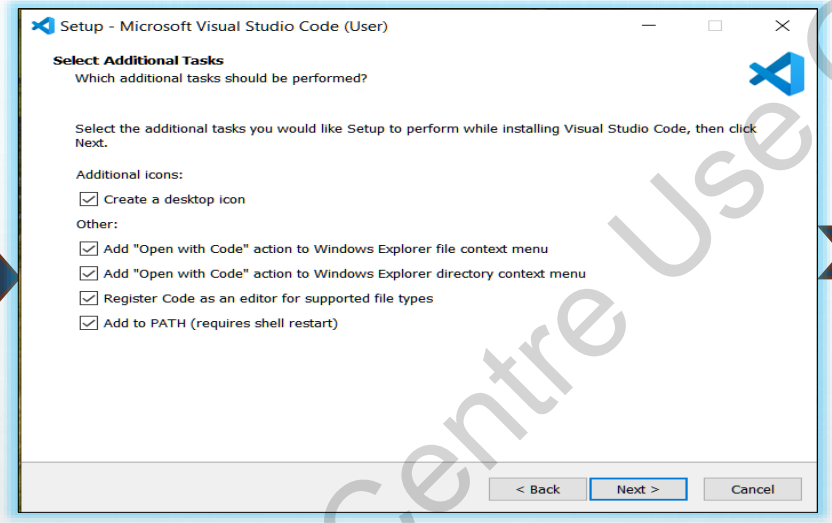
Installation Wizard

SETTING UP THE REACTJS DEVELOPMENT ENVIRONMENT [2-2]

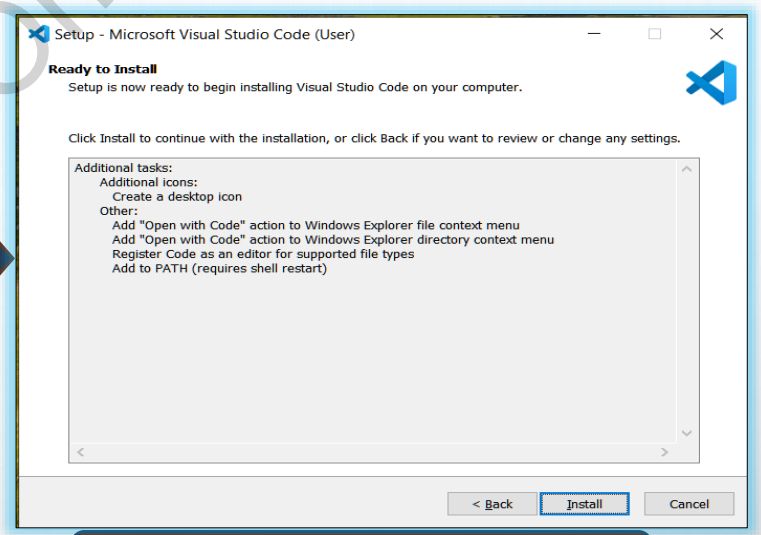
Installing the Visual Studio Code IDE



Downloading Visual Studio Code

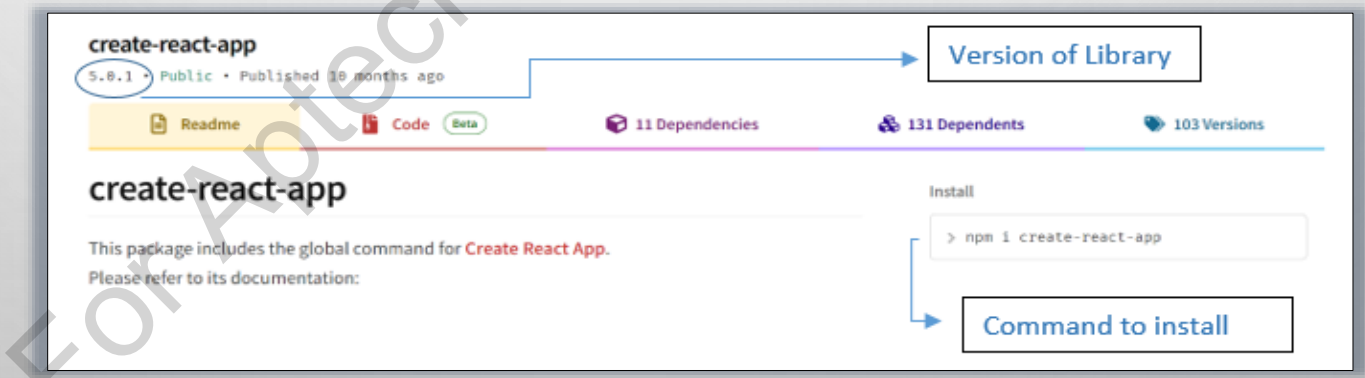


Selecting Additional Tasks

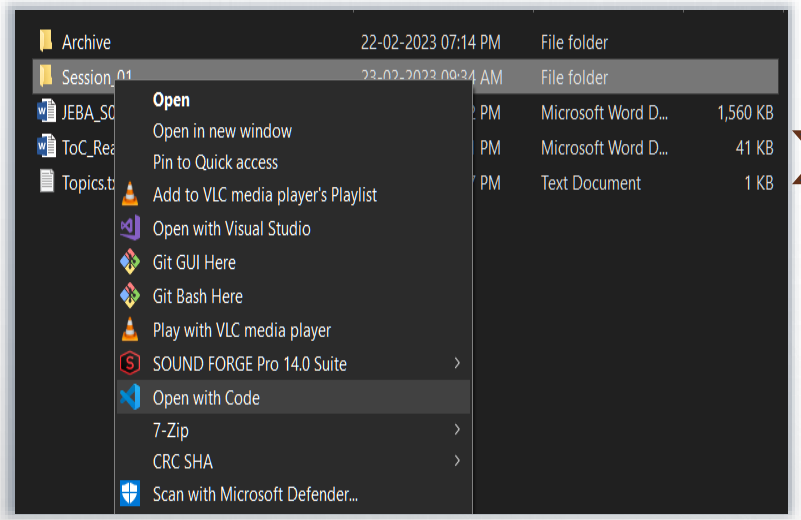


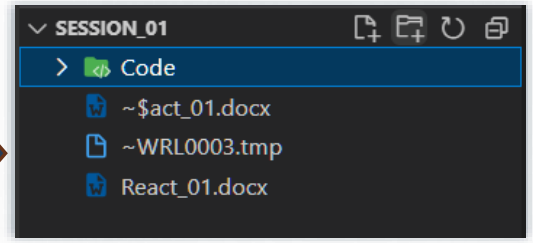
Installing Visual Studio Code

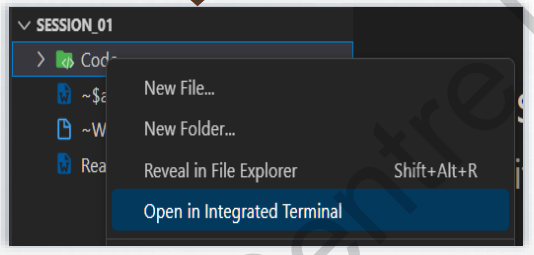
Installing the create-ReactJS-app Library

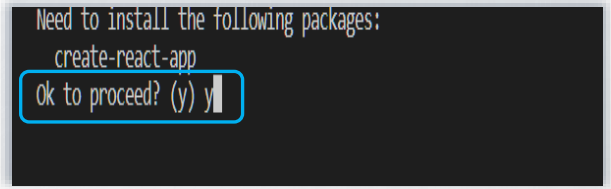


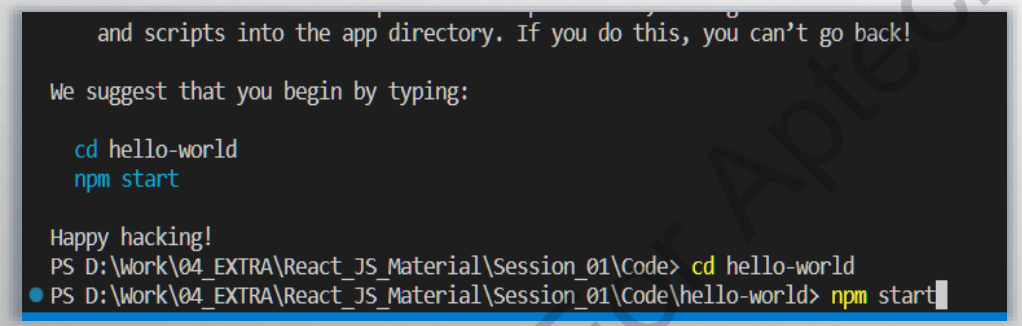
CREATING A SAMPLE REACTJS APP

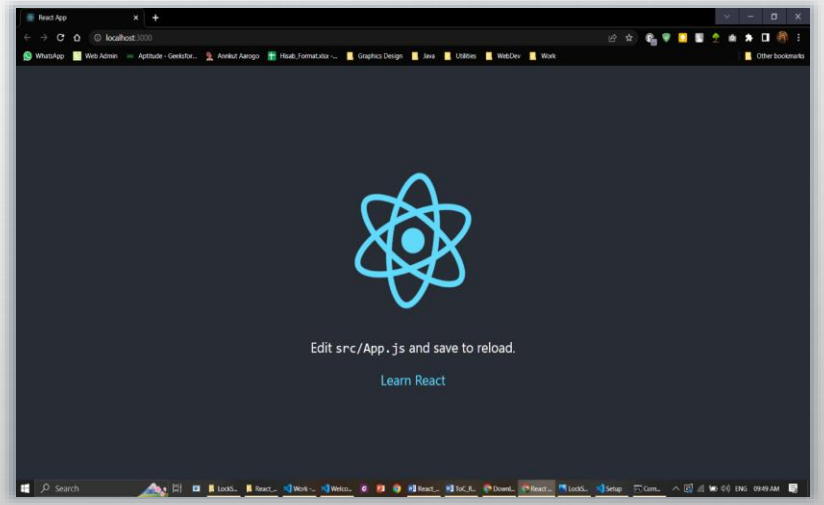












SUMMARY [1-2]

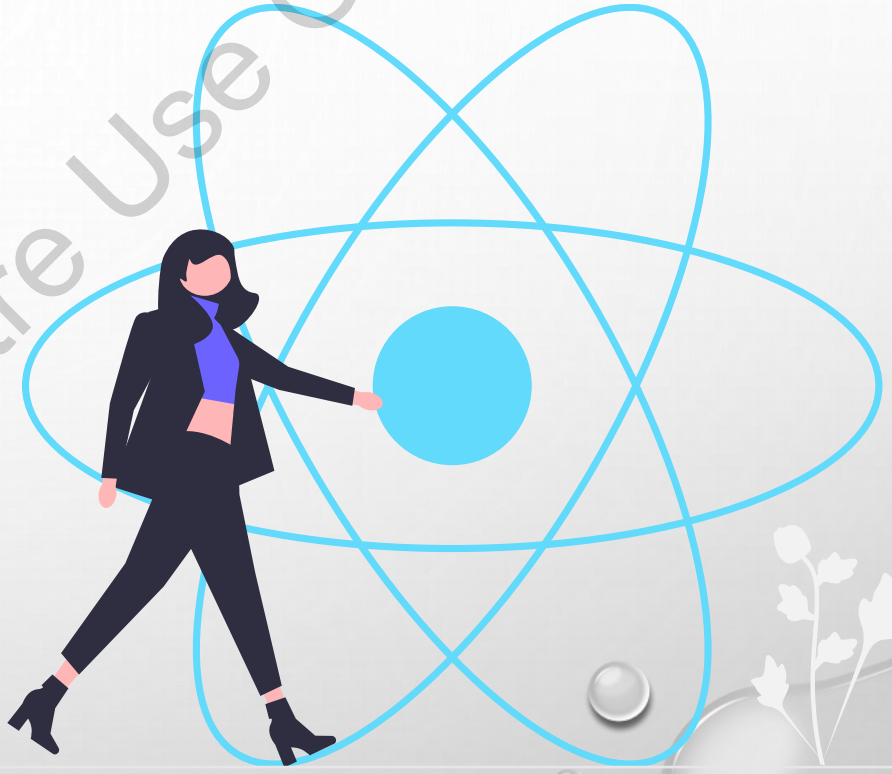
- ReactJS is an open-source front-end JavaScript library, which is used for building component-based User Interfaces (UI).
- ReactJS helps in designing simple declarative views and facilitates building component-based interactive user interfaces.
- The ReactJS ecosystem has the following components:
  - Babel
  - Webpack
  - Routing
  - Styling
  - State (Redux/Context)

SUMMARY [2-2]

- Virtual DOM is a copy, clone, or a virtual image of the DOM. ReactJS uses a virtual DOM. For every object in the DOM, there is an equivalent object in the virtual DOM.
- The process of comparing the pre-updated version of the virtual DOM and the updated version is known as Diffing.
- To set up the ReactJS development environment, users must install JavaScript Runtime, Integrated Development Environment (IDE), and the create-react-app library.

SESSION 02

REACTJS COMPONENTS



OBJECTIVES

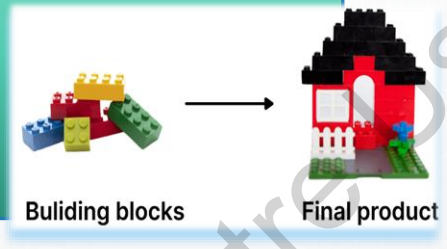
In this session, the topics discussed are as follows:

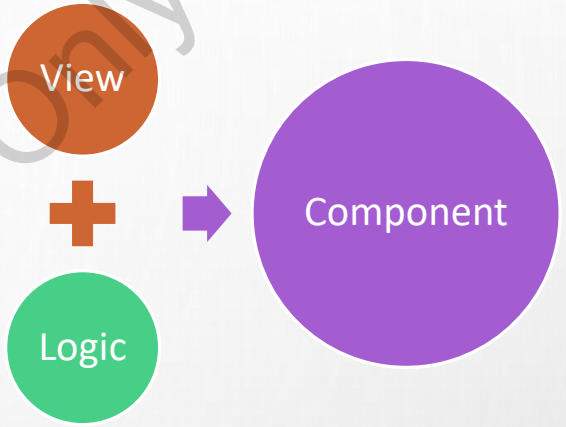
- Explain ReactJS components and the types of ReactJS components
- Distinguish between Functional and Class components
- Explain how to pass data between ReactJS components

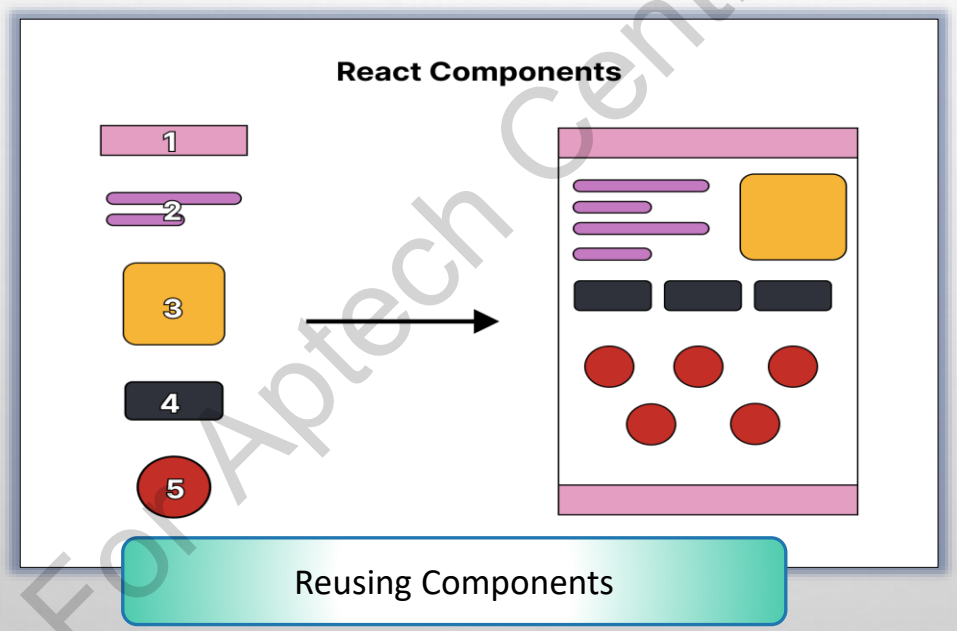
COMPONENTS

Components

- Foundation of ReactJS applications
- Reusable throughout the application
- Classified into two categories:
  - Functional components
  - Class components







USING FUNCTIONAL COMPONENTS

Code Snippet 1

```
function Welcome() {
return (
<section>
<h1>Hello World
🎉🎉</h1>
</section>
);
}
```

Function name has capital letters

Calling a function inside App component

```
function App() {
return (
<div className="App">
<Welcome />
</div>
);
}
```

Code Snippet 2

Code Snippet 3

```
function App() {
return (
<div className="App">
<Welcome />
<Welcome />
<Welcome />
</div>
);
}
```

Component reusability

Output of Welcome function

Reusability of Welcome function

USING CLASS COMPONENTS [1-2]

Code Snippet 4

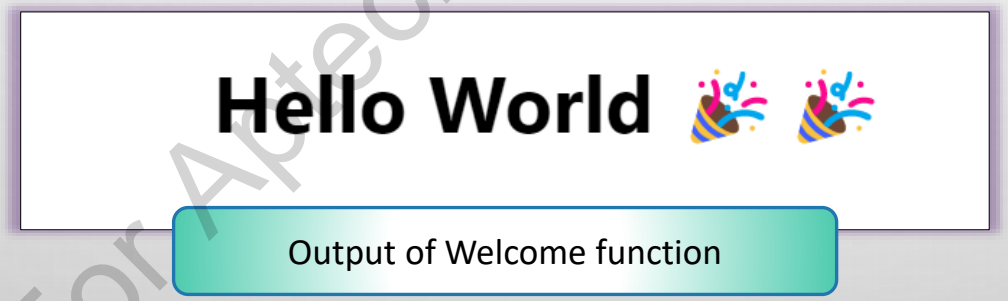
```
class Welcome extends
 React.Component {
  render() {
  return <h1>Hello World
  🎉🎉</h1>;
  }
}
```

React.Component used to extend Welcome

Calling a function inside App component

```
class App extends React.Component
 {
  render() {
  return <Welcome />;
  }
}
```

Code Snippet 5



Output of Welcome function

USING CLASS COMPONENTS [2-2]

```jsx
class MovieDetail extends React.Component {
  state = {
    isMobileScreen: window.innerWidth <= 450,
    movie: null
  };

  componentDidMount() {
    this.fetchMovie();
    window.addEventListener("resize", this.handleResize);
  }

  componentDidUpdate(prevProps) {
    if (this.props.movieId !== prevProps.movieId) {
      this.fetchMovie();
    }
  }

  componentWillUnmount() {
    window.removeEventListener("resize", this.handleResize);
  }

  handleResize = () => {
    if (this.state.isMobileScreen && window.innerWidth > 450) {
      this.setState({ isMobile: false });
    } else if (!this.state.isMobileScreen && window.innerWidth <= 450) {
      this.setState({ isMobile: true });
    }
  };

  fetchMovie = () => {
    fetch(`https://movie-api.com/movie/${this.props.movieId}`)
      .then(response => response.json())
      .then(movie => this.setState({ movie }));
  };

  render() {
    if (!this.state.movie) return null;
    return (
      <section>
        {!this.state.isMobileScreen && <img src={this.state.movie.pic} />}
        <p>{this.state.movie.name}</p>
      </section>
    );
  }
}
```

```jsx
function MovieDetail({ movieId }) {
  const [isMobileScreen, setIsMobileScreen] = useState(window.innerWidth <= 450);
  const [movie, setMovie] = useState(null);

  useEffect(() => {
    const handleResize = () => {
      if (isMobileScreen && window.innerWidth > 450) {
        setIsMobileScreen(false);
      } else if (!isMobileScreen && window.innerWidth <= 450) {
        setIsMobileScreen(true);
      }
    };
    window.addEventListener("resize", handleResize);
    return () => window.removeEventListener("resize", handleResize);
  }, []);

  useEffect(() => {
    fetch(`https://movie-api.com/movie/${this.props.movieId}`)
      .then(response => response.json())
      .then(movie => setMovie(movie));
  }, [movieId]);

  if (!movie) return null;
  return (
    <div>
      {!isMobileScreen && <img src={movie.image} />}
      <p>{movie.name}</p>
    </div>
  );
}
```

Code Snippets Using Class Component and Functional Component

PASSING DATA BETWEEN COMPONENTS

Code Snippet 6

```
function App() {
 return (
 <div className="App">
 <Welcome name="Neo" />
 </div>
 );
}
```

Flow of data from parent to child

Flow of data from parent to child?

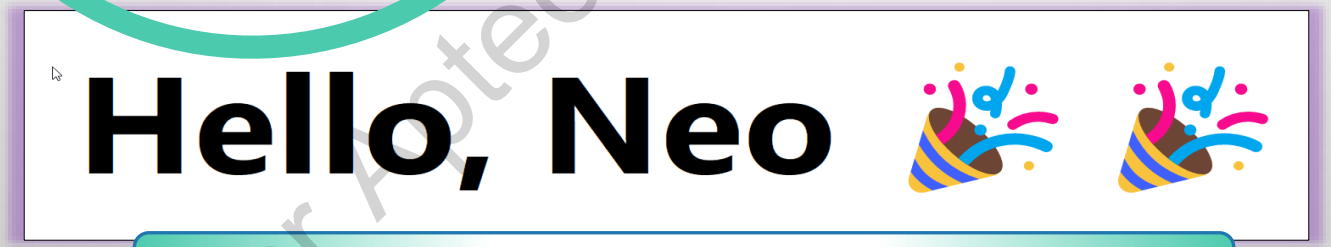
```
function Welcome(props) {
 return (
 <section>
 <h1>
 Hello, <span>{props.name}</span>
 🎉🎉
 </h1>
 </section>
 );
}
```

Code Snippet 7

Code Snippet 8

```
function Welcome({ name }) {
return (
<section>
<h1>
Hello, <span>{name}</span>
 🎉🎉
</h1>
</section>
);
}
```

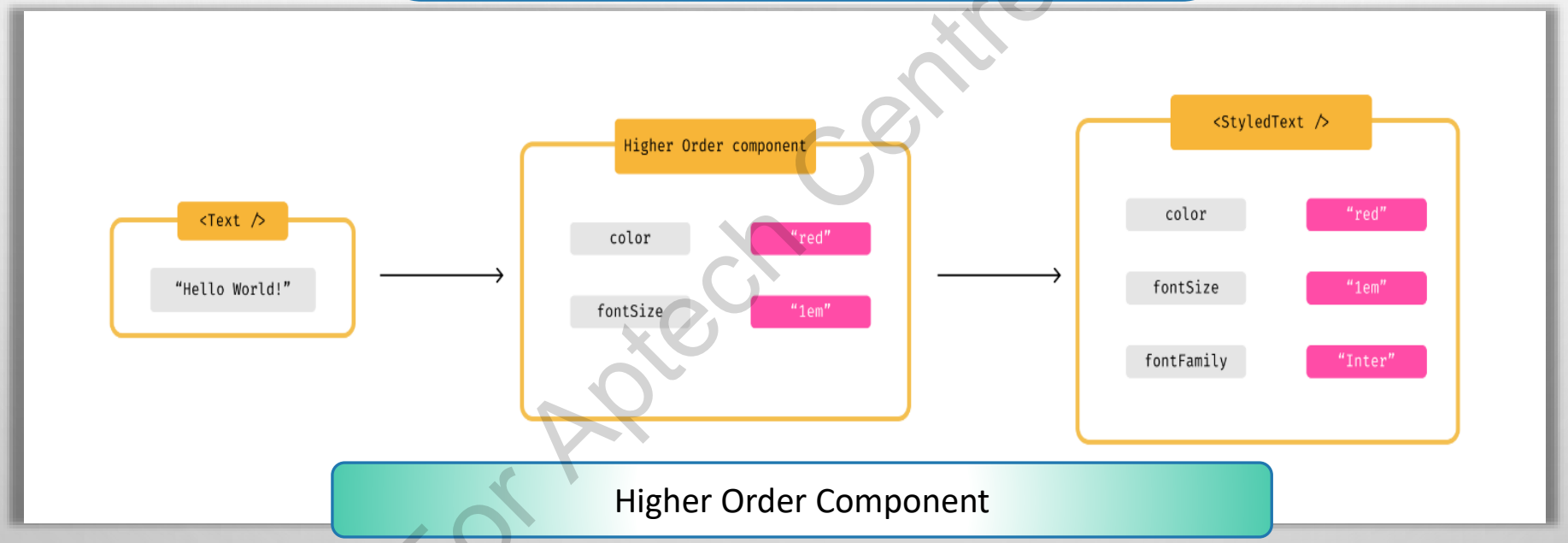
Destructuring feature



Output showing passing data between components

HIGHER ORDER COMPONENTS [1-2]

- Wrap components
- Acts as a template component for building new components
- useful in theming and customizing a component



HIGHER ORDER COMPONENTS [2-2]

Code Snippet 9

```
function withStyles(Component)
{
    return (props) => {
      const style = {
        color: "red",
        fontSize: "1em",
        // Merge props
        ...props.style,
      };

      return <Component
{...props} style={style} />;
    };
}
```

Code Snippet 10

```
const Text = ({ style = {} }) =>
(
  <p style={{ ...style,
fontFamily: "Inter" }}>Hello
world!</p>
); // existing component


const StyledText =
withStyles(Text); // new
Component
```

Code Snippet 11

```
function App() {
  return (
    <div className="App">
      <Welcome name="Neo" />
      <Text />
      <StyledText />
    </div>
  );
}
```

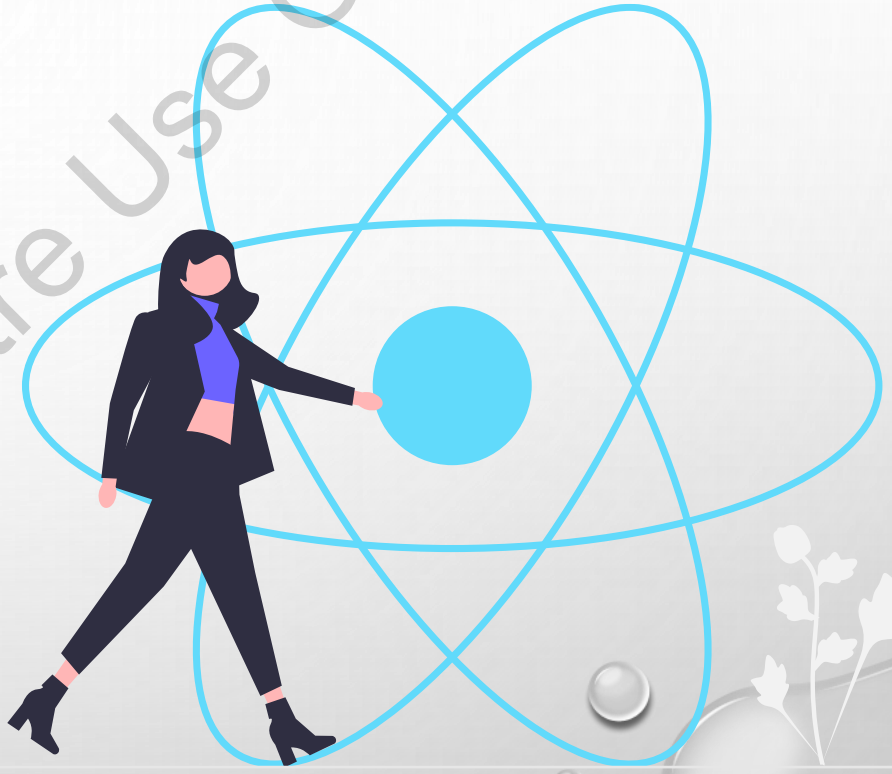
Output

SUMMARY

- Components are the building blocks of ReactJS apps.
- Two types of ReactJS components are class components and functional components.
- The functional component is the preferred one.
- Props flow in one direction.
- HOC are template components for building new components.

SESSION 03

JSX ELEMENTS



OBJECTIVES

In this session, the topics discussed are as follows:

- Illustrate creating and rendering JSX elements
- Illustrate nesting JSX Elements
- Explain JSX expressions
- Explain the properties of JSX

CREATING AND RENDERING JSX ELEMENTS

JavaScript XML (JSX)

- HTML-like markup within a JavaScript file
- JSX converted into JavaScript by Webpack and Babel
- Rendering logic and markup placed in a single file
- Defines layout and visual presentation of user interface (UI) components

```
<div>
  <p></p>
  <form>
  </form>
</div>
```

HTML File

```
isLoggedIn() {...}
onClick() {...}
onSubmit() {...}
```

JavaScript File

```
Sidebar() {
  if (isLoggedIn()) {
    <p>Welcome</p>
  } else {
    <Form />
  }
}
```

Markup of the Sidebar

```
Form() {
  onClick() {...}
  onSubmit() {...}
  <form onSubmit>
    <input onClick />
    <input onClick />
  </form>
}
```

Markup of the Button

PROPERTIES OF JSX

Rules and Properties of JSX

- Return single root element
- Close all tags
- CamelCase

Code Snippet 1

```
function UserProile() {
 return (
 <div>
 <h1>Mark</h1>
 <img
 src="#"
 alt="Mark"
 class="user-profile-pic"
 >
 <h2>Hobbies</h2>
 <ul>
 <li>Badminton
 <li>Chess
 <li>Gaming
 </ul>
 </div>
 )
 }
```

Wrong JSX code

Code Snippet 2

Correct rules followed in JSX

```
function UserProile() {
 return (
 <div>
 <h1>Mark</h1>
 <img
 src="#"
 alt="Mark"
 className="user-profile-pic"
 />
 <h2>Hobbies</h2>
 <ul>
 <li>Badminton </li>
 <li>Chess </li>
 <li>Gaming </li>
 </ul>
 </div>
 );
 }
```

Code Snippet 3

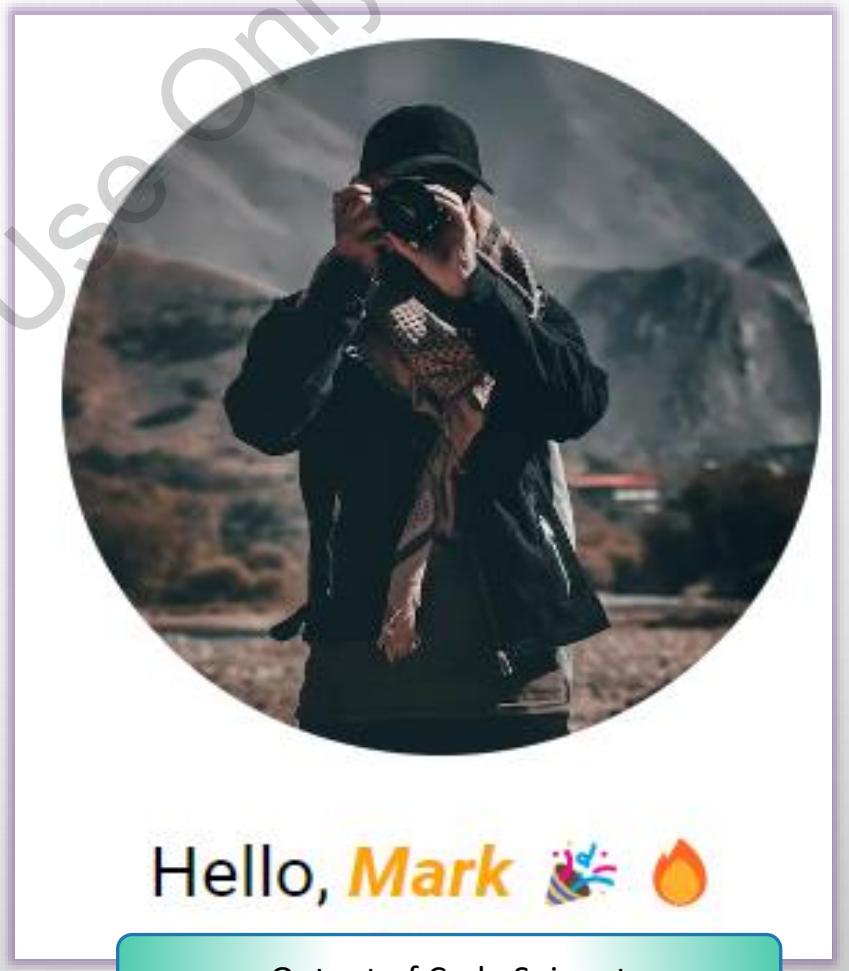
```
function UserProile() {
return (
<>
<h1>Mark</h1>
<img
src="#"
alt="Mark"
class="user-profile-pic"
>
<h2>Hobbies</h2>
<ul>
<li>Badminton
<li>Chess
<li>Gaming
</ul>
</>
)
}
```

<> </> as the wrap option

JSX EXPRESSIONS

Code Snippet 4

```
function User() {
  const name = "Mark";
  const pic =
    "https://images.pexels.com/photos/1704488/pexels-photo-
1704488.jpeg?auto=compress&cs=tinysrgb&dpr=1&w=500";

  return (
    <section>
      <img className="user-profile-pic" src={pic} alt={name}
/>
      <h2 className="user-name">
        Hello, <span className="user-first-
name">{name}</span> 🎉🔥
      </h2>
    </section>
  );
}
```



Output of Code Snippet

NESTING JSX ELEMENTS [1-2]

Code Snippet 5

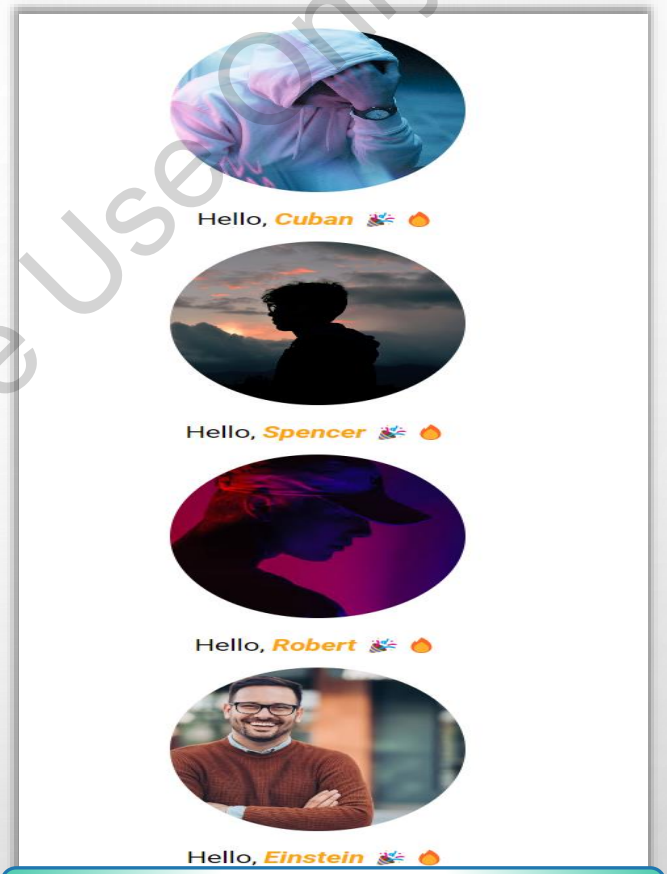
```
function App() {
  const names = ["Cuban",
"Spencer", "Robert", "Einstein"];
  return (
    <div className="App">
      {names.map((nm) => (
        <Welcome name={nm} />
      ))}
    </div>
  );
}
```

Output of Code Snippet

NESTING JSX ELEMENTS [2-2]

Code Snippet 6

```
function App() {
  const users = [
    {name: "Cuban",
      pic: "#",},
    {
      name: "Spencer",
      pic: "#",
    },
    {  name: "Robert",
      pic: "#", },
    {
name: "Einstein",
      pic: "#",
    },  ];
  return (
    <div className="App">
      {users.map((usr) => (
        <User name={usr.name} pic={usr.pic}
/>      ))}    </div>  );}
```



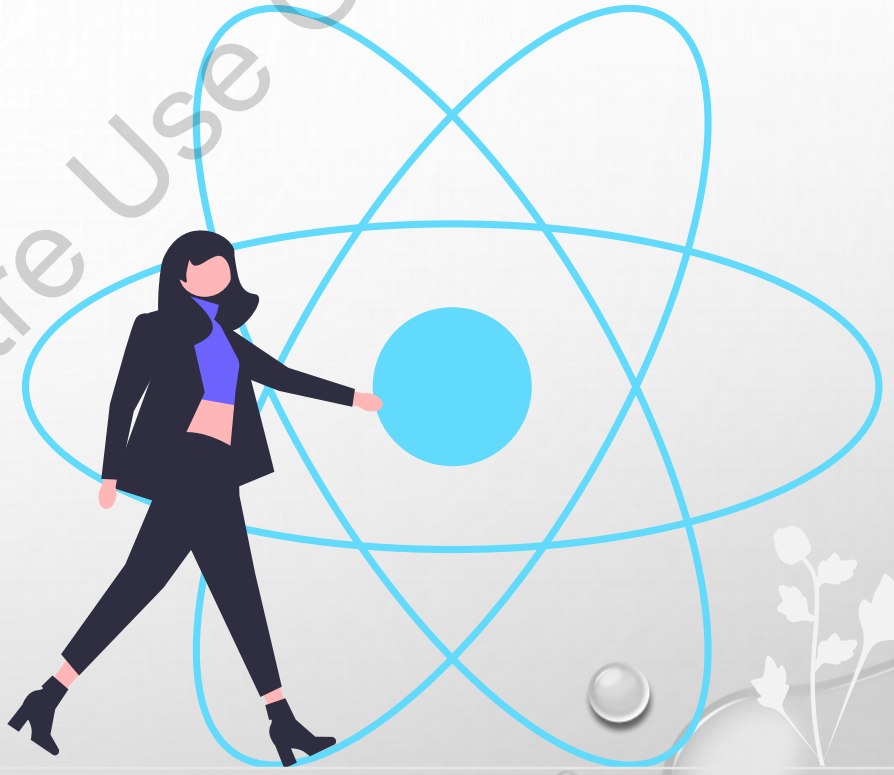
Output of Code Snippet

SUMMARY

- JavaScript XML (JSX) is used to keep related logic and View in one place.
- {} is known as Template syntax.
- Template syntax is used to evaluate JavaScript expressions.
- JSX is stricter than HTML.
- JSX follows three rules:
  - Returning a single root element.
  - Closing all tags.
  - Using Camel case (CamelCase).
- The map() method is used to iterate through array of data in ReactJS.

SESSION 04

EVENT HANDLING IN REACTJS



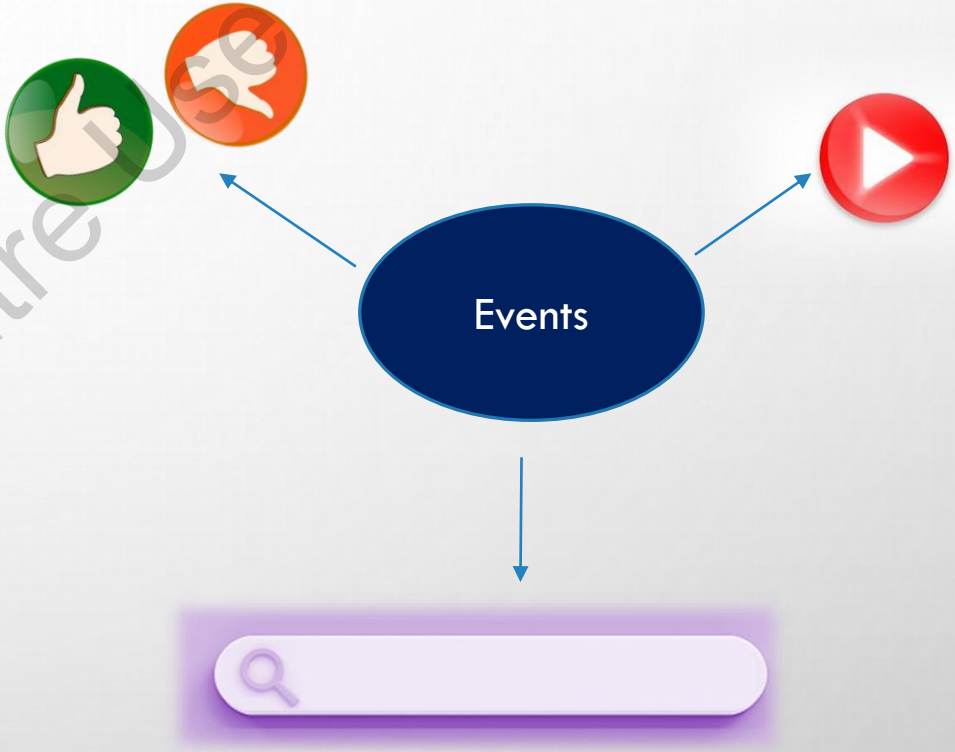
OBJECTIVES

In this session, the topics discussed are as follows:

- Explain event handling in ReactJS
- Explain the basic syntax of ReactJS event handling
- Explain different types of events
- Explain the best practices of event handling
- Explain Hooks

WHAT IS EVENT HANDLING IN REACTJS

- Adds interactivity to Web apps
- Allows responding to user input
- Similar to event handling in HTML

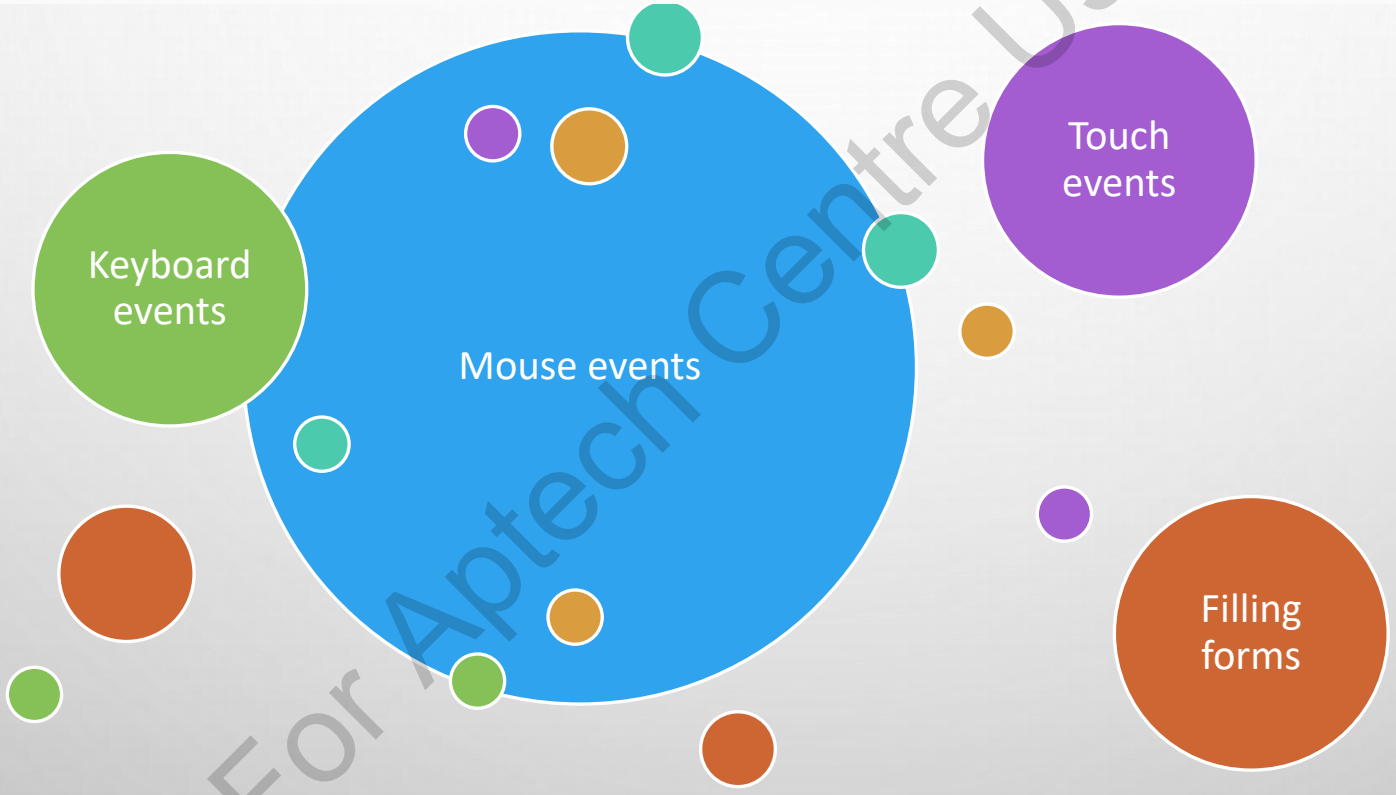


BASIC REACTJS EVENT HANDLING SYNTAX

Event handler

Callback function

```
<button onClick={() ⇒ console.log("Clicked 👆")}>Click me</button>;
```



BEST PRACTICES OF EVENT HANDLING

Naming Conventions of Event Handlers

- Use camelCase for event handler names. For example **onClick**

Arrow Functions with Event Handlers

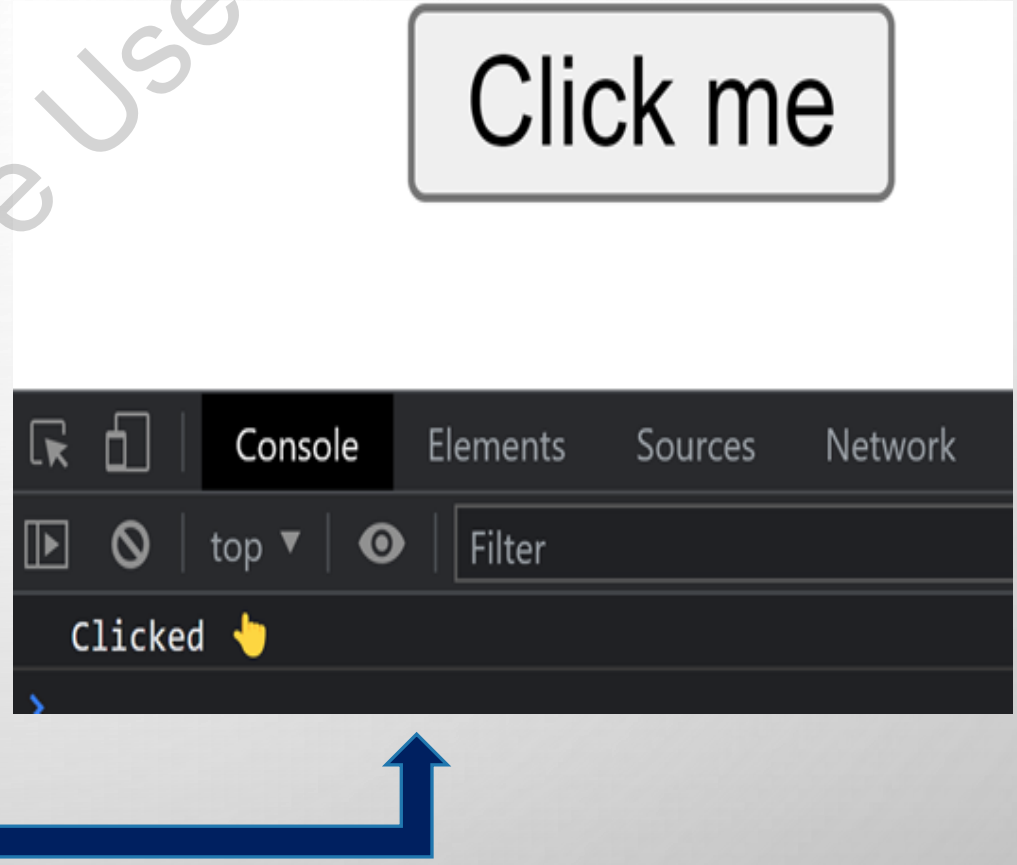
- No requirement for binding
- Easier to read
- Better performance

Callback function smaller in size

```
function ClickMe() {
return <button onClick={() => console.log("Clicked
 👆")}>Click me</button>;}
```

Callback function larger in size

```
function ClickMe() {
const onUserClick = () => console.log("Clicked
 👆");
return <button onClick={onUserClick}>Click
 me</button>;}
```



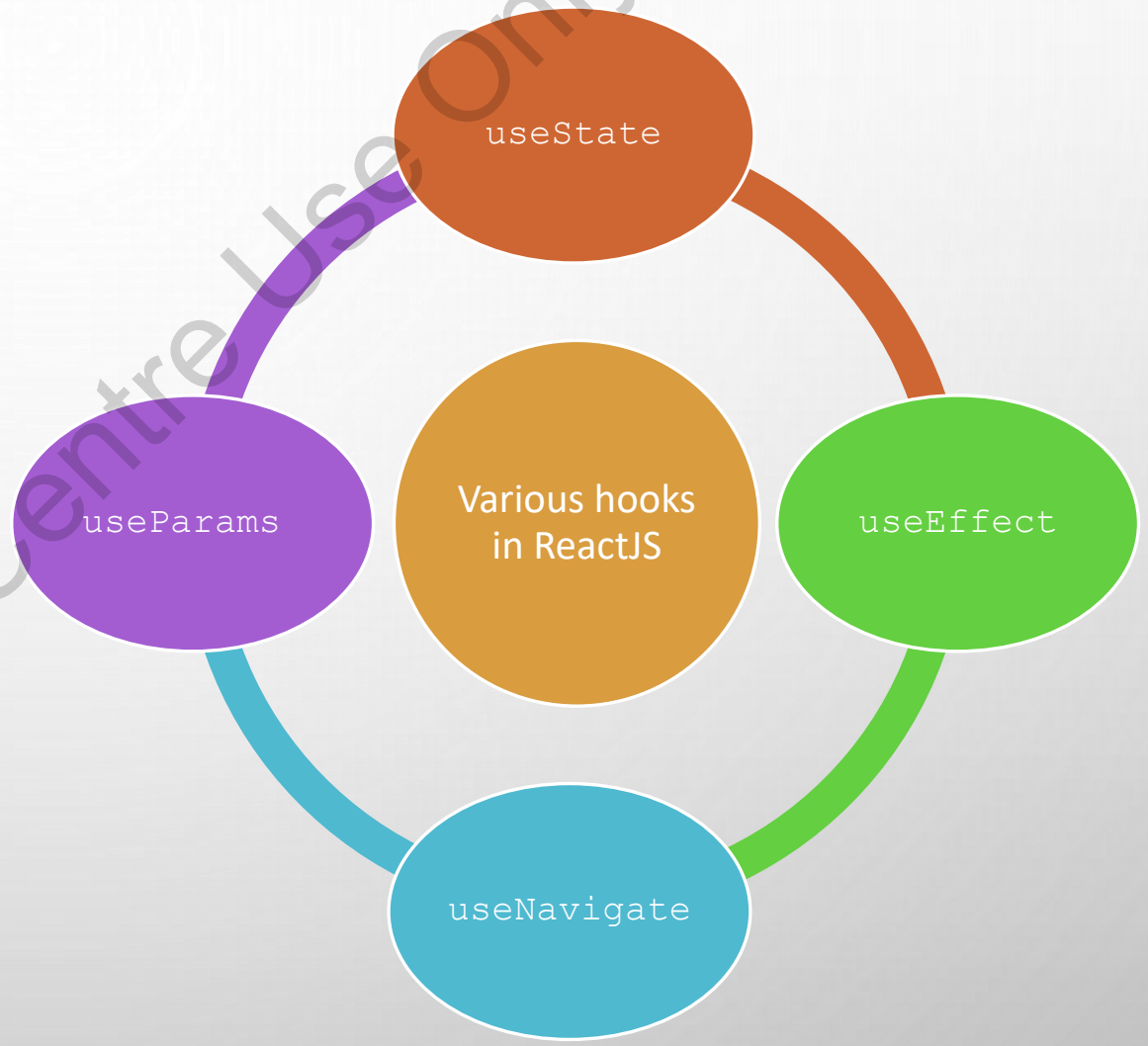
INTRODUCTION TO HOOKS

Features of Hooks

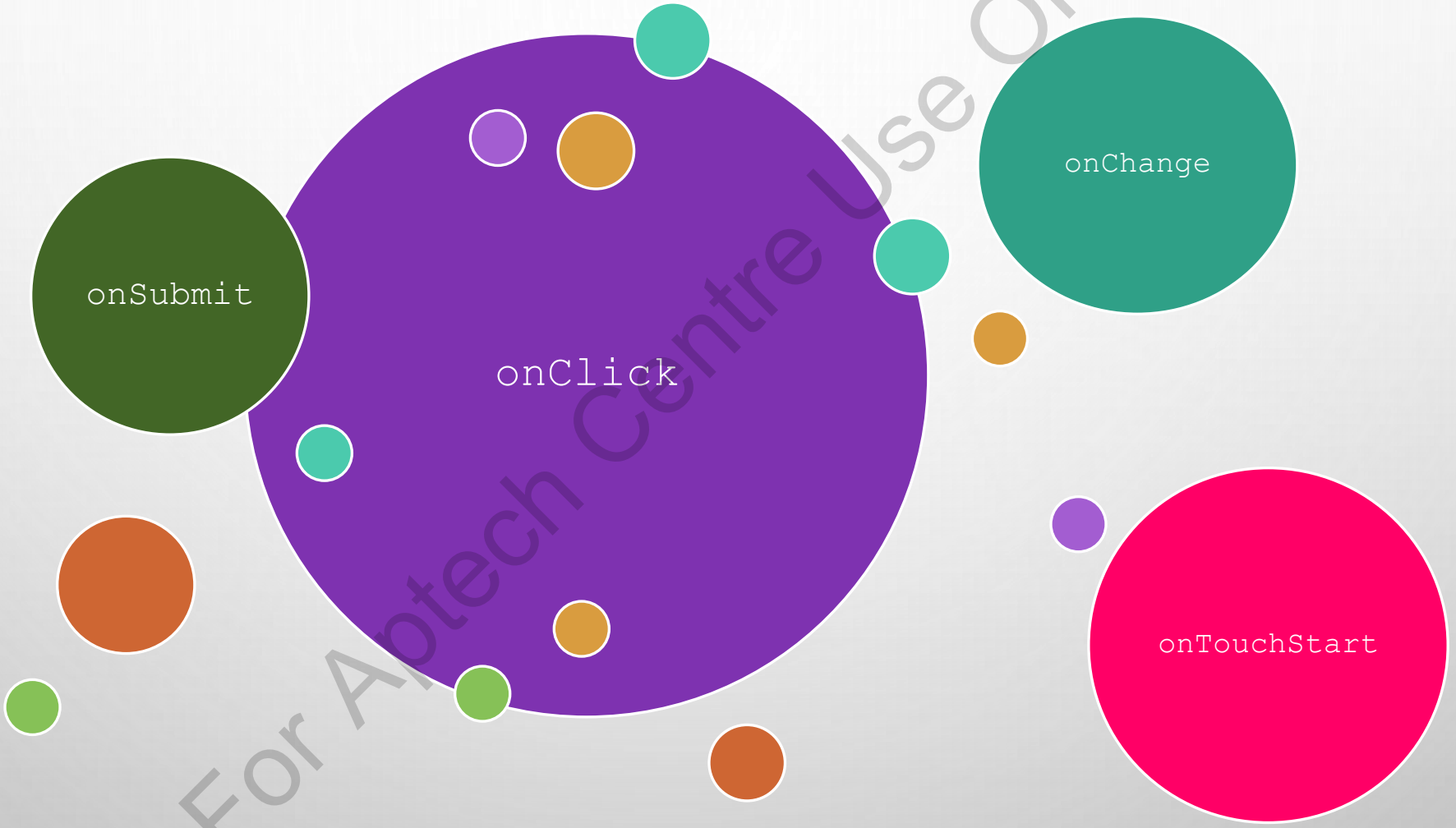
- Starts with the word, `use`
- Usually utilized in combination with event handlers
- ReactJS listens to changes made by hooks

Code Snippet 1

```
import { useState } from "react";

const [state, setState] =
useState(InitialValue)
```



TYPES OF EVENTS [1-6]

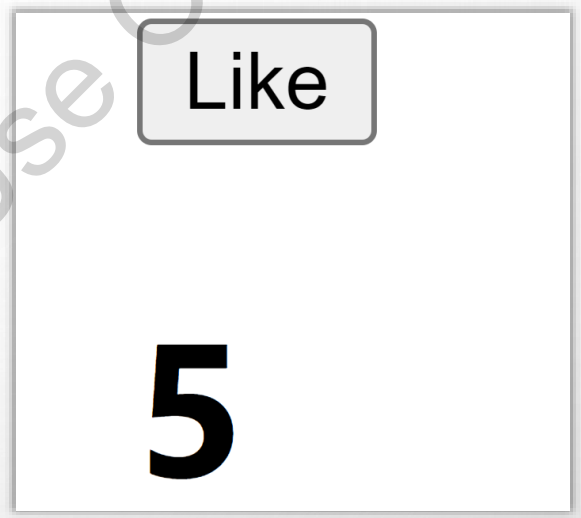


TYPES OF EVENTS [2-6]

Mouse Events

Code Snippet 2

```
function Counter() {
  let [like, setLike] =
useState(0);


  return (
    <div>
      {/* camelcase */}
      <button onClick={() =>
setLike(like + 1)}>Like</button>
      <h1>{like}</h1>
    </div>  );}
```



Output

TYPES OF EVENTS [3-6]

Keyboard Events

Code Snippet 3

```
function ColorBox() {
  const styles = {
    background: "orange",
  };
   return (
    <div>
      <h1>Color Box</h1>
      <input type="text"
style={styles}
placeholder="Type a color" />
    </div>
  );
}
```

Initialize the styles object

Code Snippet 4

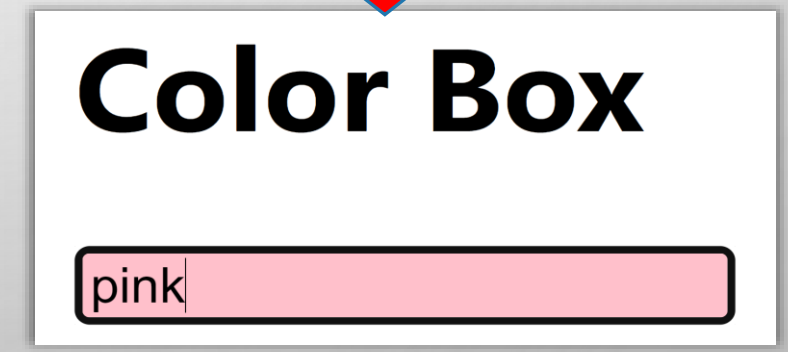
```
function ColorBox() {
  const [color, setColor] =
useState("orange");
  const styles = {
    background: color,
  };
  return (
    <div>
      <h1>Color Box</h1>
      <input type="text"
style={styles} placeholder="Type
a color" />
    </div>
  );
}
```

Color state provided to the background key

Code Snippet 5

```
function ColorBox() {
  const [color, setColor] =
useState("");
  const styles = {
    background: color,
  };
  return (
    <div>
      <h1>Color Box</h1>
      <input type="text"
style={styles}placeholder="Type a
color"
        onChange={(event) =>
setColor(event.target.value)}
      />
    </div>
  );}
```

onChange event handler listens to typing event from the user



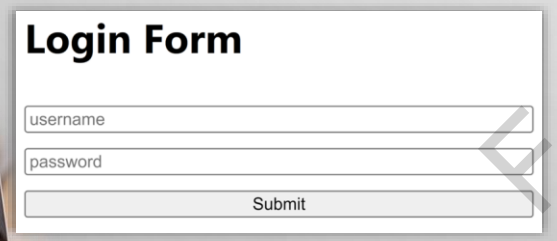
TYPES OF EVENTS [4-6]

Form Events

Code Snippet 6

```
function LoginForm() {
  return (
    <form className="login-
form">
      <h1>Login Form</h1>
      <input name="email"
type="email"
placeholder="username" />
      <input name="password"
type="password"
placeholder="password" />
      <button
type="submit">Submit</button>
    </form>
  );
}
```

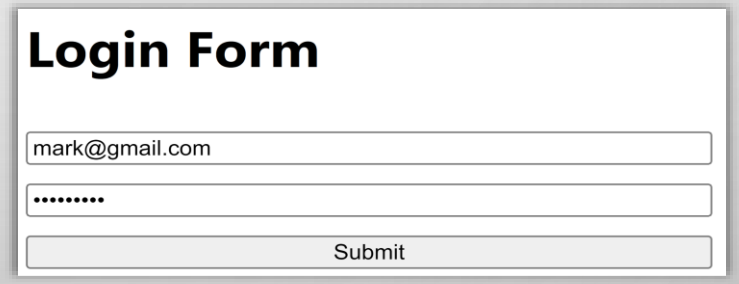
Input fields added to form element



Code Snippet 7

```
function LoginForm() {
const [email, setEmail] = useState("mark@gmail.com");
const [password, setPassword] = useState("secret123");
return (
  <form className="login-form">
    <h1>Login Form</h1>
    <input value={email}
      onChange={(event) => setEmail(event.target.value)}
      name="email"
      type="email"
      placeholder="username"
    />
    <input
      value={password}
      onChange={(event) => setPassword(event.target.value)}
      name="password"
      type="password"
      placeholder="password"
    />
    <button type="submit">Submit</button>
  </form>  );}
```

`useState` variables added to track the typed values



TYPES OF EVENTS [5-6]

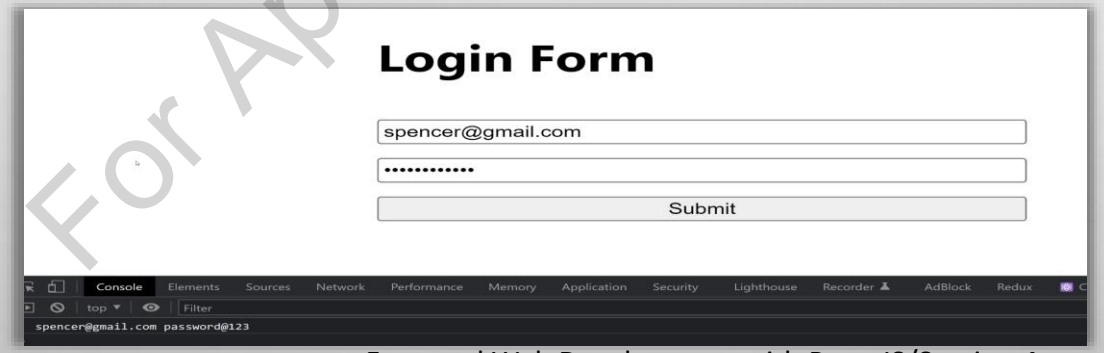
Code Snippet 8

```
function LoginForm() {
  const [email, setEmail] = useState("mark@gmail.com");
  const [password, setPassword] = useState("secret123");
  return (
    <form
      className="login-form"
      onSubmit={(event) => {
        event.preventDefault();
        console.log(email, password);
      }}
    >
      <h1>Login Form</h1>
<input
        name="email"
        value={email}
```

Code Snippet 9

```
onChange={(event) => setEmail(event.target.value)}
        type="email"
        placeholder="username"
      />
      <input
        name="password"
        value={password}
        onChange={(event) =>
setPassword(event.target.value)}
        type="password"
        placeholder="password"
      />
      <button type="submit">Submit</button>
    </form>
  );
```

Submit button triggers the `onSubmit` event handler

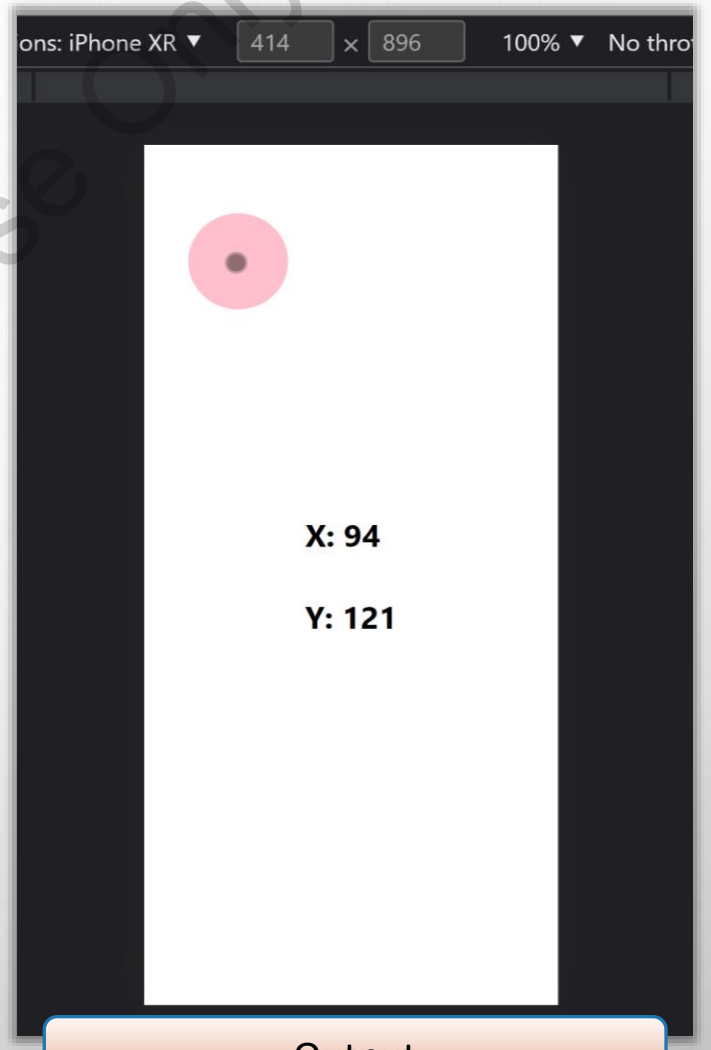


TYPES OF EVENTS [6-6]

Touch Events

Code Snippet 10

```
function PaintDot() {
  const [pos, setPos] = useState({});
  const styles = {
    position: "absolute",
    top: `${pos.pageY}px`,
    left: `${pos.pageX}px`,
    height: "100px",
    aspectRatio: "1/1",
    borderRadius: "50%",
    background: "pink",
    transform: "translateX(-50%)
translateY(-50%)",
  };
  return (
    <div
      onTouchStart={(event) =>
setPos(event.changedTouches[0])}
      className="dot-container"
    >
      <h1>X: {pos.pageX}</h1>
      <h1>Y: {pos.pageY}</h1>
      <div style={styles}></div>
    </div>
  );
}
```



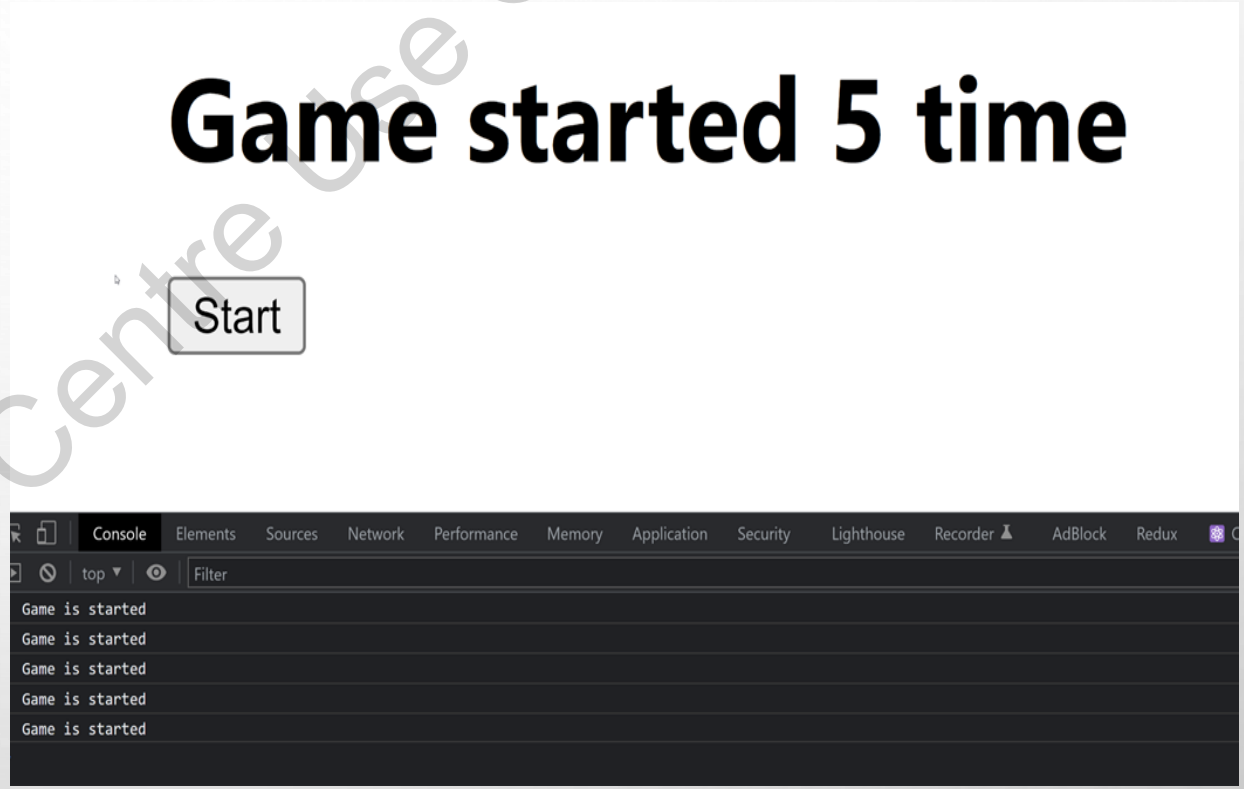
Output

HANDLING DIFFERENT EVENTS

User Event

Code Snippet 11

```
function Game() {
  let [count, setCount] = useState(0);
  const startGame = () => {
    console.log("Game is started");
    setCount(count + 1);
  };
  return (
    <div>
      <h1>Game started {count} time</h1>
      <StartGame onPlayerClick={startGame}
/>
    </div>
  );
}
function StartGame({ onPlayerClick }) {
  return (
    <div>
      <button onClick={() =>
onPlayerClick()}>Start</button>
    </div>
  );
```



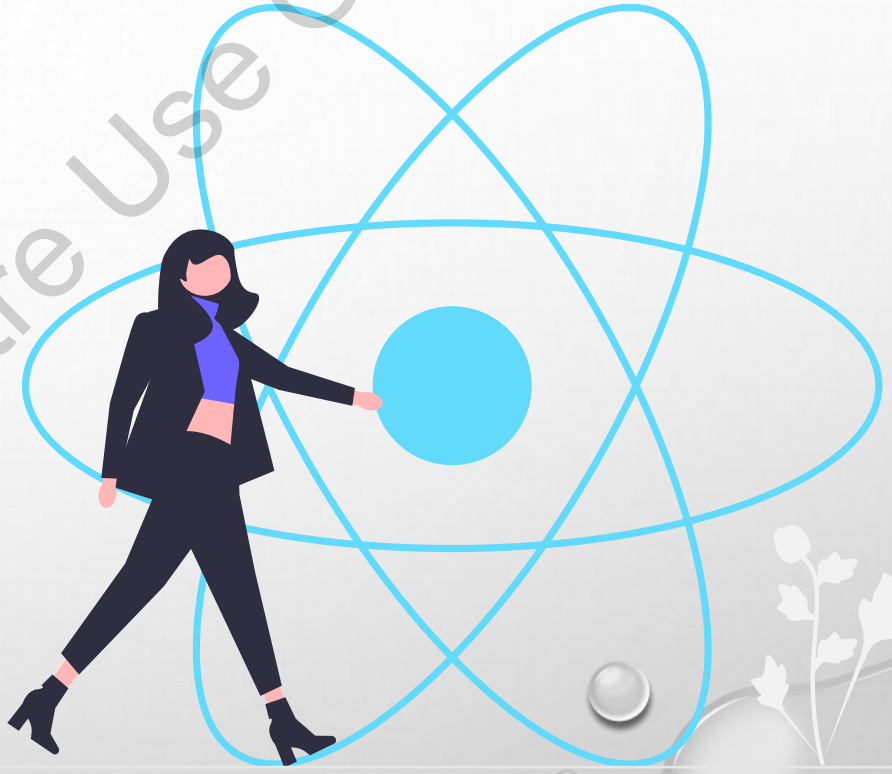
Output

SUMMARY

- Event handlers are attached to elements to add interactivity to the app.
- ReactJS listens to changes made by hooks.
- Hooks are functions that starts with the word, “use”.
- Event handlers must always be camelCased.
- The useState variables are declared to update the view.
- onClick, onSubmit, onChange, and onTouchStart are event handlers.
- Function can be passed as props to call the parent function.

SESSION 05

REACTJS ROUTER

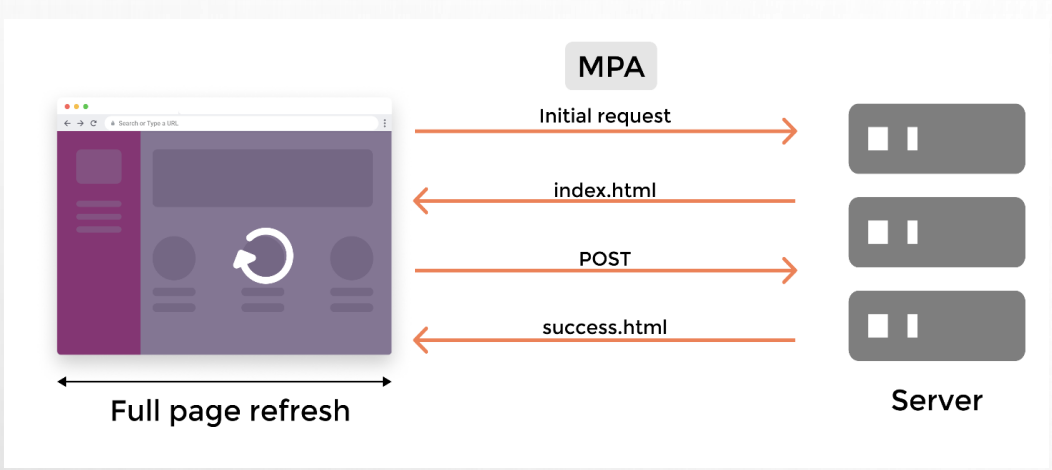


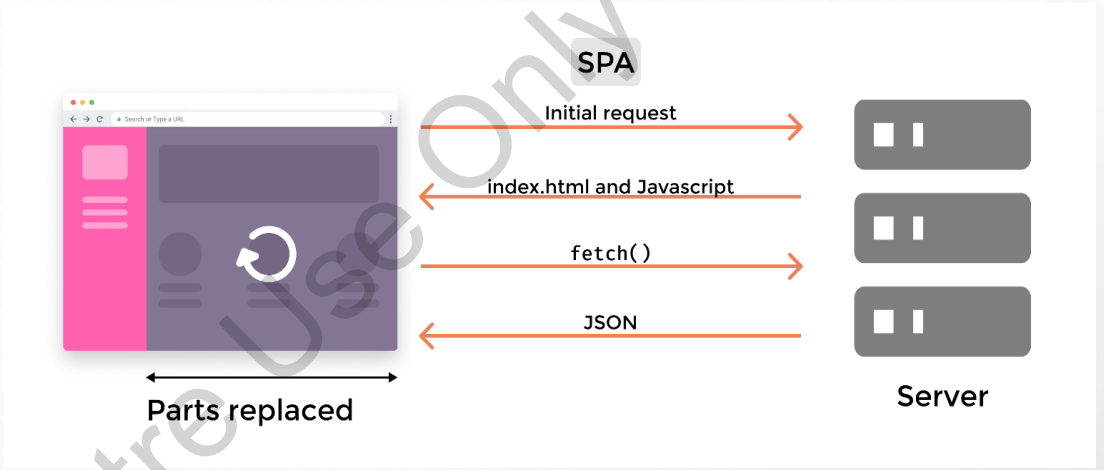
OBJECTIVES

In this session, the topics discussed are as follows:

- Explain about the ReactJS router and its basic components
- Explain the difference between single and multi-page applications
- Explain the advantages and drawbacks of single and multi-page applications

SINGLE-PAGE APPLICATIONS VS. MULTI-PAGE APPLICATIONS





Pros

- Search engine optimization
- Easier maintenance
- Better accessibility

Cons

- Blank screen for a few seconds
- Common elements are recreated
- Not bandwidth efficient
- Loss in performance

Pros

- No full page refresh
- Offline functioning
- Improved user experience
- Bandwidth efficient

Cons

- Complex development
- Additional Search Engine Optimization (SEO) Care
- Client-side JavaScript Disabled possibility
- Slow initial load

WHAT IS REACTJS ROUTER?

ReactJS Router

- Library for routing in React-based Web applications
- `react-router-dom` package
- Enables the creation of a single-page application

Adding ReactJS Router to an Existing Project

```
npm install react-router-dom@6
```

Advantages

- Well - organized app
- Easy and swift access to various app parts
- Page bookmark
- Secure pages behind a login form

BASIC COMPONENTS OF REACTJS ROUTER [1-3]

Router/ Browser Router → Route → Link → Nav

Code Snippet 1

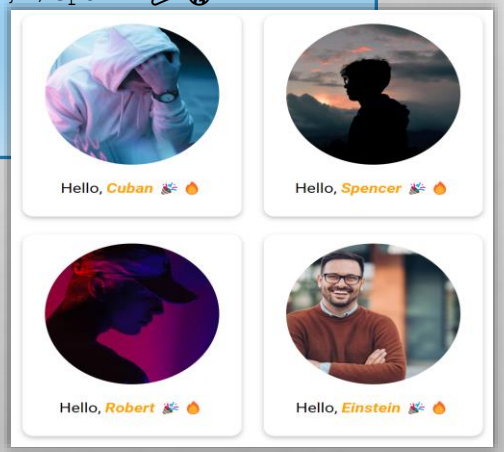
```
const USER_LIST = [
  {
    name: "Cuban",
    pic:
"https://images.unsplash.com/photo-
1618641986557-1ecd230959aa?ixlib=rb-
4.0.3&ixid=MnwxMjA3fDB8MHxzZWFyY2h8NXx8c
HJvZmlsZXxlbnwwfHwwfHw%3D&w=1000&q=80",
    bio: "Travel fan. Hipster-friendly
tv scholar. Friendly communicator.
Coffee enthusiast.",
  },{  name: "Spencer",
.... .... .... .... ....
.... .... ....},];
```

Code Snippet 2

```
function App() {
  const users = USER_LIST;
  return (
    <div className="App">
      <UserList users={users} />
    </div>  );}
function UserList({ users }) {
  return (
    <div className="user-list-
container">
      {users.map((usr) => (
        <User name={usr.name}
pic={usr.pic} />
      ))}    </div> );}
```

Code Snippet 3

```
function User({ name, pic }) {
  return (
    <section className="user-
container">
      <img className="user-profile-
pic" src={pic} alt={name} />
      <h2 className="user-name">
        Hello, <span className="user-
first-name">{name}</span>
      </h2>
    </section>
  );}
```



USER_LIST and User Component

BASIC COMPONENTS OF REACTJS ROUTER [2-3]

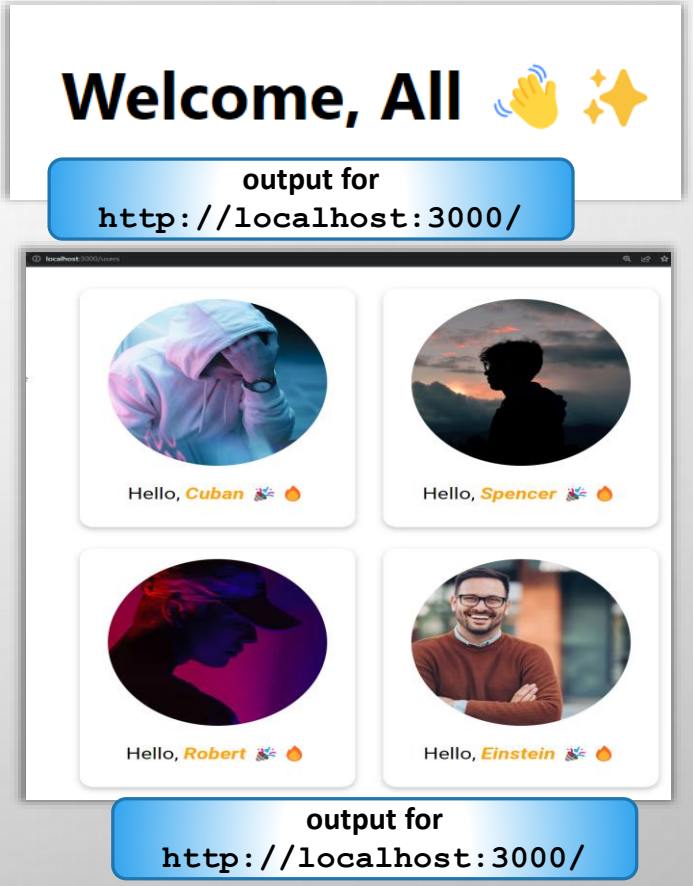
Router/ Browser Router → Route → Link → Nav

Code Snippet 4

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-
router-dom";
import "./index.css";
import App from "./App";
import reportWebVitals from
"./reportWebVitals";
const root =
ReactDOM.createRoot(document.getElementBy
d("root"));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

Code Snippet 5

```
import "./App.css";
import { Routes, Route, Link } from
"react-router-dom";
function App() {
  const users = USER_LIST;
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Home
/>} />
        <Route path="/users"
element={<UserList users={users} />}
/>
      </Routes>
    </div>
  );
}
export default App;
```



output for http://localhost:3000/

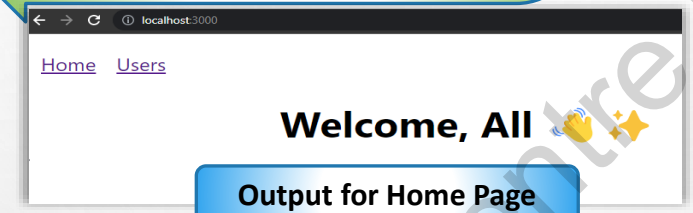
output for http://localhost:3000/

BASIC COMPONENTS OF REACTJS ROUTER [3-3]

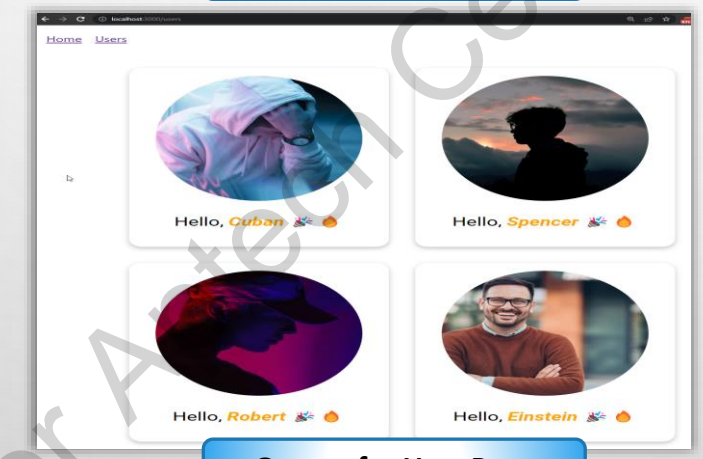
Router/ Browser Router → Route → Link → Nav

Code Snippet 6

```
function App() {
  const users = USER_LIST;
  return (
    <div className="App">
      <nav className="nav-list">
        <Link to="/">Home</Link>
        <Link
to="/users">Users</Link>
        {/* Not recommended */}
        <a href="/users">Users with
anchor</a>
      </nav>
      <Routes>
        <Route path="/"
element={<Home />} />
        <Route path="/users"
element={<UserList users={users} />}
/>
      </Routes>
    </div>
  );
}
```



Output for Home Page



Output for User Page

- useNavigate
  - Changes URL programmatically
- useParams
  - Passes parameters from URL to component

USING ROUTING MODULES [1-3]

path="/users/:id"

```
function App() {
  const users = USER_LIST;
  return (
    <div className="App">
      <nav className="nav-list">
        <Link to="/">Home</Link>
        <Link to="/users">Users</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/users"
element={<UserList users={users} />} />
        <Route path="/users/:id"
element={<UserDetail users={users} />} />
      </Routes>
    </div>  );}
```

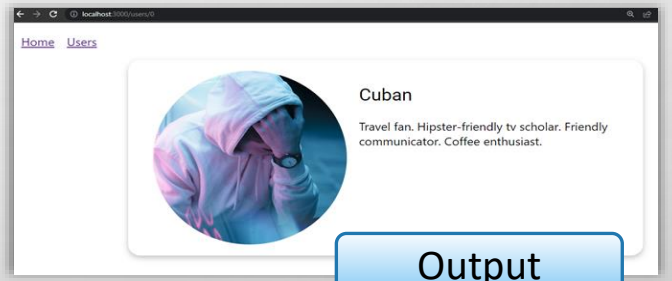
Code Snippet 7

URL Provided By the User

useParams Hook

```
import {
  Routes,
  Route,
  Link,
  useParams,
} from "react-router-dom";
function UserDetail({ users }) {
  const { id } = useParams();
  console.log(id);
  const user = users[id];
  return (
    <section className="user-detail-container">
      <img className="user-profile-pic"
src={user.pic} alt={user.name} />
      <div>   <h2 className="user-
name">{user.name}</h2>
        <p>{user.bio}</p>
      </div>    </section>  );}
```

Code Snippet 8



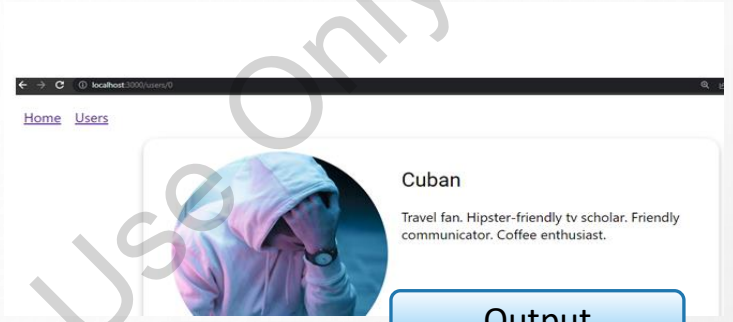
Output

USING ROUTING MODULES [2-3]

user `index` is passed as `id` prop

Code Snippet 9

```
function UserList({ users }) {
  return (
    <div className="user-list-container">
      {users.map((usr, index) => (
        <User name={usr.name} pic={usr.pic}
id={index} />
      ))}
    </div>
  );
}
```



Output

`onClick` event handler

Code Snippet 10

```
import {
  Routes,
  Route,
  Link,
  useNavigate,
  useParams,} from "react-router-dom";
function User({ name, pic, id }) {  const navigate =
useNavigate();  return (
    <section
      onClick={() => navigate(`/users/${id}`)}
      className="user-container"
    >      <img className="user-profile-pic"
src={pic} alt={name} />
      <h2 className="user-name">
        Hello, <span className="user-first-
name">{name}</span> 🎉🔥      </h2>    </section>);}
```

Code Snippet 11

```
<Route path="*" element={<NotFound />} />

function NotFound() {
  return (
    <div>
      <h1>404 Not found</h1>
    </div>
  );
}
```

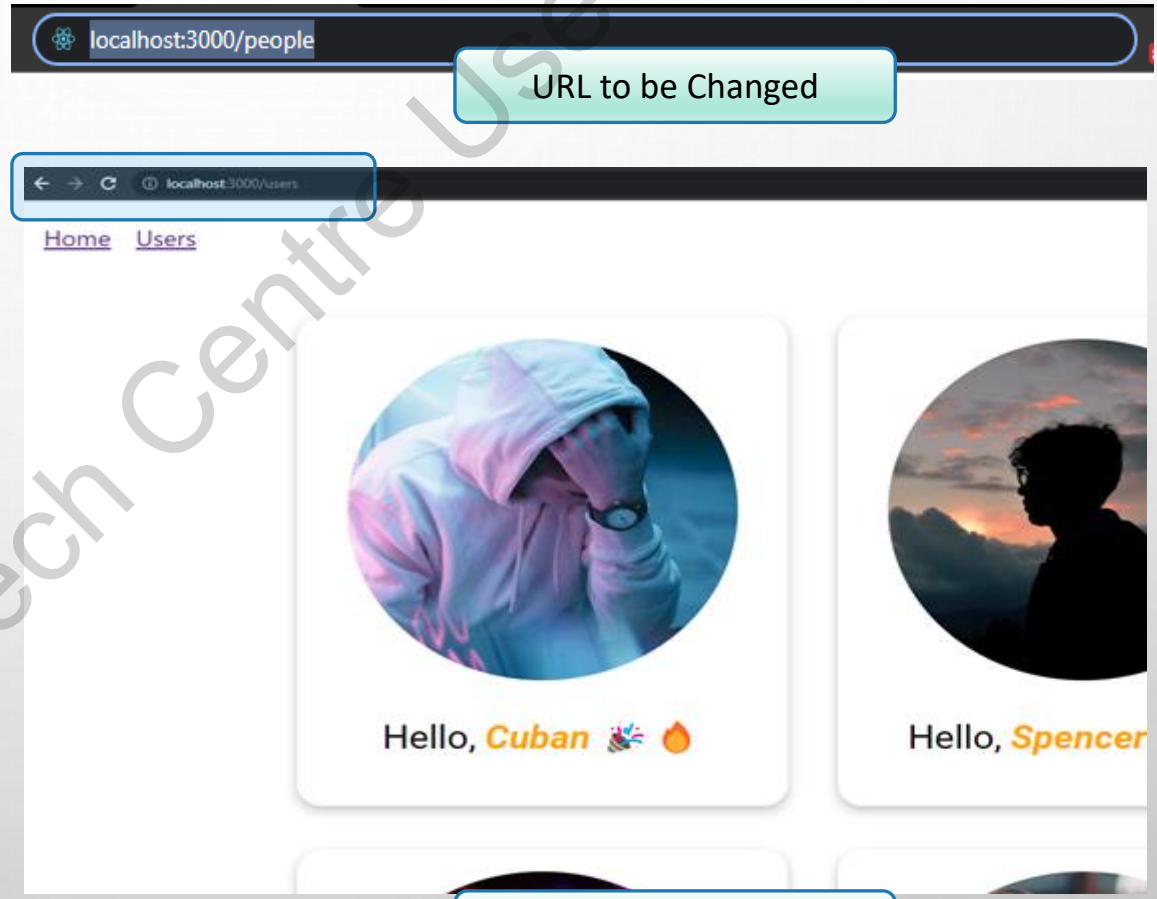
Output when user types a random URL

USING ROUTING MODULES [3-3]

Redirect to a new URL

```
 import {
  Routes,
  Route,
  Link,
  useNavigate,
  useParams,
} from "react-router-dom";
 <Route path="/people"
element={<Navigate replace
to="/users" />} />
```

Code Snippet 12

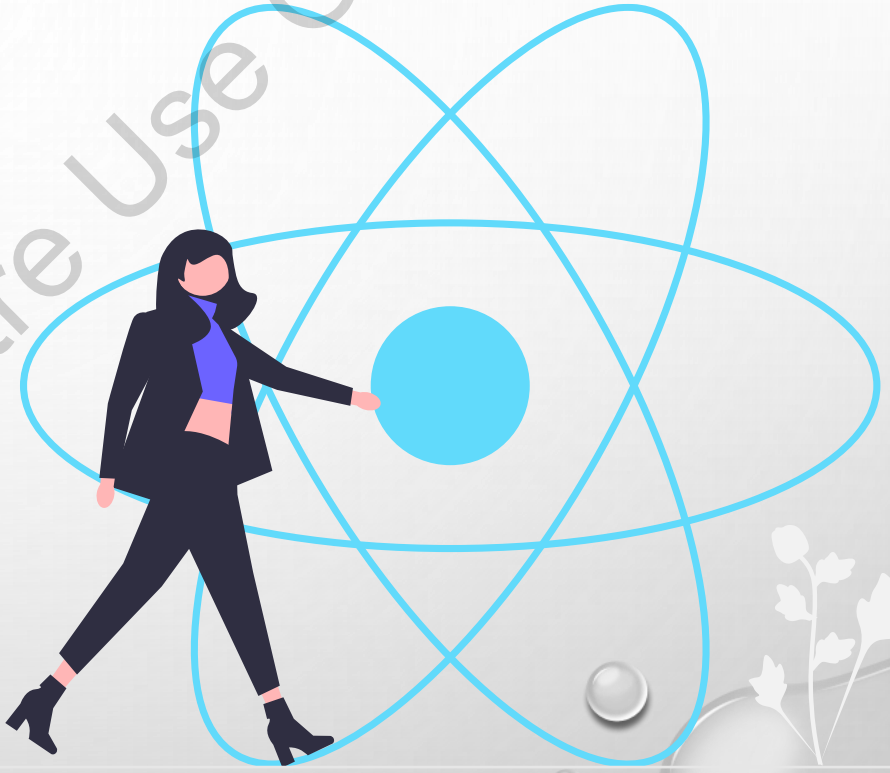


SUMMARY

- SPA – Single Page Application, which is popularized by ReactJS.
- SPA is fast and does not require refresh.
- Two types of routers are Browser and Hash.
- Link component causes a no refresh.
- `useNavigate` – Programmatically changes URL.
- `useParams` – Extracts the value from the URL.
- `path=*` for matching 404 page.
- Redirection is achieved by the Navigate component.

SESSION 06

STYLING REACT ELEMENTS

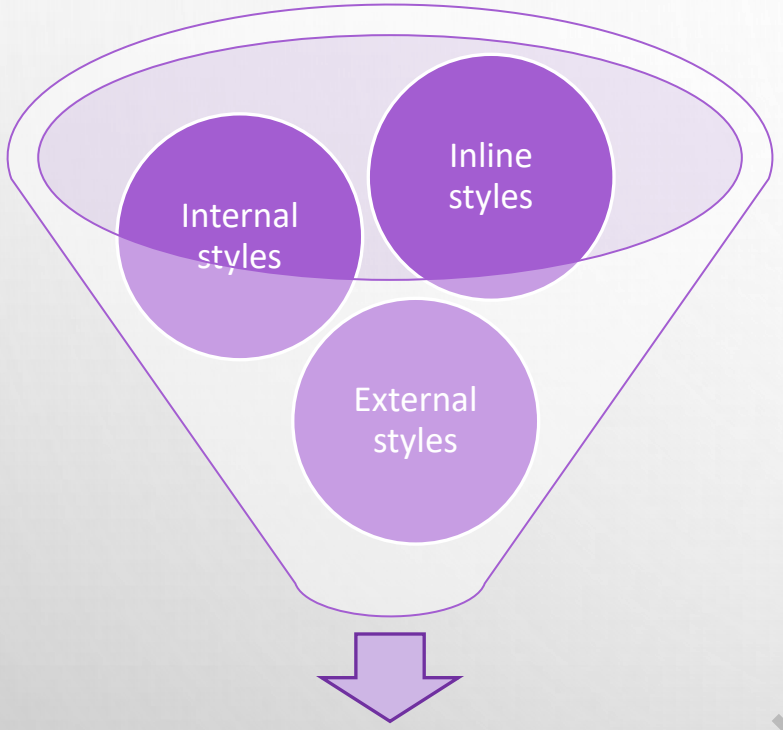


OBJECTIVES

In this session, the topics discussed are as follows:

- Explain inline, internal, and external styles
- Explain different styling methods in ReactJS
- Illustrate the usage of different styling methods

APPLYING BASIC STYLE TO HTML ELEMENTS [1-2]



Styling in HTML

Inline Styles

Code Snippet 1

```
<html>
<head>
  <title>There ways of styling</title>
  <meta charset="UTF-8" />
</head>
<body>
  <img
    style="
      height: 200px;
      aspect-ratio: 1/1;
      border-radius: 50%;
      object-fit: cover;"
src="https://images.pexels.com/photos/1704488/pexels-photo1704488.jpeg?auto=compress&cs=tinysrgb&dpr=1&w=500"
    alt="Mark"
  />  </body> </html>
```

Output

APPLYING BASIC STYLE TO HTML ELEMENTS [2-2]

Internal Styles

Code Snippet 2

```
<!DOCTYPE html>
<html>
  <head>
    <title>There ways of styling</title>
    <meta charset="UTF-8" />
    <style>
      .profile-pic {
        height: 200px;
        aspect-ratio: 1/1;
        border-radius: 50%;
        object-fit: cover;
      }
    </style>
  </head>
<body>
    <img
      class="profile-pic"

src="https://images.pexels.com/photos/170448
8/pexels-photo-
1704488.jpeg?auto=compress&cs=tinysrgb&dpr=1
&w=500"
      alt="Mark"
    />
  </body>
</html>
```

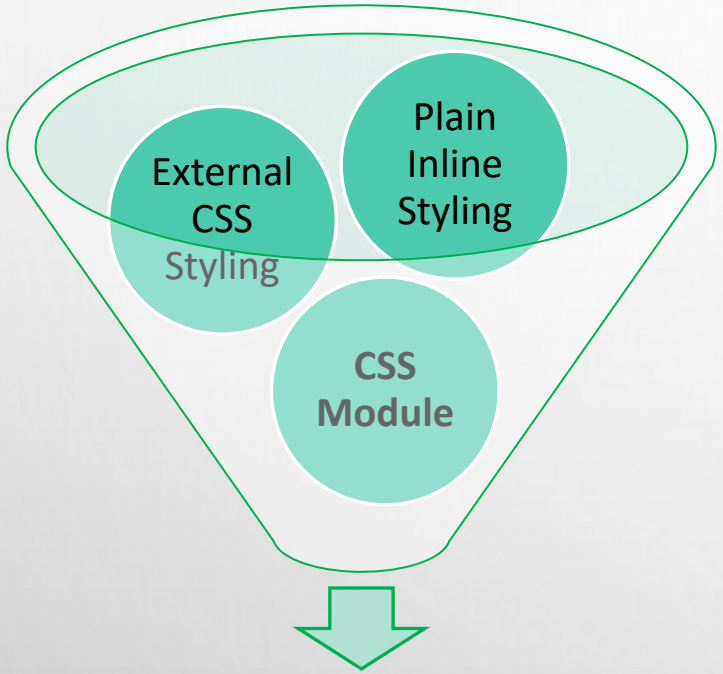
External Styles

Code Snippet 3

```
<html>
  <head>
    <title>There ways of styling</title>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="./style.css"
/>
  </head>
  <body>
    <img
      class="profile-pic"

src="https://images.pexels.com/photos/1704488/
pexels-photo-
1704488.jpeg?auto=compress&cs=tinysrgb&dpr=1&w
=500"
      alt="Mark"
    />
  </body>
</html>
```

```
.profile-pic {
  height: 200px;
  aspect-ratio: 1/1;
  border-radius: 50%;
  object-fit: cover;
}
```

DIFFERENT STYLING METHODS IN REACTJS [1-6]



Styling Methods in ReactJS

Code Snippet 4: Erroneous Styling Method

```
function UserPic() {
  return (
    <div>
      <img
        style="
        height: 200px;
        aspect-ratio: 1/1;
        border-radius: 50%;
        object-fit: cover;
      " src="#"
        alt="Mark"
      />
    </div>
  );}
```

```
}
function App() {
  return (
    <div className="App">
      <UserPic />
    </div>
  );}
```

```
Uncaught Error: The `style` prop expects a mapping from style properties   react-dom.development.js:27507
to values, not a string. For example, style={{marginRight: spacing + 'em'}} when using JSX.
    at assertValidProps (react-dom.development.js:3451:1)
    at setInitialProperties (react-dom.development.js:10493:1)
    at finalizeInitialChildren (react-dom.development.js:11482:1)
    at completeWork (react-dom.development.js:22769:1)
    at completeUnitOfWork (react-dom.development.js:27169:1)
    at performUnitOfWork (react-dom.development.js:27139:1)
    at workLoopSync (react-dom.development.js:27034:1)
    at renderRootSync (react-dom.development.js:27002:1)
    at recoverFromConcurrentError (react-dom.development.js:26406:1)
    at performConcurrentWorkOnRoot (react-dom.development.js:26307:1)
```

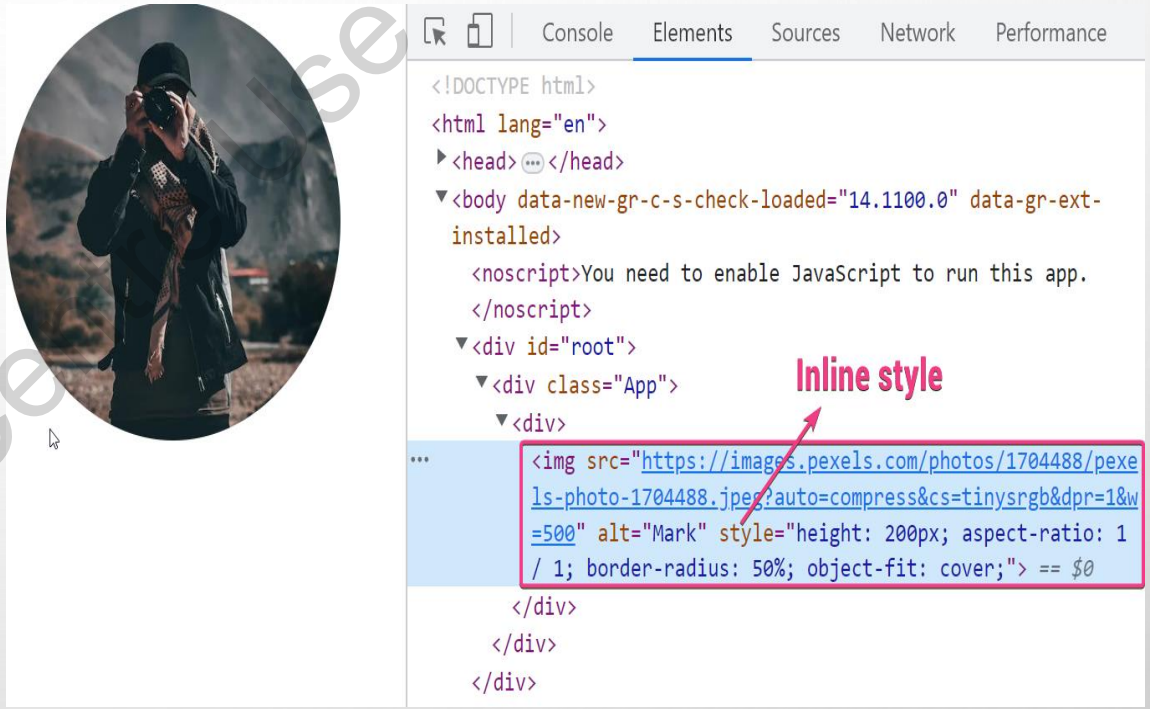
Error Output

DIFFERENT STYLING METHODS IN REACTJS [2-6]

Inline Styles

Code Snippet 5

```
function UserPic() {
  const styles = {
    height: "200px",
    aspectRatio: "1/1",
    borderRadius: "50%",
    objectFit: "cover",
  };
  return (
    <div>
      <img
        style={styles}
        src="#"
        alt="Mark"
      />
    </div>
  );
}
function App() {
  return (
    <div className="App">
      <UserPic />
    </div>
  );
}
```



Output

DIFFERENT STYLING METHODS IN REACTJS [3-6]

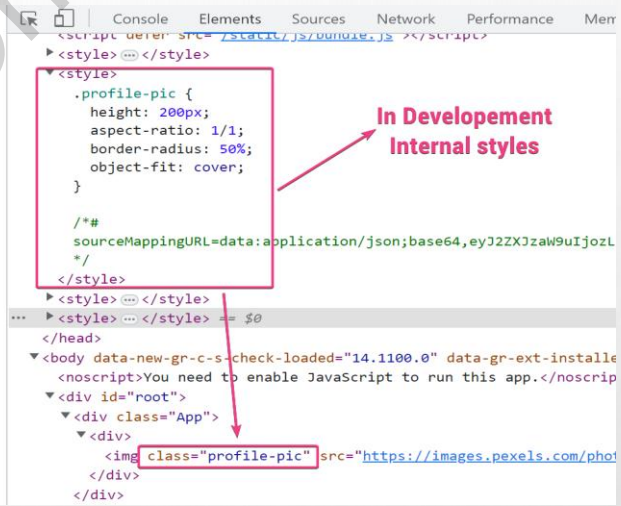
External CSS

Code Snippet 6

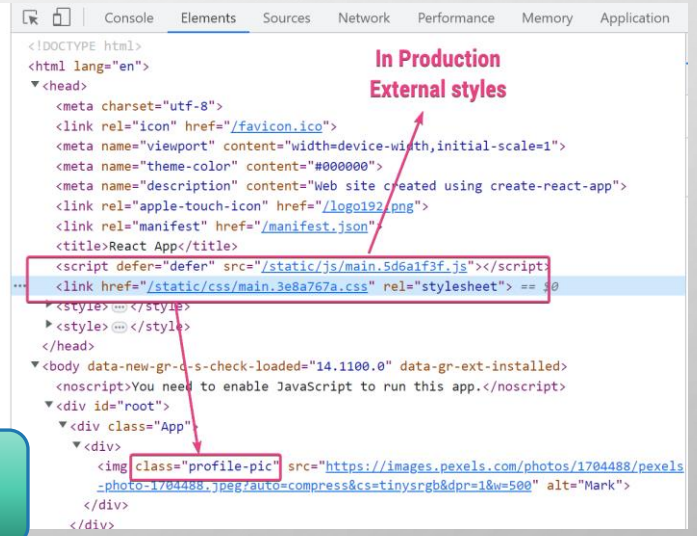
```
import "./App.css";
function UserPic() {
  return (
    <div>
      <img
        className="profile-pic"

src="https://images.pexels.com/photos/17
04488/pexels-photo-
1704488.jpeg?auto=compress&cs=tinysrgb&d
pr=1&w=500"
        alt="Mark"
      />
    </div>
  );
}
```

```
.profile-pic {
  height: 200px;
  aspect-ratio: 1/1;
  border-radius: 50%;
  object-fit: cover;
}
```

Output when in development



Output when in production

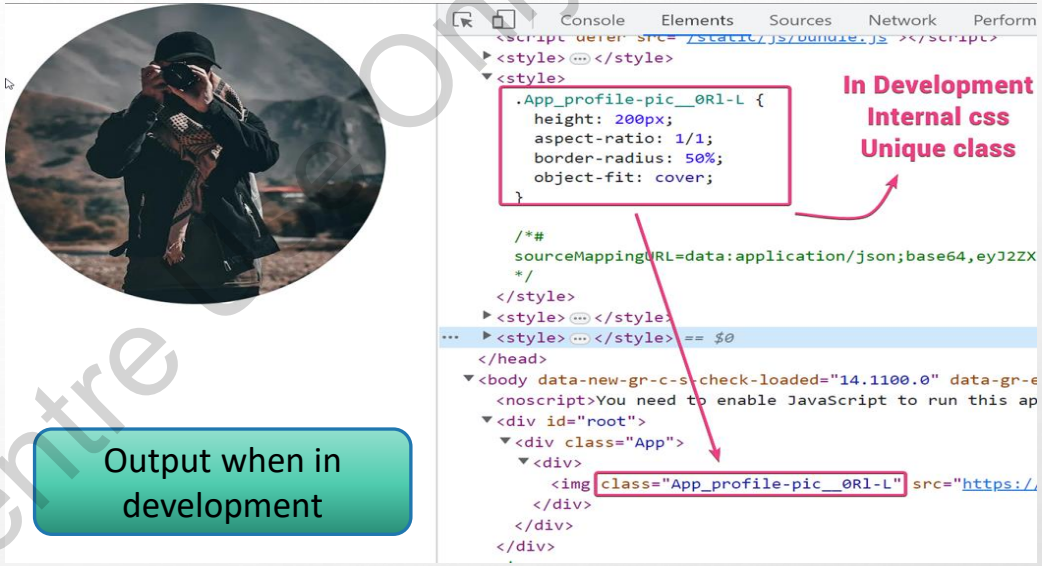


DIFFERENT STYLING METHODS IN REACTJS [4-6]

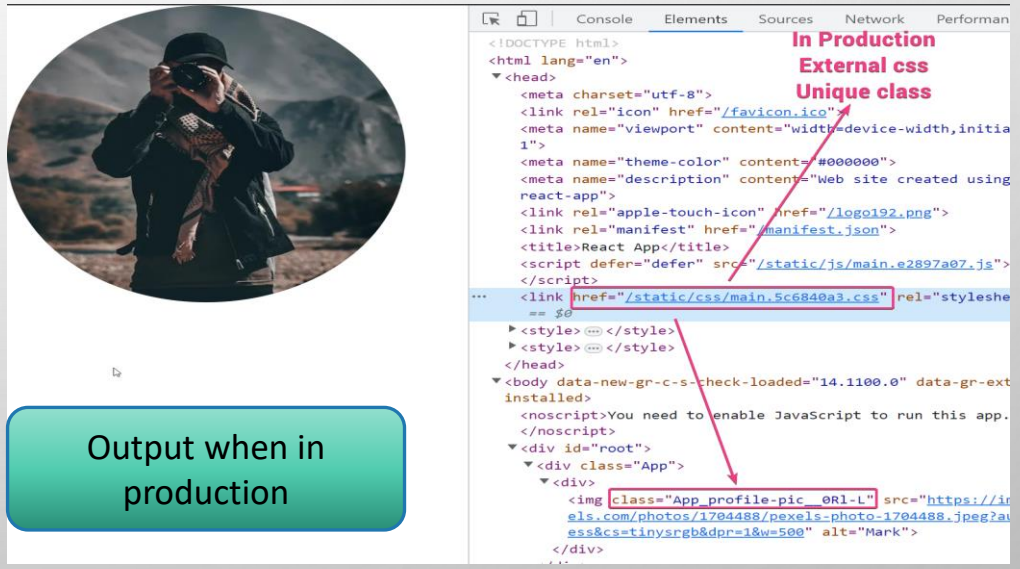
CSS Module

Code Snippet 7

```
import styles from "./App.module.css";
function UserPic() {
  return (
    <div>
      <img
      className={styles["profile-pic"]}
src="https://images.pexels.com/photos/1704488/pexels-photo-1704488.jpeg?auto=compress&cs=tinysrgb&dpr=1&w=500"
        alt="Mark"
      />
    </div>
  );
}
.profile-pic {
  height: 200px;
  aspect-ratio: 1/1;
  border-radius: 50%;
  object-fit: cover;
}
```



Output when in development



Output when in production

DIFFERENT STYLING METHODS IN REACTJS [5-6]

External Fonts

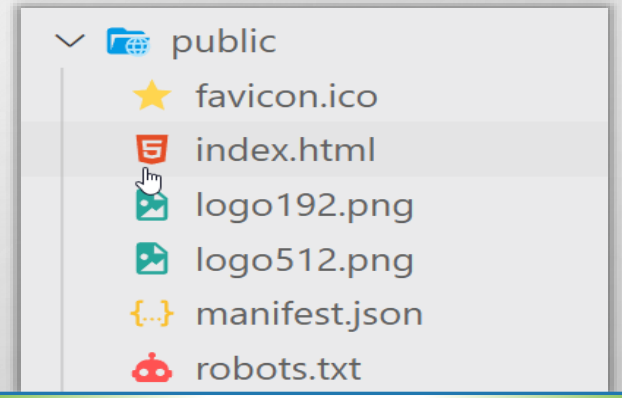
Visit https://fonts.google.com/

Choose the required font

Code Snippet 8

```
<link rel="preconnect"
href="https://fonts.googleapis.com" />
    <link rel="preconnect"
href="https://fonts.gstatic.com"
crossorigin />
    <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700;900&display=swap"
      rel="stylesheet"    />
```

Copy the <link> tag



Navigate to the public/index.html file

```
    Notice the use of %PUBLIC_URL% in the tags above.
    It will be replaced with the URL of the `public` folder during the build.
    Only files inside the `public` folder can be referenced from the HTML.

    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a non-root public URL.
    Learn how to configure a non-root public URL by running `npm run build`.
  -->
  <title>React App</title>
  <link rel="preconnect" href="https://fonts.googleapis.com" />
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
  <link
    href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700;900&display=swap"
    rel="stylesheet"
  />
</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
```

Adding the Link tags

```
body{

  font-family:
'Roboto', sans-
serif;

}
```

Font-family is ready to be utilized in the app

DIFFERENT STYLING METHODS IN REACTJS [6-6]

Bootstrap Icons

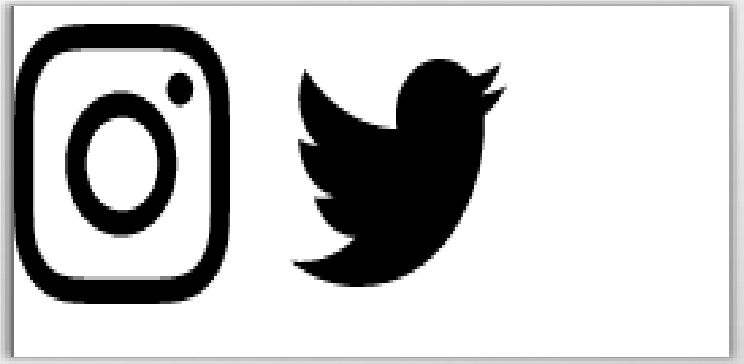
- Install the `react-bootstrap-icons` through the command line
- `npm i react-bootstrap-icons`

Installation

Import Icons

- Import the necessary icons and call the component

```
import { Instagram, Twitter } from "react-bootstrap-icons";
function App() {
return (
<div className="App">
<Instagram /> <Twitter />
</div>);}
```



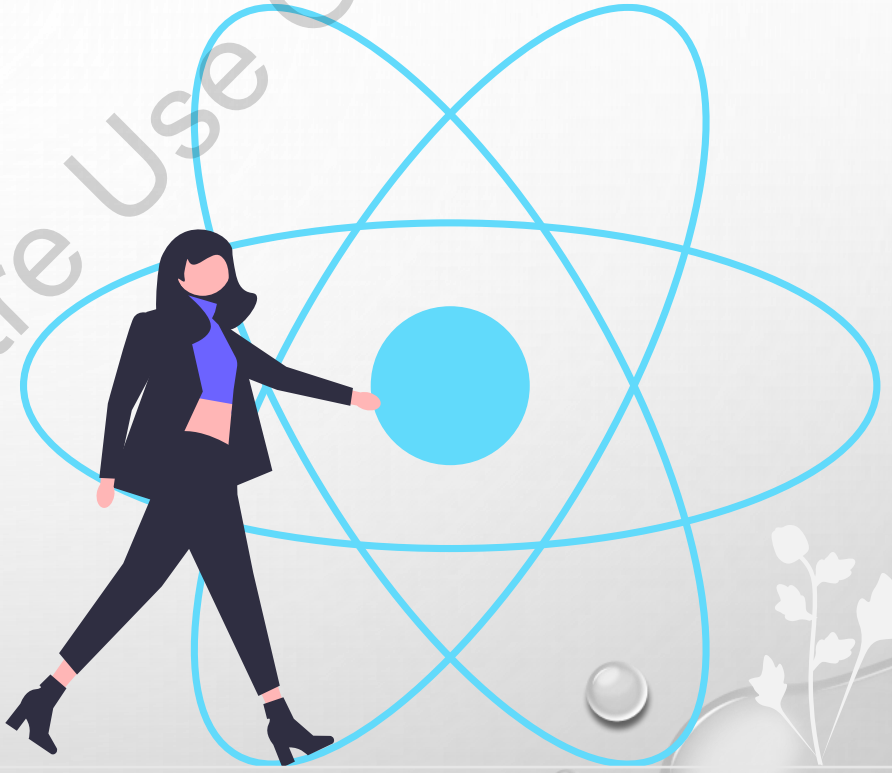
Output

SUMMARY

- Three different types of styling in HTML are:
  - Inline.
  - Internal.
  - External.
- Three different types of styling in ReactJS are:
  - Using styled objects.
  - External CSS.
  - CSS modules.
- External CSS is always recommended.
- Inline styles are generally used in ReactJS when the style requires to be changed dynamically.
- CSS modules always provide unique style class names.
- Icons can easily be imported as components from the `react-bootstrap-icons`.
- Fonts can be directly added to the `<head>` tag in index.html.

SESSION 07

REACTJS COMPONENT LIFECYCLE

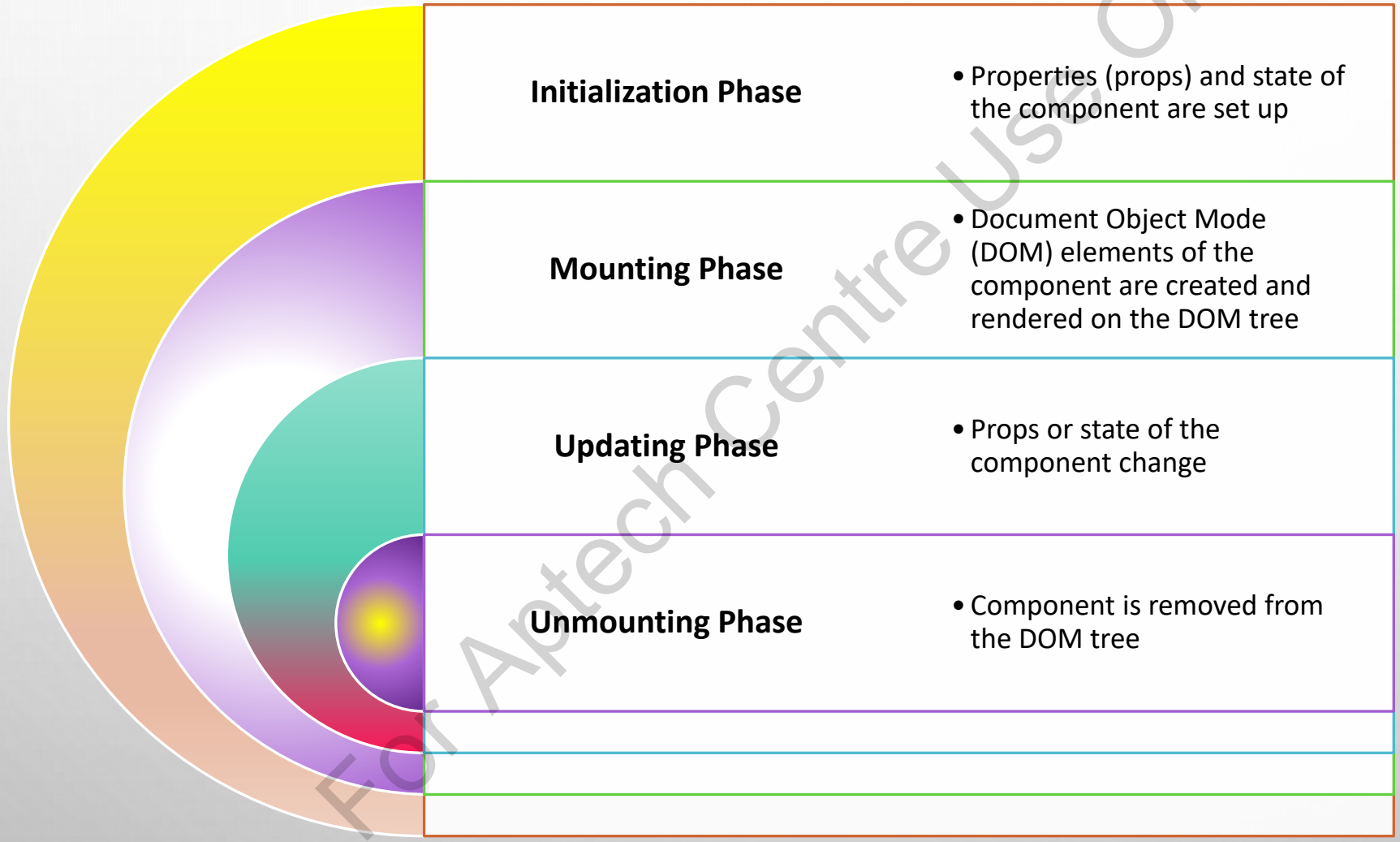


OBJECTIVES

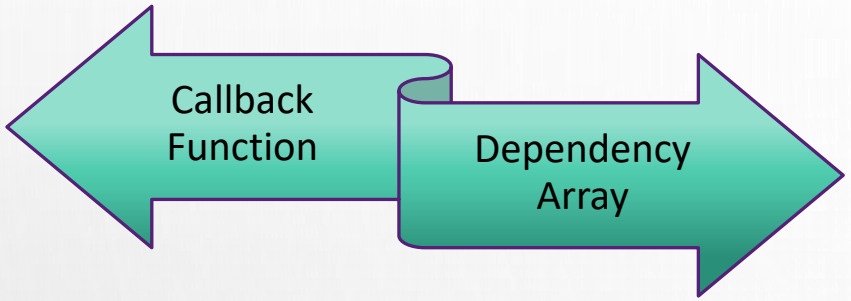
In this session, the topics discussed are as follows:

- Explain different phases of the ReactJS component
- Explain how to manage lifecycles using the useEffect hook
- Illustrate the implementation of useEffect in apt scenarios

DIFFERENT PHASES OF REACTJS COMPONENTS



MANAGING LIFECYCLE USING THE USEEFFECT HOOK



Syntax

```
useEffect(callback function, dependency array);
```

Empty Dependency Array

```
useEffect(() => {}, []);
```

No Dependency Array

```
useEffect(() => {});
```

Values in Dependency Array

```
useEffect(() => {}, [v1, v2]);
```

Return in `useEffect`

```
useEffect(() => {
return () => {};
}, []);
```

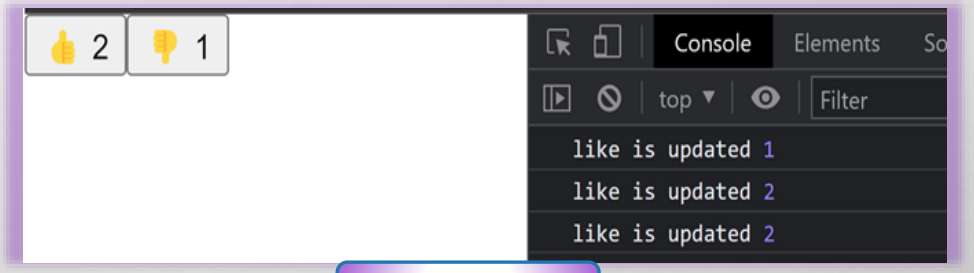
IMPLEMENTING USEEFFECT IN APT SCENARIOS [1-3]

Code Snippet 1: Testing the app without strictmode

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import { BrowserRouter } from "react-router-dom";
const root =
ReactDOM.createRoot(document.getElementById("root"
));
root.render(
  // <React.StrictMode>
  <BrowserRouter>
    <App />
  </BrowserRouter>
  // </React.StrictMode>
);
// 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();
```

Code Snippet 2: No dependency array given

```
function Counter() {
  let [like, setLike] = useState(0);
  let [dislike, setDisLike] = useState(0);
  useEffect(() => {
    console.log("like is updated", like);
  });
  return (
    <div>
      <button onClick={() => setLike(like +
1)}>👍 {like}</button>
      <button onClick={() =>
setDisLike(dislike + 1)}>👎
{dislike}</button>
    </div>  );}
```

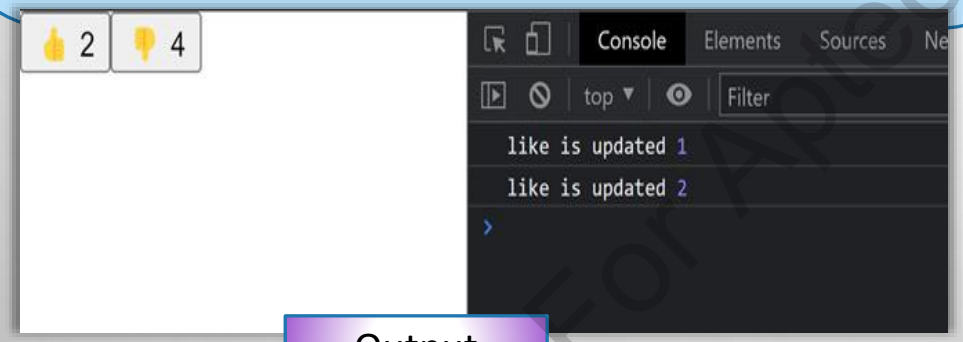


Output

IMPLEMENTING USEEFFECT IN APT SCENARIOS [2-3]

Code Snippet 3: Dependency array with 'like'

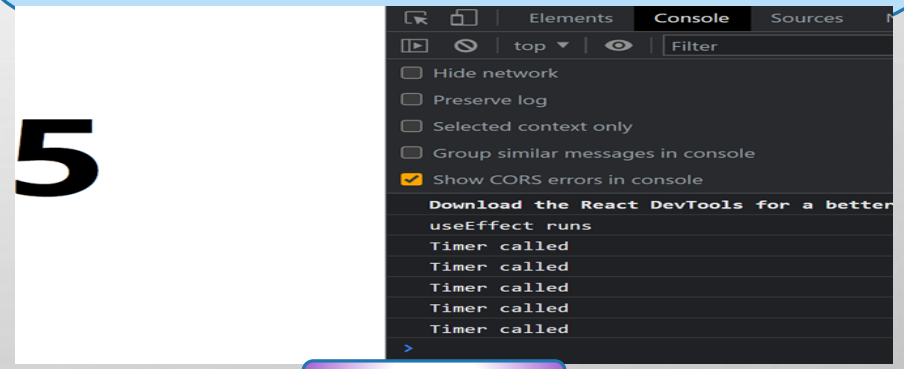
```
function Counter() {
  let [like, setLike] = useState(0);
  let [dislike, setDisLike] = useState(0);
  useEffect(() => {
    console.log("like is updated", like);
  }, [like]);
  return (
    <div>
      <button onClick={() => setLike(like + 1)}>👍
{like}</button>
      <button onClick={() => setDisLike(dislike +
1)}>👎 {dislike}</button>
    </div>
  );
}
```



Output

Code Snippet 4: setTime is called

```
function Timer() {
  const [time, setTime] = useState(0);
  useEffect(() => {
    console.log("useEffect runs");
    setInterval(() => {
      console.log("Timer called");
      setTime((prev) => prev + 1);
    }, 1000); }, []);
  return (
    <div> <h1>{time}</h1></div>  );}
function App() {
  return (
    <div className="App">
      <Timer />
    </div>  );}
```



Output

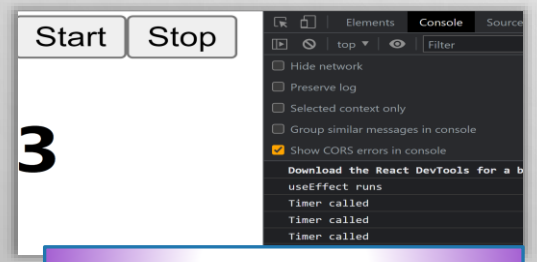
IMPLEMENTING USEEFFECT IN APT SCENARIOS [3-3]

Code Snippet 5: Dependency array with 'like'

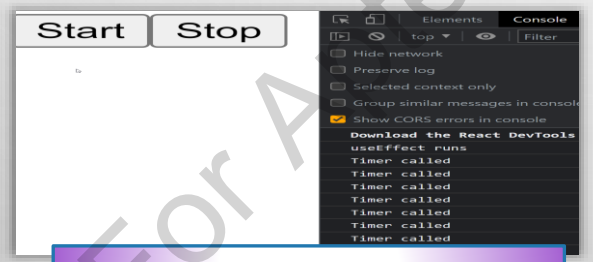
```
function App() {
  const [show, setShow] = useState(false);
  return (
    <div className="App">
      <button onClick={() =>
setShow(true)}>Start</button>
      <button onClick={() =>
setShow(false)}>Stop</button>
      {show ? <Timer /> : null}
    </div>
  );
}
```

Code Snippet 6: setTime is called

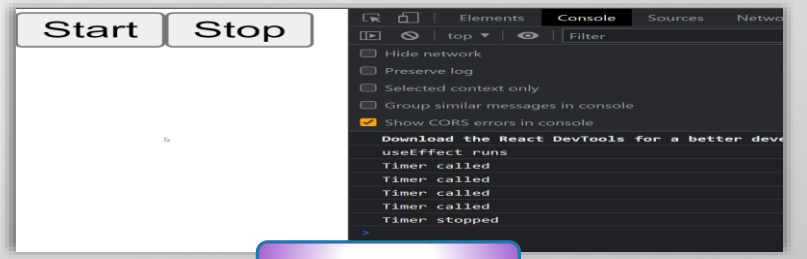
```
function Timer() {
  const [time, setTime] = useState(0);
  useEffect(() => {
    console.log("useEffect runs");
    const timer = setInterval(() => {
      console.log("Timer called");
      setTime((prev) => prev + 1);
    }, 1000);
    return () => {
      console.log("Timer stopped");
      clearInterval(timer);
    };  }, []);
  return (
    <div>
      <h1>{time}</h1>
    </div>  );}
```



Output with timer shown



Output with timer not shown



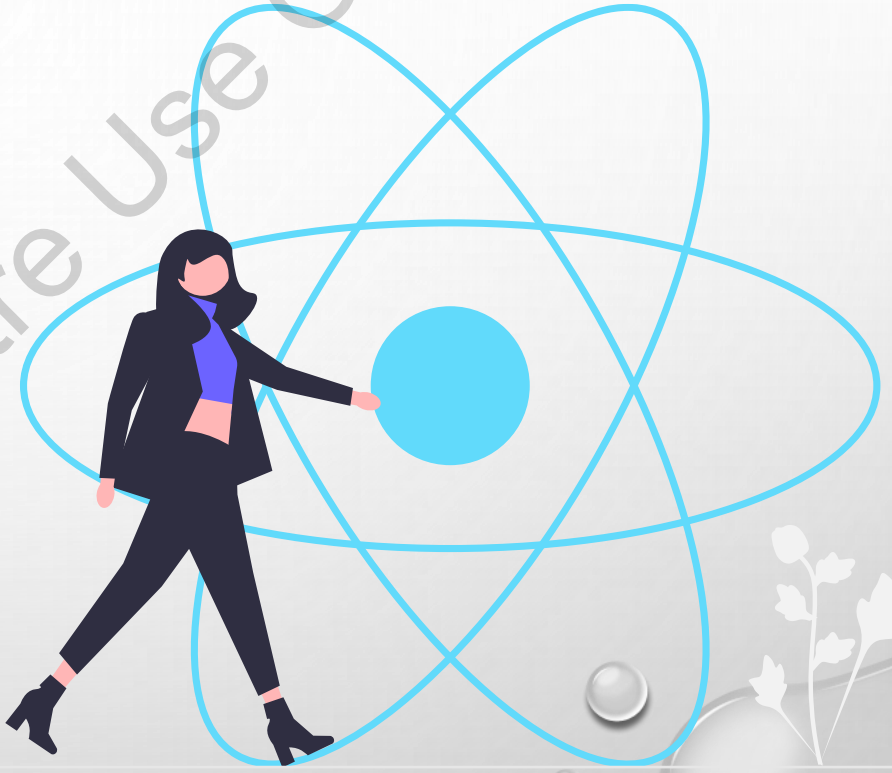
Output

SUMMARY

- Four phases of the lifecycle of ReactJS components are:
  - Initialization.
  - Mounting.
  - Updating.
  - Unmounting.
- Using the `useEffect` hook is an efficient way to handle the lifecycle phases of the component.
- Dependency array dictates in which phase of the component the callback function will be called.
- Conditional Rendering is a technique where based on a condition, a component is shown or removed from the DOM.
- Memory leaks are required to be handled in the clean part of `useEffect`.

SESSION 08

CREATING A CRUD APPLICATION USING REACTJS FORMS

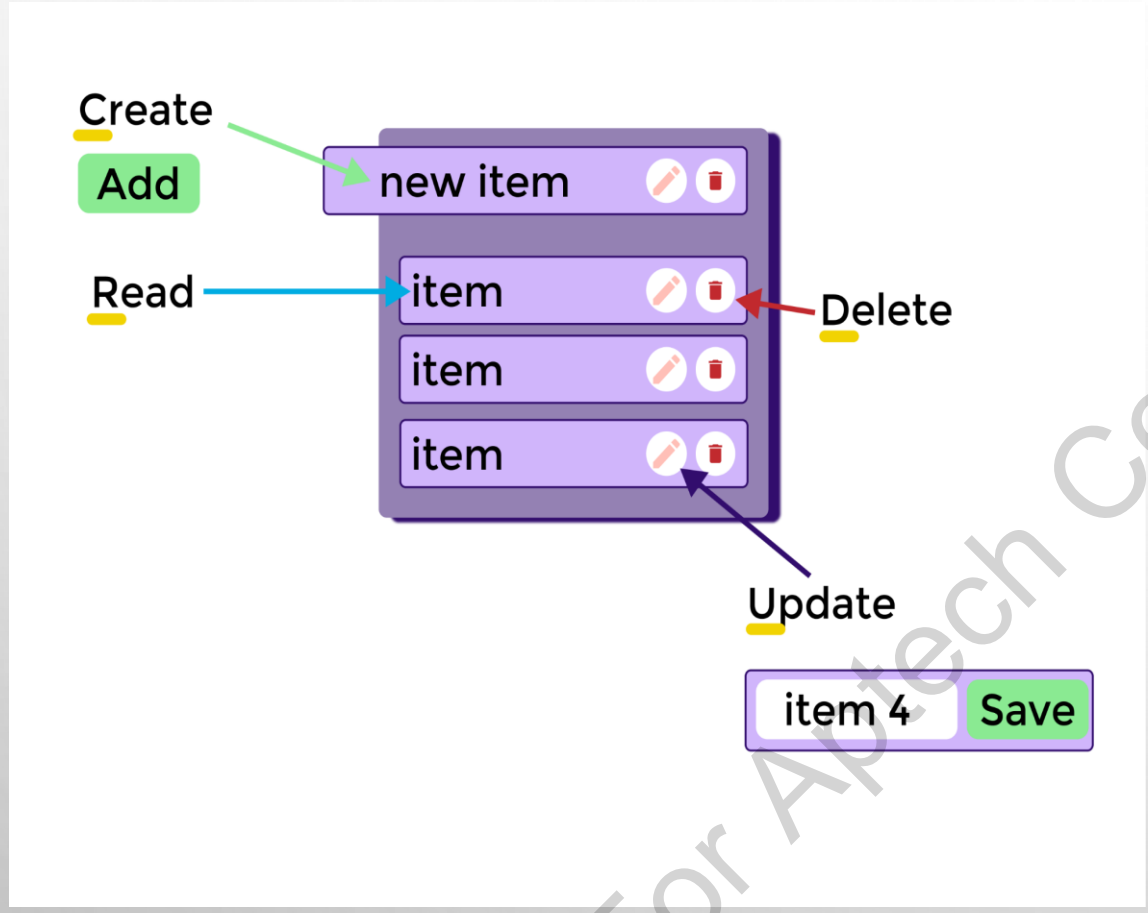


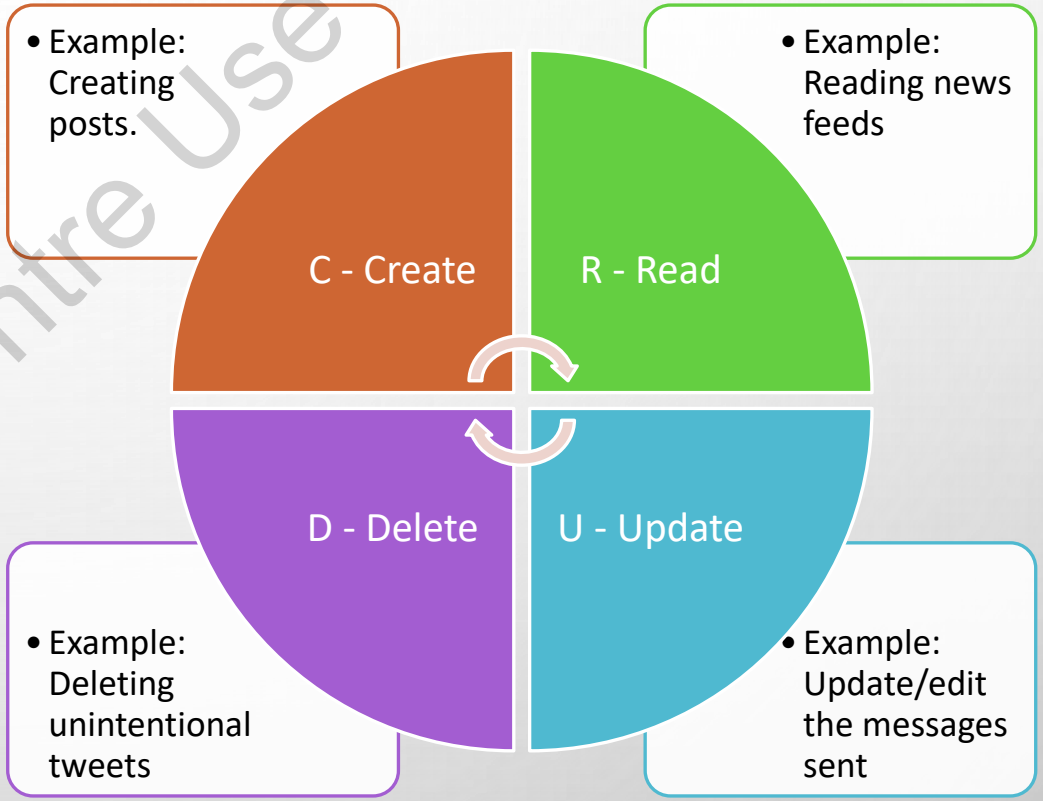
OBJECTIVES

In this session, the topics discussed are as follows:

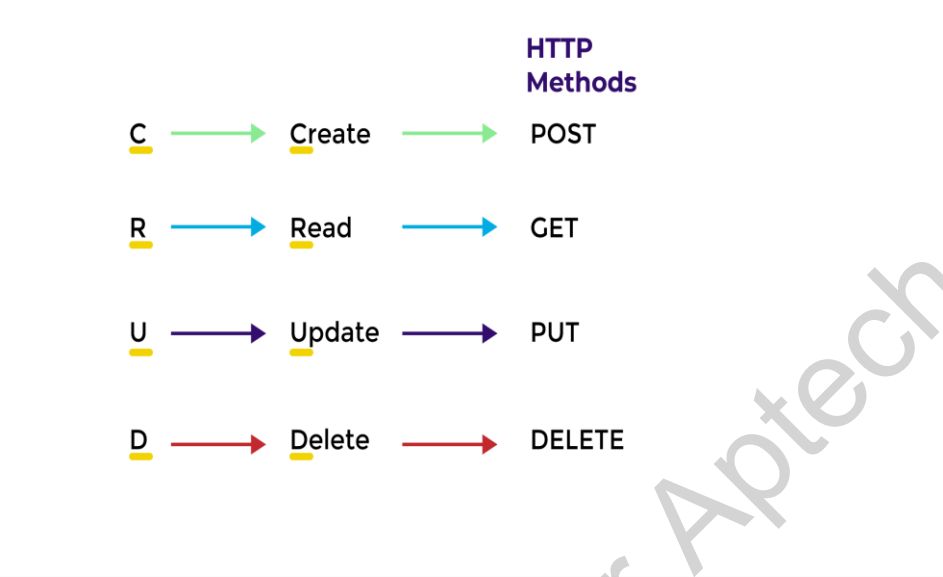
- Explain CRUD
- Illustrate the development of a simple CRUD application using ReactJS forms
- Illustrate the development of a complex CRUD application using ReactJS forms

WHAT IS CRUD?

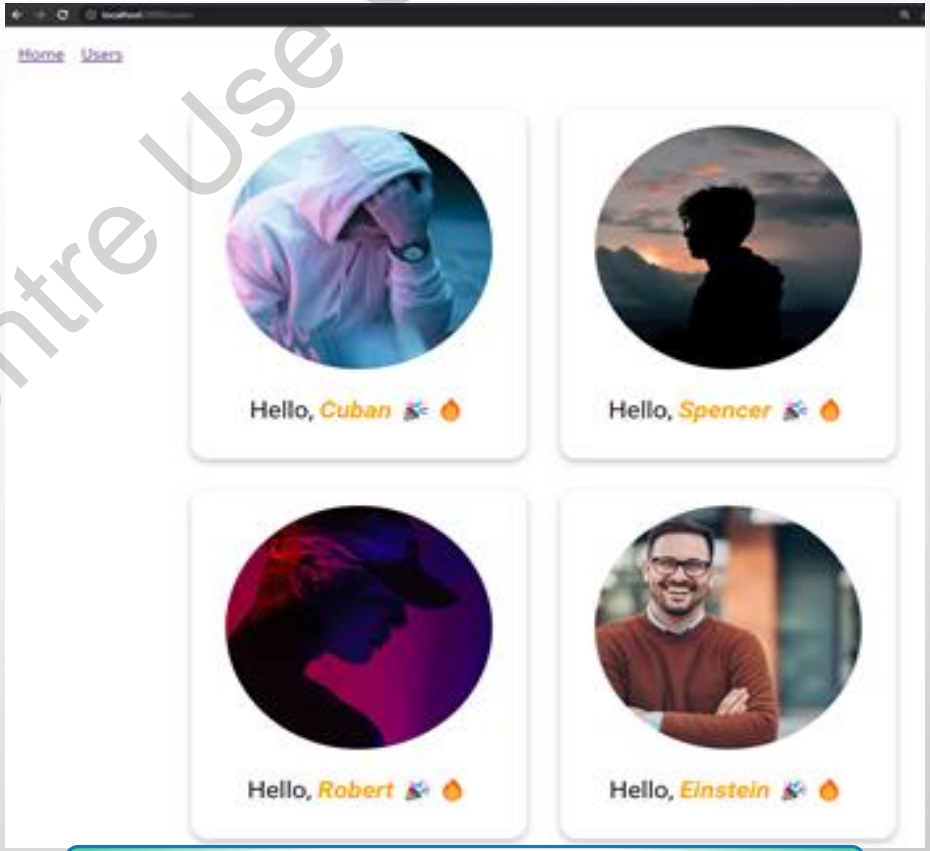




CREATING A SIMPLE CRUD APPLICATION



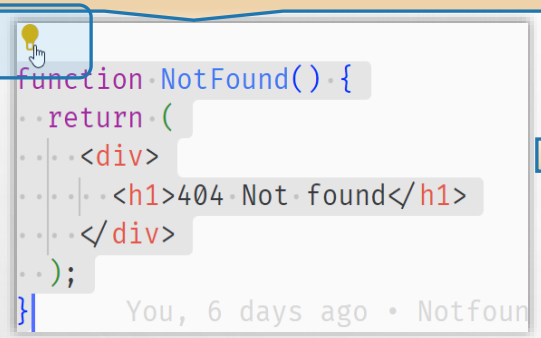
Methods Used by CRUD Operations

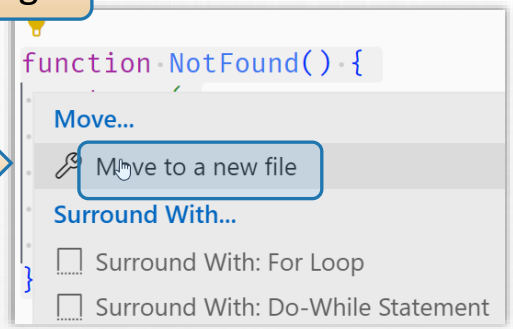


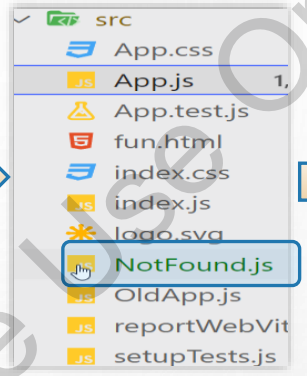
Output of session8_starter.zip File

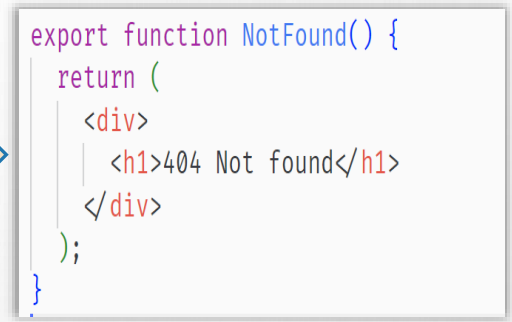
CREATING A SIMPLE CRUD APPLICATION [1-5]

Displaying user data on the Web page










```
<Route path="/users" element={<UserList
/>} />

import { User } from "./User";
export function UserList() {
  const users = [];
  return (
    <div className="user-list-container">
      {users.map((usr, index) => (
        <User name={usr.name}
pic={usr.pic} id={index} key={index} />
      ))}
    </div>  );}
```

Code Snippet 1


```
export function UserList() {
  const users = [];

fetch("https://640fc234864814e5b63f0d2f.mockapi.
io/users")
    .then((data) => data.json())
    .then((userList) => console.log(userList));
  return (
    <div className="user-list-container">
      {users.map((usr, index) => (
        <User name={usr.name} pic={usr.pic}
id={index} key={index} />
      ))}
    </div>  );}
```

Code Snippet 2

0: {id: '99', name: 'Cuban', pic: 'https://images.unsplash.com/photo-1618641986557-1e…WFyY2h8NXx8cHJvZmlsZXxlbnwwfHwwfHw%3D&w=1000&q=80', bio: 'Travel fan. Hipster-friendly tv scholar. Friendly communicator. Coffe
1: {id: '100', name: 'Spencer', pic: 'https://images.unsplash.com/photo-1529665253569-6d…WFyY2h8NHx8cHJvZmlsZXxlbnwwfHwwfHw%3D&w=1000&q=80', bio: 'Award-winning web lover. Thinker. Social media advocate. Creator.
2: {id: '101', name: 'Robert', pic: 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcT2C_xadF4WT19MkU5PpYyU8njyMgMIuttwXQ&usqp=CAU', bio: 'Professional communicator. Travel scholar. Friendly music junkie. Ha
3: {id: '102', name: 'Einstein', pic: 'https://media.istockphoto.com/id/1179420343/photo/…20&c=8l-gOboGEFSyCFXr09EguDmV0E0bFT5usAms1wyFBh8=', bio: 'Typical travel guru. Friendly entrepreneur. Zombie expert. Thinke

CREATING A SIMPLE CRUD APPLICATION [2-5]

User Details Printed

Code Snippet 3

```
export function UserList() {
  const [users, setUsers] = useState([]);
  useEffect(() => {

fetch("https://640fc234864814e5b63f0d2f.m
ockapi.io/users")
      .then((data) => data.json())
      .then((userList) =>
setUsers(userList));
  }, []);
  return (
    <div className="user-list-container">
      {users.map((usr) => (
        <User name={usr.name}
pic={usr.pic} id={usr.id} key={usr.id} />
      ))}
    </div>
  );
}
```

UserList.js

Code Snippet 4

```
import { useNavigate } from "react-
router-dom";
export function User({ name, pic, id })
{
  const navigate = useNavigate();
  return (
    <section
      onClick={() =>
navigate(`/users/${id}`)}
      className="user-container"
    >
      <img className="user-profile-pic"
src={pic} alt={name} />
      <h2 className="user-name">
        Hello, <span className="user-
first-name">{name}</span> 🎉👌
      </h2>
    </section>
  );
}
```

User.js

Code Snippet 5

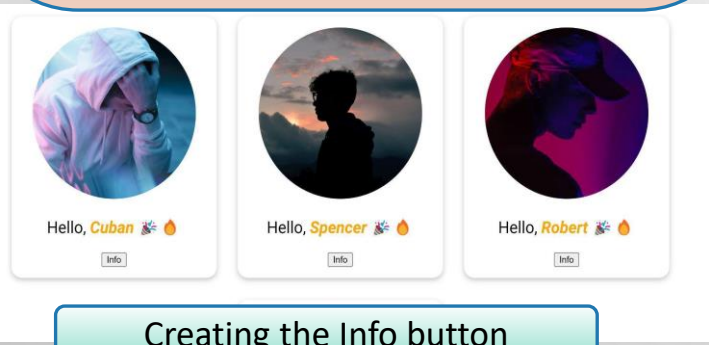
```
export function UserDetail() {
  const { id } = useParams();
  console.log(id);
  // const user = users[id];
  const [user, setUser] = useState({});
  useEffect(() => {

fetch(`https://640fc234864814e5b63f0d2f.mockapi
.io/users/${id}`)
      .then((data) => data.json())
      .then((userInfo) => setUser(userInfo));
  }, []);
  return (
    <section className="user-detail-container">
      <img className="user-profile-pic"
src={user.pic} alt={user.name} />
      <div>
        <h2 className="user-
name">{user.name}</h2>
        <p>{user.bio}</p>
      </div>
    </sectio
```

UserDetail.js

CREATING A SIMPLE CRUD APPLICATION [3-5]

Code Snippet 6

```
export function User({ name, pic, id }) {
  const navigate = useNavigate();
  return (
    <section className="user-container">
      <img className="user-profile-pic"
src={pic} alt={name} />
      <h2 className="user-name">
        Hello, <span className="user-
first-name">{name}</span> 🎉🔥
      </h2>
      <button onClick={() =>
navigate(`/users/${id}`)}>Info</button>
    </section>
  );
}
```



Creating the Info button

Code Snippet 7

```
export function UserList() {
  const [users, setUsers] = useState([]);
  useEffect(() => {
 ……..
        <User
          name={usr.name}
          pic={usr.pic}
          id={usr.id}
          key={usr.id}
deleteButton={<button>Delete</button>}
        />
      ))}    </div>  );}
```

JSX is passed to `deleteButton` prop

Code Snippet 8

```
export function User({ name, pic, id,
deleteButton }) {
  ……..
      </h2>
      <button onClick={() =>
navigate(`/users/${id}`)}>Info</button>
      {deleteButton}
    </section>
  );
}
```

`deleteButton` is received as props

Code Snippet 9

```
export function UserList() {
  const [users, setUsers] = useState([]);
  const getUsers = () => {

fetch("https://640fc234864814e5b63f0d2f.m
ockapi.io/users")
      .then((data) => data.json())
      .then((userList) =>
setUsers(userList));
  };
  useEffect(() => getUsers(), []);
  const deleteUser = (id) => {
fetch(`https://640fc234864814e5b63f0d2f.m
ockapi.io/users/${id}`, {
      method: "DELETE",
    })
      .then((data) => data.json())
      .then(() => getUsers());
  };
  return (
    <div className="user-list-container">
      {users.map((usr, index) => (
        <User     name={usr.name}
pic={usr.pic}     id={usr.id}
          key={index}
          deleteButton={
            <button onClick={() =>
deleteUser(usr.id)}>Delete</button>
          }  />  ))} </div> );}
```

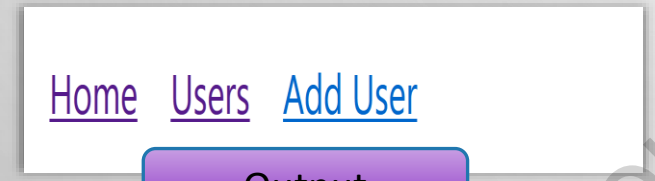
Call the API

CREATING A SIMPLE CRUD APPLICATION [4-5]

Creating a Form for Adding User

```
function App() {
  return (
    <div className="App">
      <nav className="nav-list">
          …
        <Link to="/users/add">Add
User</Link>
      </nav>
      <Routes>
          …
        <Route path="/users/add"
element={<AddUser />} />
      </Routes>
    </div>
  );
}
```

Code Snippet 10



Output

Creating three input fields

```
export function AddUser() {
  return (
    <div className="add-user-form">
      <input type="text"
placeholder="Name" />
      <input type="text"
placeholder="Pic" />
      <input type="text"
placeholder="Bio" />
      <button>Add user</button>
    </div>
  );
}
```

Code Snippet 11

CREATING A SIMPLE CRUD APPLICATION [5-5]

Code Snippet 12

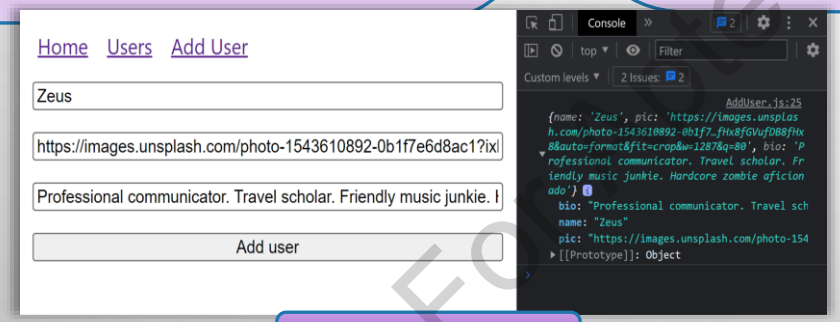
```
export function AddUser() {
  const [name, setName] =
useState("");
  const [pic, setPic] =
useState("");
  const [bio, setBio] =
useState("");
  const addUser = () => {
    const newUser = {
      name,
      pic,bio,
    };
console.log(newUser);
  }; return (
    <div className="add-user-
form">      <input
        onChange={(event) =>
setName(event.target.value)}
```

Code Snippet 13

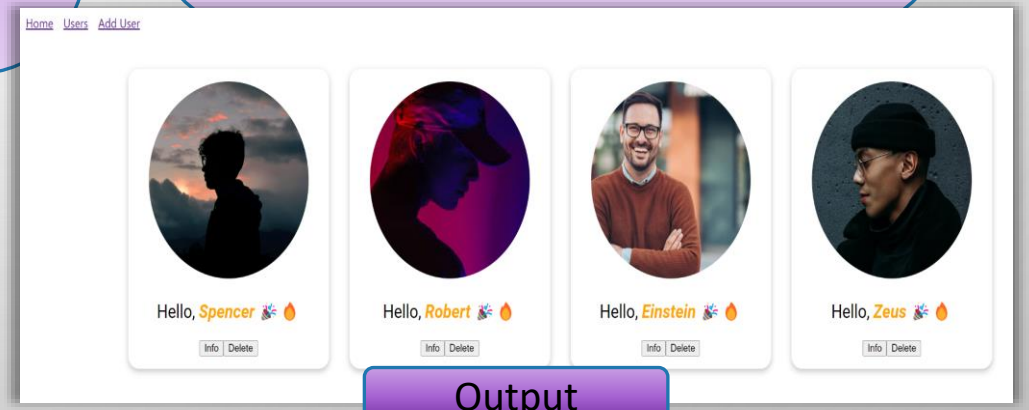
```
type="text"
placeholder="Name"      />
<input
onChange={(event) =>
setPic(event.target.value)}
        type="text"
        placeholder="Profile
Pic Url"
      />
      <input
        onChange={(event) =>
setBio(event.target.value)}
        type="text"
        placeholder="Bio"
      />
      <button
onClick={addUser}>Add
user</button>
    </div>  );}
```

Code Snippet 14

```
const addUser = () => {
    const newUser = {
      name,
      pic,
      bio,
    };    console.log(newUser);
fetch(`https://640fc234864814e5b63f0
d2f.mockapi.io/users`, {
      method: "POST",
      body: JSON.stringify(newUser),
      headers: {
        "Content-Type":
"application/json",
      },
    }).then(() =>
navigate("/users"));  };
```



Output



Output

CREATING MORE COMPLEX CRUD APPLICATION [1-9]

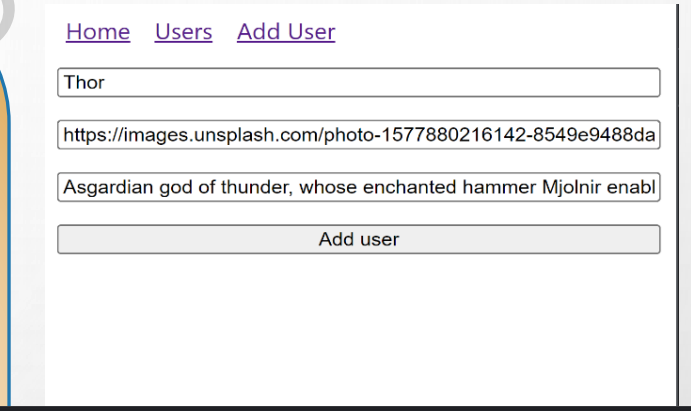
Integrating Formic to Add User Form

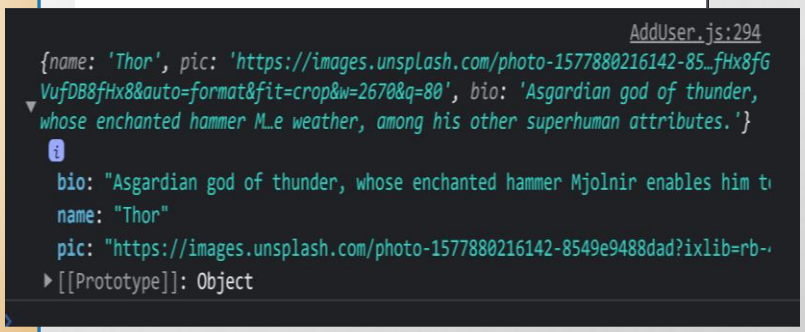
```
import { useFormik } from "formik";
import { useNavigate } from "react-
router-dom";
export function AddUser() {
  const formik = useFormik({
    initialValues: {
      name: "",
      pic: "",
      bio: "",
    },  });
  const navigate = useNavigate();
  const addUser = () => {};
  return (
    <div className="add-user-form">
      <input
onChange={formik.handleChange}
type="text" placeholder="Name" />
      <input
        onChange={formik.handleChange}
        type="text"
        placeholder="Profile Pic Url"
      />
      <input
onChange={formik.handleChange}
type="text" placeholder="Bio" />
      <button onClick={addUser}>Add
user</button>    </div>  );}
```

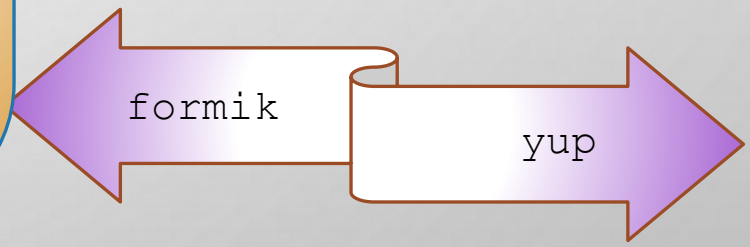
Code Snippet 15

```
export function AddUser() {
  const formik = useFormik({
    initialValues: {
      name: "",
      pic: "",
      bio: "",
    },
    onSubmit: (values) => {
      console.log(values);
    },
  });
  const navigate = useNavigate();
  const addUser = () => {};
  return (
    <form
onSubmit={formik.handleSubmit}
className="add-user-form">
<input name="name" type="text"
 placeholder="Name"  required   />
<input onChange={formik.handleChange}
 name="pic" type="text"
       placeholder="Profile Pic Url"
     />
<inputonChange={formik.handleChange}
name="bio" type="text"
       placeholder="Bio"
     />
      <button type="submit">Add
user</button>     </form>  );}
```

Code Snippet 16





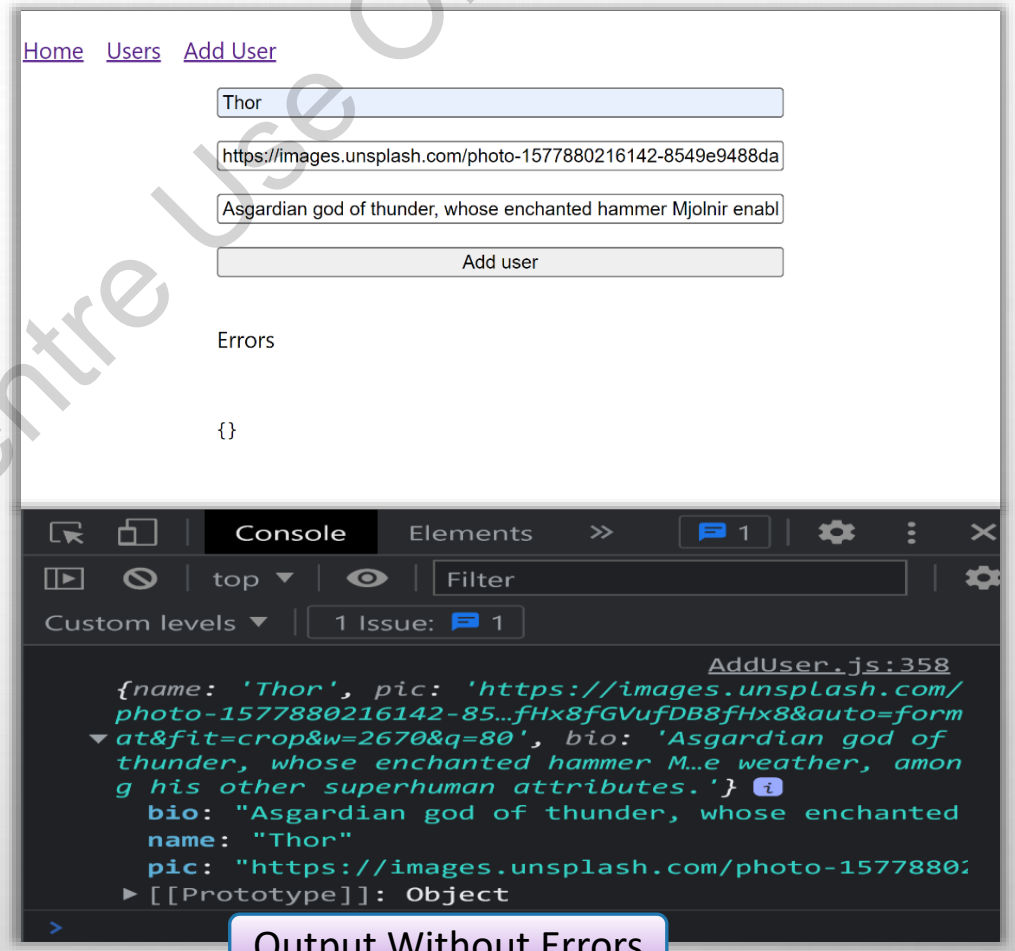


CREATING MORE COMPLEX CRUD APPLICATION [2-9]

Validating Form Data Before Submission

```
import { object, string } from "yup";
const userValidationSchema = object({
  name: string().required(),
  pic: string().url().required(),
  bio: string().min(10).required(),
});
export function AddUser() {
  const formik = useFormik({
    initialValues: {
      name: "",
      pic: "",
      bio: "",
    },
    onSubmit: (values) => {
      console.log(values);
    },
    validationSchema: userValidationSchema,
  });
  const navigate = useNavigate();
  const addUser = () => {};
  return (
    <form onSubmit={formik.handleSubmit}
className="add-user-form">
      …
      <p>Errors</p>
      <pre>{JSON.stringify(formik.errors)}</pre>
</form>  );}
```

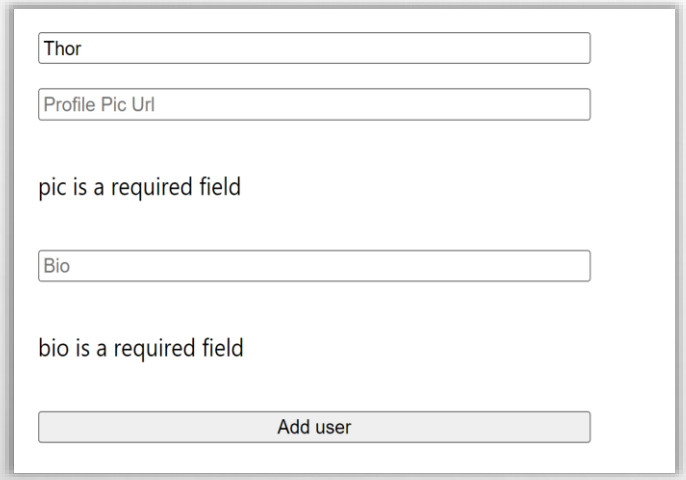
Code Snippet 17



Output Without Errors

CREATING MORE COMPLEX CRUD APPLICATION [3-9]

Handling Errors and Displaying Errors



Output preemptively shows all error messages just when the user starts typing on any of the fields

```
export function AddUser() {
  const formik = useFormik({
    initialValues: {
      name: "",
      pic: "",
      bio: "",
    },
    onSubmit: (values) => {
      console.log(values);
    },
    validationSchema:
userValidationSchema,
  });
  const navigate = useNavigate();
  const addUser = () => {};
  return (
    <form
onSubmit={formik.handleSubmit}
className="add-user-form"><input
onChange={formik.handleChange}
      onBlur={formik.handleBlur}
        name="name"
        type="text"
        placeholder="Name"
        required
      />  {formik.touched.name &&
formik.errors.name ? (
        <p>{formik.errors.name}</p>
) : null}
```

Code Snippet 18

```
<input

onChange={formik.handleChange}
    onBlur={formik.handleBlur}
        name="pic"
        type="text"
        placeholder="Profile Pic
Url"
      />
      {formik.touched.pic &&
formik.errors.pic ? (
        <p>{formik.errors.pic}</p>
      ) : null}
      <input

onChange={formik.handleChange}
    onBlur={formik.handleBlur}
        name="bio"
        type="text"
        placeholder="Bio"
      />
      {formik.touched.bio &&
formik.errors.bio ? (
        <p>{formik.errors.bio}</p>
      ) : null}
      <button type="submit">Add
user</button>
    </form>
  );
}
```

Code Snippet 19

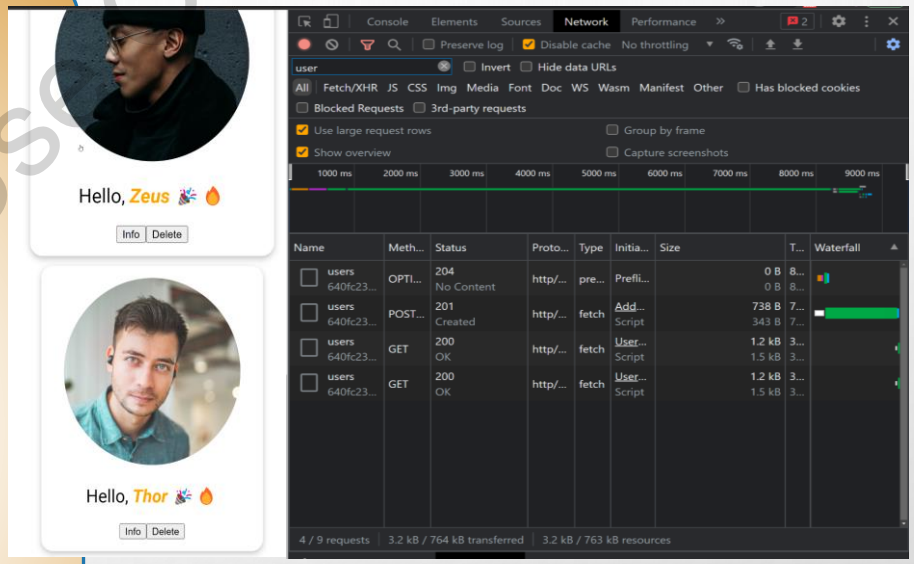
CREATING MORE COMPLEX CRUD APPLICATION [4-9]

```
  export function AddUser() {
  const formik = useFormik({
    initialValues: {   name: "",
      pic: "",
      bio: "",   },
    onSubmit: (newUser) => {
      addUser(newUser);
    }, validationSchema:
userValidationSchema,
  });
  const navigate = useNavigate();
  const addUser = (newUser) => {
    console.log(newUser);
fetch(`https://640fc234864814e5b63f0d2f.moc
kapi.io/users`, {
      method: "POST",
      body: JSON.stringify(newUser),
      headers: {
        "Content-Type": "application/json",
      },
    }).then(() => navigate("/users"));
  };  return (
    <form onSubmit={formik.handleSubmit}
className="add-user-form">
      <input
        onChange={formik.handleChange}
       onBlur={formik.handleBlur}
        name="name"
        type="text"
        placeholder="Name" required />
```

Code Snippet 20

```
{formik.touched.name &&
formik.errors.name ? (
       <p>{formik.errors.name}</p>
     ) : null}
<inputonChange={formik.handleChange}
       onBlur={formik.handleBlur}
           name="pic"
           type="text"
           placeholder="Profile Pic
   Url"
       />
       {formik.touched.pic &&
   formik.errors.pic ? (
         <p>{formik.errors.pic}</p>
       ) : null}
       <input

   onChange={formik.handleChange}
     onBlur={formik.handleBlur}
       name="bio"
       type="text"
       placeholder="Bio"
     />
       {formik.touched.bio &&
   formik.errors.bio ? (
         <p>{formik.errors.bio}</p>
       ) : null}
       <button type="submit">Add
   user</button>    </form> );}
```

Code Snippet 21



Output

CREATING MORE COMPLEX CRUD APPLICATION [5-9]

Code Snippet 22

```
<User
          name={usr.name}
          pic={usr.pic}
          id={usr.id}
          key={index}
          deleteButton={
            <button onClick={()
=>deleteUser(usr.id)}>Delete</button>          }
          editButton={
            <button onClick={() =>
navigate(`/users/edit/${usr.id}`)}>
              Edit /></button> }
```

Code Snippet 23

```
export function User({ name, pic, id, deleteButton,
editButton }) {
…
return (
    <section className="user-container">
      …
      {deleteButton}
      {editButton}
    </section>
  );
}
```

Code Snippet 24

```
import { useFormik } from "formik";
import { useNavigate, useParams } from "react-
router-dom";
import { object, string } from "yup";
import { useEffect} from "react";
export function EditUser() {
  const { id } = useParams();
  console.log(id);
  return <h1>Editing user {id}</h1>;
}
```

Code Snippet 25

```
export function EditUser() {
…
return (
    <form onSubmit={handleSubmit} className="add-
user-form">
…
      <button type="submit">Save</button>
    </form>
  );
}
```

CREATING MORE COMPLEX CRUD APPLICATION [6-9]

Code Snippet 26

```
export function EditUser() {
  const { id } = useParams();
  console.log(id);
  const { handleSubmit, handleChange, errors, touched, values,
handleBlur
 } = useFormik({
    initialValues: {
      name: "Spencer",
      pic: "https://images.unsplash.com/photo-1529665253569-
6d01c0eaf7b6?ixlib=rb-
4.0.3&ixid=MnwxMjA3fDB8MHxzZWFyY2h8NHx8cHJvZmlsZXxlbnwwfHwwfHw
%3D&w=1000&q=80",
      bio: "Award-winning web lover. Thinker. Social media
advocate. Creator. Bacon scholar. Zombie geek",
    },
    onSubmit: (newUser) => {
      addUser(newUser);
    },
    validationSchema: userValidationSchema,
  });
  const navigate = useNavigate();
  const addUser = (newUser) => {
    console.log(newUser);
    fetch(`https://640fc234864814e5b63f0d2f.mockapi.io/users`,
{
      method: "POST",
      body: JSON.stringify(newUser),
      headers: {
        "Content-Type": "application/json",
      },
    }).then(() => navigate("/users"));  };
```

Code Snippet 27

```
return (
<form onSubmit={handleSubmit} className="add-user-form">
      <input
        value={values.name}
        onChange={handleChange}
        onBlur={handleBlur}
        name="name"
        type="text"
        placeholder="Name"
        required
      />
      {touched.name && errors.name ? <p>{errors.name}</p> :
  null}
      <input
        value={values.pic}
        onChange={handleChange}
        onBlur={handleBlur}
        name="pic"
        type="text"
        placeholder="Profile Pic Url"
      />
      {touched.pic && errors.pic ? <p>{errors.pic}</p> : null}
      <input
        value={values.bio}
        onChange={handleChange}
        onBlur={handleBlur}
        name="bio"
        type="text"
        placeholder="Bio"
      />
      {touched.bio && errors.bio ? <p>{errors.bio}</p> : null}
      <button type="submit">Save</button>    </form>  );}
```

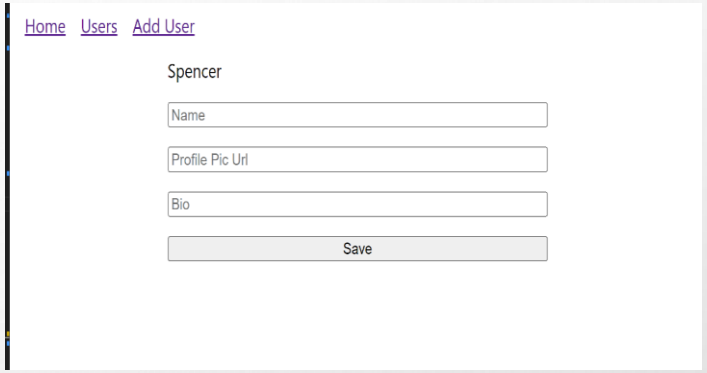
CREATING MORE COMPLEX CRUD APPLICATION [7-9]

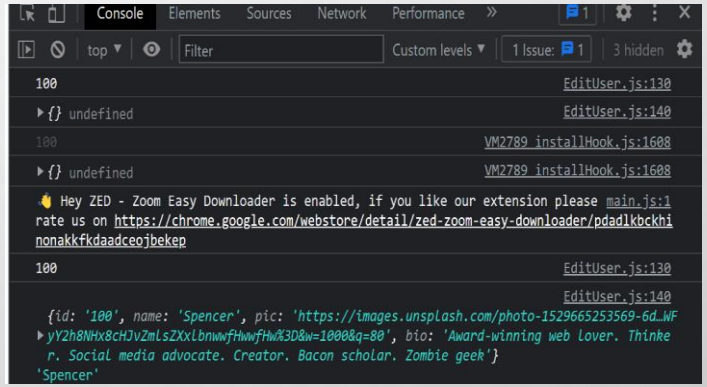
Code Snippet 28

```
export function EditUser() {
  const { id } = useParams();
  console.log(id);
  const [user, setUser] = useState({});
  useEffect(() => {
fetch(`https://640fc234864814e5b63f0d2f.mock
api.io/users/${id}`)
      .then((data) => data.json())
      .then((userInfo) =>
setUser(userInfo));
  }, []);  console.log(user, user.name);
  const { handleSubmit, handleChange,
errors, touched, values, handleBlur
    } = useFormik({
    initialValues: {
      name: user.name,
      pic: user.pic,
      bio: user.bio,
    },
    onSubmit: (newUser) => {
      addUser(newUser);
    },    validationSchema:
userValidationSchema,  });
  const navigate = useNavigate();
  const addUser = (newUser) => {
    console.log(newUser);
fetch(`https://640fc234864814e5b63f0d2f.mock
api.io/users`, {
      method: "POST",
      body: JSON.stringify(newUser),
      headers: {
        "Content-Type": "application/json",
      },    }).then(()=>navigate("/users"));
};
```

Code Snippet 29

```
return (
    <form onSubmit={handleSubmit}
className="add-user-form">
      {user.name}
      <input
        value={values.name}
        onChange={handleChange}
        onBlur={handleBlur}
        name="name"
        type="text"
        placeholder="Name"
        required
      />      {touched.name && errors.name?
<p>{errors.name}</p> : null}
      <input
        value={values.pic}
        onChange={handleChange}
        onBlur={handleBlur}
        name="pic"
        type="text"
        placeholder="Profile Pic Url"
      />      {touched.pic && errors.pic ?
<p>{errors.pic}</p> : null}
      <input
        value={values.bio}
        onChange={handleChange}
        onBlur={handleBlur}
        name="bio"
        type="text"
        placeholder="Bio"
      />
      {touched.bio && errors.bio ?
<p>{errors.bio}</p> : null}
      <button type="submit">Save</button>
</form>  );}
```





Output

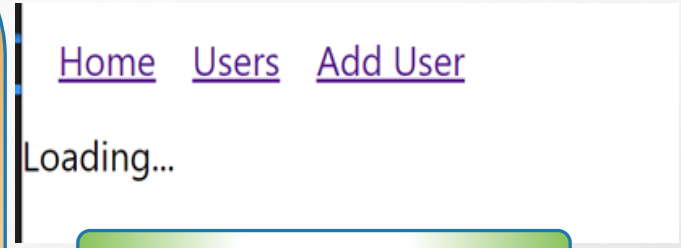
CREATING MORE COMPLEX CRUD APPLICATION [8-9]

Code Snippet 30

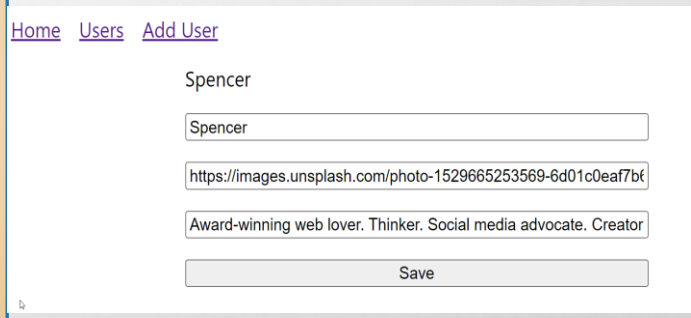
```
export function EditUser() {
  const { id } = useParams();
  console.log(id);
  const [user, setUser] = useState(null);
  useEffect(() => {
fetch(`https://640fc234864814e5b63f0d2f.mocka
pi.io/users/${id}`)
      .then((data) => data.json())
      .then((userInfo) => setUser(userInfo));
  }, []);  return user ? <EditUserForm
user={user} /> : <p>Loading...</p>;
} export function EditUserForm({ user }) {
  const { handleSubmit, handleChange, errors,
touched, values, handleBlur
 } = useFormik({    initialValues: {
      name: user.name,
      pic: user.pic,
      bio: user.bio,
    },    onSubmit: (newUser) => {
      addUser(newUser);
    },    validationSchema:
userValidationSchema,
  });  const navigate = useNavigate();
  const addUser = (newUser) => {
    console.log(newUser);
fetch(`https://640fc234864814e5b63f0d2f.mocka
pi.io/users`, {
      method: "POST",
      body: JSON.stringify(newUser),
      headers: {
    "Content-Type": "application/json",  },
```

Code Snippet 31

```
}).then(() => navigate("/users"));
  }; return (
    <form onSubmit={handleSubmit}
className="add-user-form">
      {user.name}      <input
        value={values.name}
        onChange={handleChange}
        onBlur={handleBlur}
        name="name"
        type="text"
        placeholder="Name"
        required
      />      {touched.name && errors.name ?
<p>{errors.name}</p> : null}
      <input      value={values.pic}
        onChange={handleChange}
        onBlur={handleBlur}
        name="pic"
        type="text"
        placeholder="Profile Pic Url"
      />      {touched.pic && errors.pic ?
<p>{errors.pic}</p> : null}
      <input        value={values.bio}
onChange={handleChange}
onBlur={handleBlur}
        name="bio"        type="text"
        placeholder="Bio"
      />      {touched.bio && errors.bio ?
<p>{errors.bio}</p> : null}
      <button type="submit">Save</button>
    </form>  );}
```



Loading the User Data

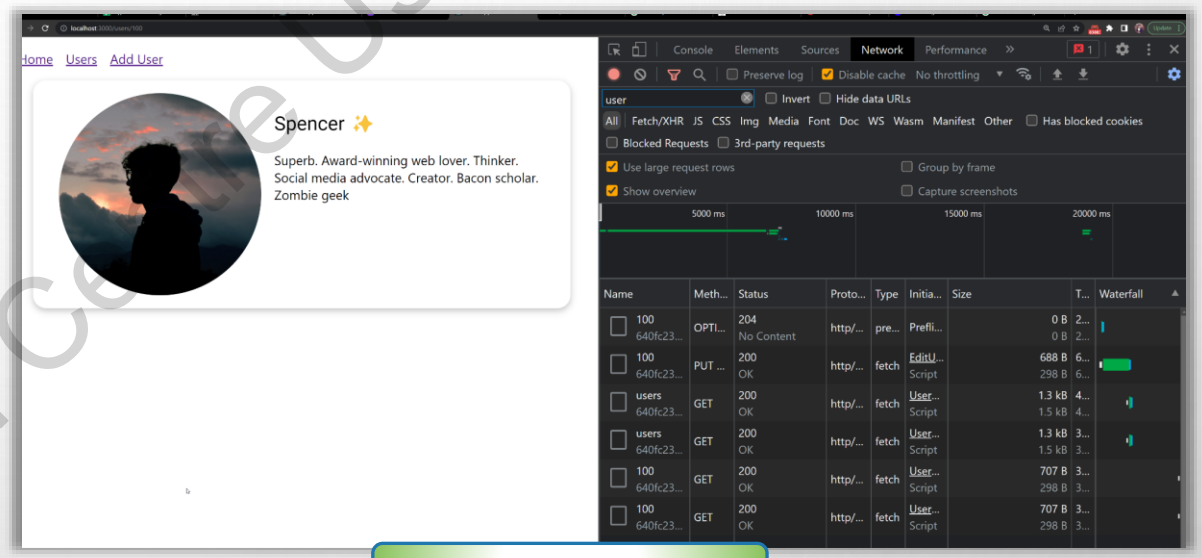


Output of with Fetched User Details

CREATING MORE COMPLEX CRUD APPLICATION [9-9]

Code Snippet 31

```
const updateUser = (updatedUser) => {
    console.log(updatedUser);

fetch(`https://640fc234864814e5b63f0d2f.mockapi.io/users/${user.id}`, {
      method: "PUT",
      body: JSON.stringify(updatedUser),
      headers: {
        "Content-Type": "application/json",
      },
    }).then(() => navigate("/users"));
  };
```



Output

SUMMARY

- To organize the app, the code is split into different files using the move to file feature in Visual Studio Code Editor.
- The `mockapi.io` is used to mock the data and easily test CRUD operation in the app.
- Four major CRUD methods are: `GET`, `POST`, `PUT`, and `DELETE`.
- Using `formik` and `yup` makes building forms easier.
- Validations are added to:
  - Avoid junk data.
  - Improve user experience.