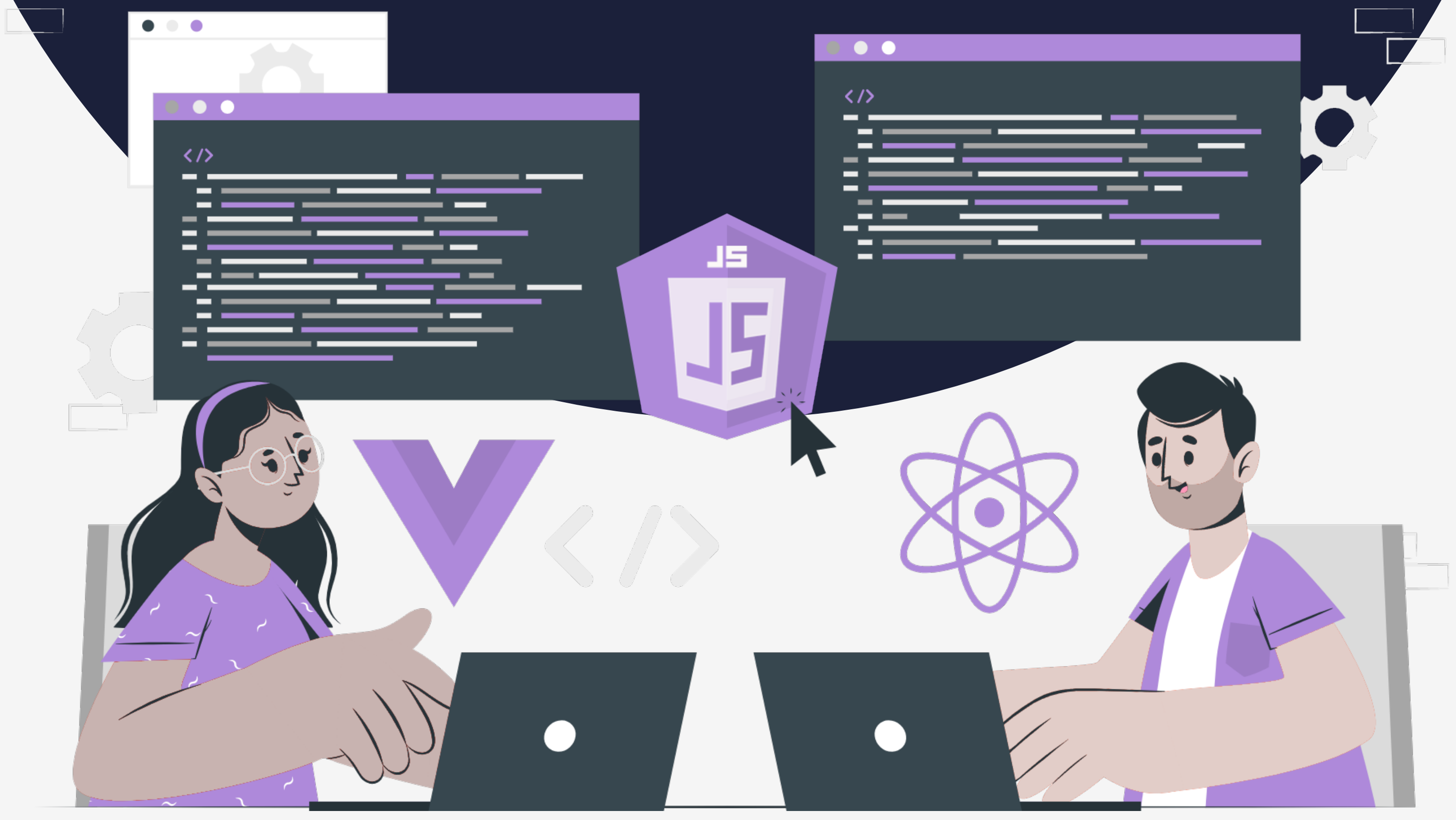
Lesson:

createRoot() & render() methods of react18



PW SKILLS

Topics Covered:

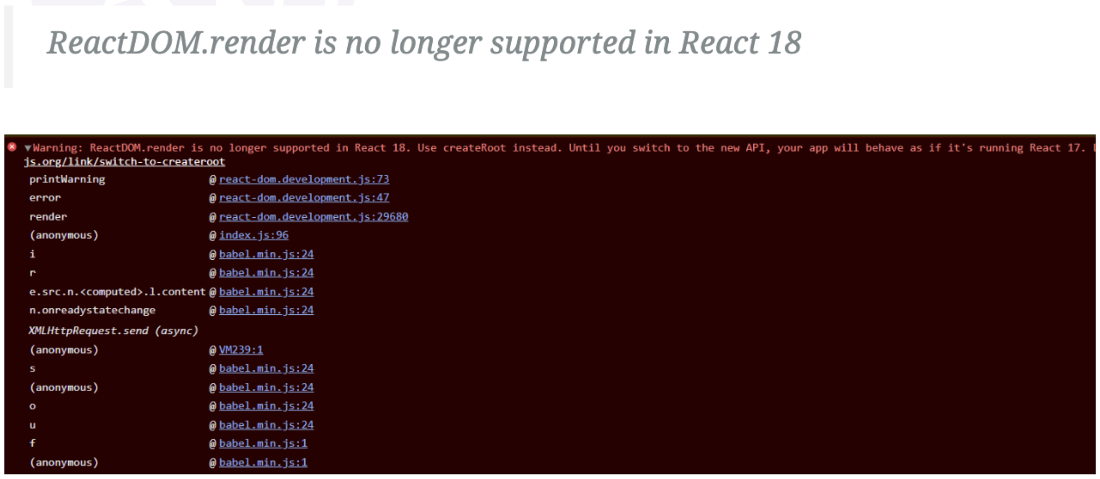
1. Introducing the new root API
    a. Legacy Root API
    b. New Root API
2. Example

Introducing the new root API
A root in React refers to the top-level data structure that renders a tree. In React 18, we will have two root APIs: **the legacy root API and new root API.**

Legacy Root API

The legacy root API is the existing API called with the ReactDOM.render method. It will create a root running in legacy mode, which is similar to usage in React version 17. If We are using ReactDOM.render() in your React 18 app, we will definitely see the below issue.

A JSX element is converted into a regular JavaScript object during compilation so that it can be used to construct a genuine DOM element. For instance, the JavaScript generated from the JSX code above might be as follows:



> *ReactDOM.render is no longer supported in React 18*

```
▾Warning: ReactDOM.render is no longer supported in React 18. Use createRoot instead. Until you switch to the new API, your app will behave as if it's running React 17.
js.org/link/switch-to-createroot
 printWarning                  @ react-dom.development.js:73
 error                         @ react-dom.development.js:47
 render                        @ react-dom.development.js:29680
 (anonymous)                   @ index.js:96
 i                             @ babel.min.js:24
 r                             @ babel.min.js:24
 e.src.n.<computed>.l.content  @ babel.min.js:24
 n.onreadystatechange          @ babel.min.js:24
 XMLHttpRequest.send (async)
 (anonymous)                   @ VM239:1
 s                             @ babel.min.js:24
 (anonymous)                   @ babel.min.js:24
 o                             @ babel.min.js:24
 u                             @ babel.min.js:24
 f                             @ babel.min.js:1
 (anonymous)                   @ babel.min.js:1
```

The reason is **ReactDOM.render is no longer supported in React 18**.

Earlier, we used to render the component in below way:

```javascript
ReactDOM.render(<NavBar />, document.getElementById('root'))
```

The render() method of the react-dom package is considered legacy starting react-dom version 18.

The method is replaced with the createRoot() method that is exported from react-dom/client.

The createRoot() method takes the root element as a parameter and creates a React root.

New Root API
createRoot() is a new method introduced in React 18 that allows you to create a separate root for a React tree outside of the main React DOM tree. It takes a single argument, which is a reference to an existing DOM element. It returns a new root object that has a render() method for rendering React components into the specified DOM element.

Call createRoot to create a React root for displaying content inside a browser DOM element.

```javascript
/**
 * 🚀 Old way to do
 */
import ReactDOM from 'react-dom';

const rootNode = document.getElementById('root');
ReactDOM.render(<App />, rootNode);
```

```
/**
 * 🚀 New way with the react-dom/client API
 */
import ReactDOM from 'react-dom/client';

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

React will create a root for the domNode, and take over managing the DOM inside it. After we've created a root, we need to call root.render to display a React component inside of it:

JavaScript
```
root.render(<App />);
```

An app fully built with React will usually only have one createRoot call for its root component. A page that uses "sprinkles" of React for parts of the page may have as many separate roots as needed.

Let's take an example to understand better how to use createRoot()

index.html

JavaScript
```
<!DOCTYPE html>
<html>
  <head><title>My app</title></head>
  <body>
    <!-- This is the DOM node -->
    <div id="root"></div>


  </body>
</html>
```

index.js

JavaScript
```
import { createRoot } from 'react-dom/client';
import App from './App.js';
import './styles.css';

const root = createRoot(document.getElementById('root'));
root.render(<App />);
```

App.js

```javascript
import { useState } from 'react';

export default function App() {
  return (
    <>
      <h1>Hello, world!</h1>
      <Counter />
    </>
  );
}

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      You clicked me {count} times
    </button>
  );
}
```

Let’s take another example where we can call createRoot methods multiple times to create a root for each top-level piece of UI managed by React.We can display different content in each root by calling root.render.

index.html

```javascript
<nav id="navigation"></nav>
<main>
  <p>This paragraph is not rendered by React (open index.html to verify).</p>
  <section id="comments"></section>
</main>
```

index.js

```javascript
import { createRoot } from 'react-dom/client';
import { Comments, Navigation } from './Components.js';

const navDomNode = document.getElementById('navigation');
const navRoot = createRoot(navDomNode);
navRoot.render(<Navigation />);

const commentDomNode = document.getElementById('comments');
const commentRoot = createRoot(commentDomNode);
commentRoot.render(<Comments />);
```

Component.js

```javascript
export function Navigation() {
  return (
```

```javascript
    <ul>
      <NavLink href="/">Home</NavLink>
      <NavLink href="/about">About</NavLink>
    </ul>
  );
}

function NavLink({ href, children }) {
  return (
    <li>
      <a href={href}>{children}</a>
    </li>
  );
}

export function Comments() {
  return (
    <>
      <h2>Comments</h2>
      <Comment text="Hello!" author="Sophie" />
      <Comment text="How are you?" author="Sunil" />
    </>
  );
}

function Comment({ text, author }) {
  return (
    <p>{text} — <i>{author}</i></p>
  );
}
```