Name: Deepanshu Rathore

Superset ID: 6358199

Week 7

Features of ES6

ECMAScript 6 (ES6) introduced several modern features to JavaScript that improved code readability, maintainability, and performance. Some of the key features include the let and const keywords for better variable declaration, arrow functions for shorter function syntax, template literals for easier string formatting, and default parameters in functions. ES6 also introduced classes for object-oriented programming, modules for code reusability, and new data structures such as Map and Set. Additionally, it brought promises for handling asynchronous operations, destructuring assignments for simpler extraction of values, and the spread and rest operators for working with arrays and objects more efficiently.

JavaScript let

The let keyword in JavaScript is used to declare block-scoped variables. Unlike the traditional var keyword, variables declared with let are limited to the scope of the block, statement, or expression where they are defined. This reduces the chances of accidental variable overwriting and helps in writing cleaner, more predictable code. Variables declared with let can be reassigned, but they cannot be redeclared within the same scope.

Differences between var and let

The main difference between var and let lies in their scope and redeclaration behavior. Variables declared with var have function scope or global scope, which means they are accessible outside the block in which they are declared. In contrast, let is block-scoped, meaning it is only accessible within the block where it is defined. Another difference is that var allows redeclaration of variables in the same scope, while let does not permit this. Moreover, var variables are hoisted to the top of their scope and initialized with undefined, whereas let variables are hoisted but not initialized, leading to a temporal dead zone until the declaration is encountered.

JavaScript const

The const keyword in JavaScript is used to declare variables whose values cannot be reassigned after they are initialized. Similar to let, const is block-scoped and cannot be redeclared within the same scope. While the value of a const primitive cannot be changed, objects and arrays declared with const can still have their contents modified

because the binding is constant, not the value itself. This makes const ideal for values that should remain fixed throughout the execution of a program.

ES6 Class Fundamentals

ES6 introduced a new syntax for creating classes, making object-oriented programming in JavaScript more structured and easier to understand. A class in ES6 is defined using the class keyword, followed by a constructor method that initializes object properties. Other methods can be defined directly inside the class without using the function keyword. ES6 classes are essentially syntactic sugar over JavaScript's existing prototype-based inheritance model but provide a cleaner and more familiar syntax for developers coming from other programming languages.

ES6 Class Inheritance

Inheritance in ES6 allows one class to extend another using the extends keyword. This enables a child class to inherit properties and methods from a parent class, promoting code reusability and easier maintenance. The super keyword is used inside the child class constructor to call the parent class constructor, ensuring proper initialization of inherited properties. This mechanism also allows method overriding, where a child class can provide its own implementation of a method that exists in the parent class.

ES6 Arrow Functions

Arrow functions in ES6 provide a shorter syntax for writing functions. They are defined using the => syntax and do not have their own this context. Instead, they inherit this from the surrounding lexical scope, which makes them especially useful in callbacks and event handling. Arrow functions cannot be used as constructors and do not have the arguments object. They are best suited for short, concise function expressions.

Set and Map in ES6

The Set object in ES6 is a collection of unique values, meaning it cannot store duplicate elements. It allows efficient checks for value existence and provides methods such as add(), delete(), and has(). On the other hand, the Map object is a collection of key-value pairs where keys can be of any data type. Map maintains the order of insertion and provides methods such as set(), get(), and delete(). Both Set and Map are useful for storing and managing collections of data more efficiently than plain objects or arrays in certain scenarios.

JSX

JSX stands for JavaScript XML and is a syntax extension for JavaScript used in React to describe what the user interface should look like. It allows developers to write HTML-like code inside JavaScript, making it easier to visualize the structure of UI components. JSX is not understood by browsers directly, so it must be transpiled into regular JavaScript using tools like Babel. This syntax improves code readability and allows seamless integration of HTML structures with JavaScript logic in a single file.

ECMAScript

ECMAScript is the standardized specification for JavaScript, maintained by ECMA International. It defines the core features and functionalities of the JavaScript language, ensuring consistency across different browsers and environments. New versions of ECMAScript are released periodically, each introducing new syntax, features, and improvements to make the language more powerful and developer-friendly. For example, ES6 introduced important features like let, const, arrow functions, classes, template literals, and new data structures such as Map and Set.

React.createElement()

The React.createElement() function is used to create React elements without using JSX. It takes three main arguments: the type of element (such as a tag name or a React component), an object containing the element's attributes or properties, and the element's children. React internally uses React.createElement() even when JSX is written. JSX is essentially syntactic sugar for calling React.createElement() in a more readable format. For example, <h1>Hello</h1> in JSX is equivalent to React.createElement('h1', null, 'Hello').

Creating React Nodes with JSX

React nodes represent the building blocks of the React virtual DOM. With JSX, creating nodes is simple and intuitive because you can write them as HTML-like elements. For example, a heading can be created using <h1>Hello World</h1>. Behind the scenes, JSX converts these elements into JavaScript objects that represent React nodes, which React then uses to update and manage the virtual DOM efficiently. JSX allows nesting of elements, passing props, and integrating logic directly inside the markup.

Rendering JSX to the DOM

To render JSX to the DOM, React uses the ReactDOM.render() method in earlier versions or the createRoot API in modern versions. The JSX element is passed as the first argument, and the target DOM node is passed as the second argument. For example, ReactDOM.render(<App />, document.getElementById('root')) mounts the App component into the HTML element with the id root. React efficiently updates the DOM by comparing the virtual DOM with the real DOM and making only the necessary changes.

Using JavaScript Expressions in JSX

JSX allows embedding JavaScript expressions inside curly braces {}. This feature enables dynamic rendering of values, conditional logic, and integration of variables directly within the UI markup. For example, {name} can be used inside JSX to display a variable's value, and {items.length > 0 ? 'Items available' : 'No items'} can be used for conditional rendering. This integration of JavaScript logic within JSX provides flexibility and enhances interactivity in React applications.

Using Inline CSS in JSX

Inline CSS in JSX is applied using the style attribute, which takes a JavaScript object instead of a string. The property names in this object use camelCase instead of kebab-case as in regular CSS. For example, <div style={{ backgroundColor: 'blue', fontSize: '16px' }}>Hello applies a blue background and font size of 16 pixels. Inline styles in JSX are useful for applying dynamic styles based on component state or props, as they can be controlled using JavaScript variables.

React Events

React events are actions or occurrences that happen in the user interface, such as clicking a button, typing in a text field, or submitting a form. React handles events in a way similar to handling events in regular HTML but with some differences. Events in React are named using camelCase and passed as functions instead of strings. React also uses its own event system that works across all browsers to ensure consistent behavior, making it easier for developers to manage user interactions.

Event Handlers

Event handlers in React are functions that are triggered in response to specific events. For example, a function can be called when a button is clicked to perform an action such as updating state or displaying a message. In React, event handlers are passed as props to elements using curly braces. For example, <button onClick={handleClick}>Click Me</button> attaches the handleClick function to the button's click event. Event handlers help manage user interactions and make applications dynamic and interactive.

Synthetic Event

A synthetic event in React is a wrapper around the browser's native event system. It provides a consistent interface across different browsers, ensuring that event handling works the same way everywhere. Synthetic events have the same properties as native events, such as target, type, and currentTarget, but they are managed by React's internal event system for performance optimization. React uses event delegation to handle events efficiently, attaching listeners to the root of the DOM rather than to individual elements.

React Event Naming Convention

React follows a specific naming convention for events to maintain consistency and avoid conflicts with standard HTML attributes. Event names in React use camelCase instead of lowercase. For example, the HTML onclick attribute becomes onClick in React, and onchange becomes onChange. This convention applies to all event types, such as onSubmit, onMouseOver, and onKeyDown. Following this naming style ensures that event handlers are recognized and executed correctly by React's event system.

Conditional Rendering in React

Conditional rendering in React refers to displaying different UI elements or components based on certain conditions. It works the same way as conditional statements in JavaScript, allowing developers to control what gets displayed to the user. Common techniques for conditional rendering include using the if statement, the ternary operator, and logical && operators directly inside JSX. For example, {isLoggedIn ? <Dashboard /> : <Login />} renders the Dashboard component if the user is logged in or the Login component otherwise. This approach helps create dynamic and interactive user interfaces that respond to application state and user actions.

Element Variables

Element variables in React are variables that store React elements. They are useful for deciding what to render before returning the final JSX. For example, a variable can be assigned one element under certain conditions and a different element under other conditions, and then be included in the JSX output. This method makes conditional rendering cleaner and easier to manage, especially when multiple conditions need to be checked before deciding what content should be displayed.

Preventing Components from Rendering

In some cases, a component may need to avoid rendering entirely. This can be achieved by returning null from the component's render method or functional component body. Returning null tells React not to render anything for that component while still keeping it in the component hierarchy. This technique is useful when a component's output is not needed until a specific condition is met, such as hiding a UI section when a user is not logged in. Preventing rendering in this way helps optimize performance and keeps the user interface clean.

Various Ways of Conditional Rendering

There are multiple ways to implement conditional rendering in React. The most common method is using the if statement, which allows rendering different components based on certain conditions. Another approach is using the ternary operator condition ? truePart : falsePart for inline conditional rendering. Logical && is also widely used, especially when rendering an element only if a condition is true. Developers can also use element variables to store and return the desired JSX based on conditions. These techniques make React components dynamic and adaptable to changes in application state or user input.

Rendering Multiple Components

In React, multiple components can be rendered together by grouping them inside a parent element. This can be done using a or the React fragment syntax <>...</> to avoid adding unnecessary nodes to the DOM. Rendering multiple components is common when building layouts that display several independent UI sections at once.

For example, a dashboard may render a navigation bar, a sidebar, and a content section as separate components displayed together within one parent container.

List Component

A list component in React is a component that renders a list of data items dynamically. It typically takes an array of data as input and uses the JavaScript map() function to generate a series of child elements. For example, a list component can display a list of tasks, products, or messages by iterating over the data array and rendering each item within its own JSX structure. This approach makes lists reusable and easy to maintain, as they adapt automatically when the underlying data changes.

Keys in React Applications

Keys in React are special string attributes used to uniquely identify elements in a list. They help React determine which items have changed, been added, or removed, enabling efficient re-rendering of only the affected elements. Without keys, React might re-render all elements in a list, reducing performance. Keys should be unique among sibling elements and are usually assigned using a stable property like an ID from the data source.

Extracting Components with Keys

When a list is broken down into smaller reusable components, keys must be passed to the elements generated inside the list. If a component is extracted from a list rendering function, the key should be assigned to the component instance inside the map() function, not within the component itself. This ensures that React can still track each element's identity correctly and perform optimized rendering.

React Map and map() Function

In React, the JavaScript map() function is commonly used to render lists of elements dynamically. It iterates over an array and returns a new array containing React elements based on each item's data. For example, items.map(item => <li key={item.id}>{item.name}) creates a list of <li> elements from an array of objects. This method is essential for displaying data-driven components such as lists, tables, and grids in a React application.