React JS

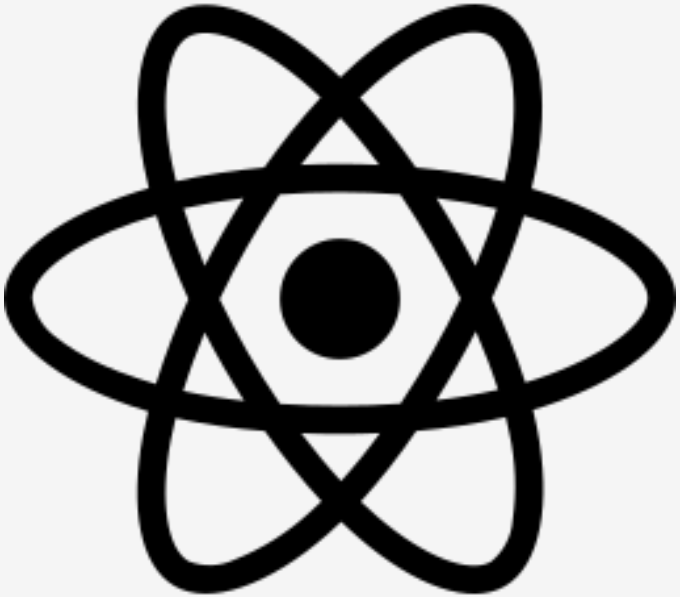
A Comprehensive Guide to Modern Web Development

What is React?

JavaScript Library
Developed by Facebook (Meta) in 2013

Purpose
Building user interfaces for web applications

Key Characteristic
Component-based architecture for reusable UI elements



Single-Page Applications (SPA)

Traditional Websites

Each page has its own template (HTML/CSS) that is returned to the user on every navigation

Single-Page Applications

One page with dynamic content modification. A single template that updates its content without full page reloads

Benefits of SPAs

- Faster user experience
- No page reloads
- Smooth transitions
- Reduced server load
- Better mobile experience

React Components

Components are reusable, independent pieces of the UI. Each component is a JavaScript function or class that returns HTML content (JSX).

Function-Based Component

(Modern Approach)

```
function Profile() {
  return (
    <div>
      <p>Welcome!</p>
    </div>
  );
}
```

Class-Based Component

(Legacy Approach)

```
class Profile extends Component {
  render() {
    return (
      <div>
        <p>Welcome!</p>
      </div>
    );
  }
}
```

JSX (JavaScript XML)

JSX is a syntax extension that allows you to write HTML-like code in JavaScript. It gets transpiled to JavaScript function calls at build time.

JSX Example with Props

```
function Profile({ name }) {
  return (
    <div>
      <h1>Hello, {name}!</h1>
      <p>Welcome to React</p>
    </div>
  );
}
```

Key Points

- HTML-like syntax
- Transpiled to JS
- Use {} for expressions
- Not interpreted by browser

React Router

React Router enables navigation in single-page applications by syncing the UI with the URL without actual page changes.

Import and setup

```
import { BrowserRouter, Routes, Route, Navigate }
  from 'react-router-dom';


function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='/about' element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Props (Properties)

Props allow data to flow from parent to child components, similar to function parameters.

Parent Component (Passing Props)

```
function Tweet() {
  let item = {
    author: 'Dennis',
    tweet: 'My First Tweet!'
  };
  return (
    <div>
      <Author name={item.author} />
      <Content body={item.tweet} />
    </div>
  );
}
```

Child Component (Receiving)

```
function Content(props) {
  return (
    <p>{props.body}</p>
  );
}
```

Prop Drilling

Props can pass through multiple component layers, but excessive nesting becomes messy and hard to maintain.

React State

State is a JavaScript object that represents information about a component's current situation. When state changes, React re-renders the component.

Basic State Object

```
state = {
  user: 'Dennis',
  isAdmin: true,
  notes: [],
}
```

Class-Based State

```
class Notes extends Component {
  constructor(props) {
    super(props);
    this.state = {
      user: 'Dennis',
      isAdmin: true,
      notes: [],
    };
  }
  render() { /* ... */ }
}
```

Modern State with Hooks

useState Hook - Displaying Data

```
function Notes() {
  // Declare state variable and setter function
  let [notes, setNotes] = useState([]);
  // notes: state variable
  // setNotes: function to update notes

  return (
    <ul>
      {notes.map(note => (
        <li key={note.id}>{note.title}</li>
      ))}
    </ul>
  );
}
```

Updating State

Fetching and Updating State

```
let [notes, setNotes] = useState([]);

let getNotes = async () => {
  let res = await fetch('/notes');
  let data = await res.json();

  setNotes(data);
  // ↑ State update triggers re-render
};
```

Important

State updates trigger the component lifecycle:

1. State changes
2. Component re-renders
3. Virtual DOM updates
4. Real DOM syncs

Component Lifecycle

Understanding the lifecycle is critical for every React developer and is a common interview topic.

Mounting
Component added to DOM

→

Updating
Component being modified

→

Unmounting
Component removed from DOM

Each phase has specific methods and behaviors that control how components behave.

Lifecycle Methods - Class Components

Class Component Lifecycle Methods

```
class MyComponent extends Component {

  componentDidMount() {
    // Called after component is added to DOM
    // Perfect for: API calls, subscriptions, timers
  }

  componentDidUpdate(prevProps, prevState) {
    // Called after component updates (props or state change)
    // Perfect for: Responding to prop/state changes
  }

  componentWillUnmount() {
    // Called before component is removed from DOM
    // Perfect for: Cleanup, cancel subscriptions, clear timers
  }
}
```

useEffect Hook - Function Components

useEffect - Handles All 3 Lifecycle Phases

```
useEffect(() => {
  // Mounting + Updating logic
  console.log('Component rendered');

  return () => {
    // Unmounting/cleanup logic
    console.log('Cleanup');
  };
}, [dependencies]);
// ↑ Controls when effect runs
```

Dependency Array

[] - Empty
Runs once on mount

[var] - With deps
Runs when deps change

No array
Runs on every render

React Hooks (React 16.8+)

Hooks are functions that let you 'hook into' React features from function components. They enable state and lifecycle in functional components.

`useState()`	Add and update state in functional components
`useEffect()`	Perform side effects in lifecycle phases
`useContext()`	Access context values without prop drilling
`useReducer()`	Manage complex state logic
`useCallback()`	Memoize callback functions
`useMemo()`	Memoize expensive computations
`useRef()`	Access DOM elements or persist values

Global State Management

For data that needs to be accessible across multiple components (authentication, theme, user settings), global state solutions prevent prop drilling.

React Context API

(Built-in Solution)

✓ No additional library
✓ Perfect for simple global state
✓ Built into React

Use for:
- Theme toggling
- User authentication
- Language preferences

Third-Party Solutions

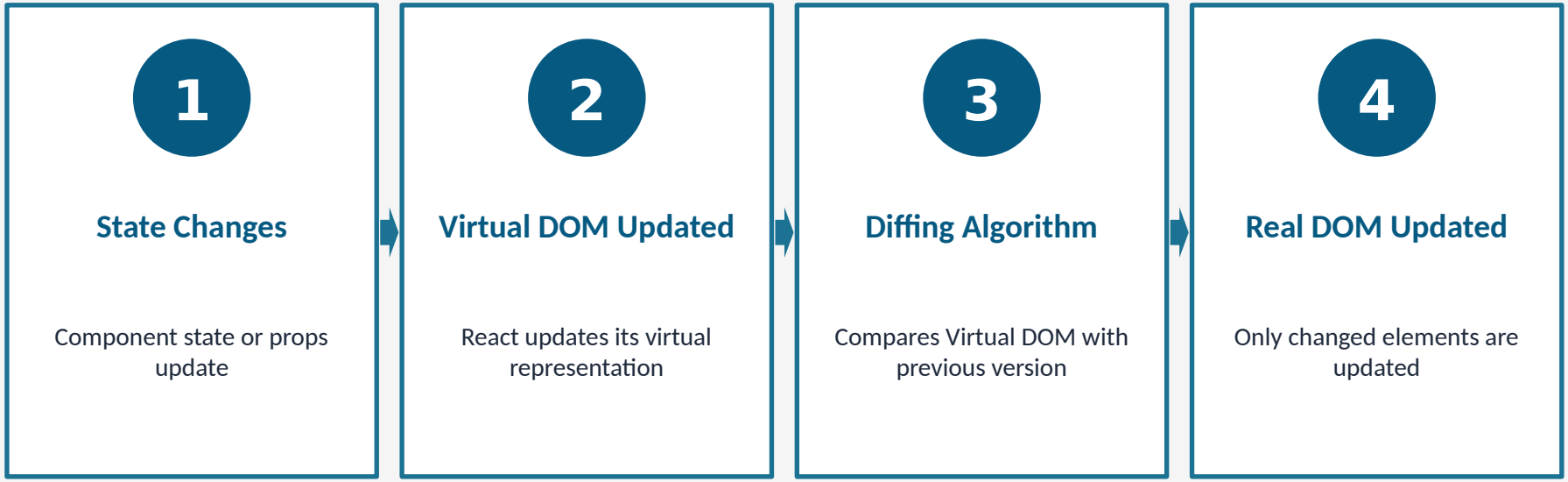
(Redux, MobX, Zustand, etc.)

✓ Advanced features
✓ Better performance at scale
✓ DevTools integration

Use for:
- Large applications
- Complex state logic
- Time-travel debugging

Virtual DOM

React creates a virtual representation of the DOM. When components update, React compares the Virtual DOM with the Real DOM and only updates what changed.



Result: Optimal performance by minimizing expensive DOM operations

The Key Prop

Every item in a dynamically rendered list must have a unique 'key' prop to help React identify which items have changed.

❌ **WITHOUT Key (Console Error)**

```
{notes.map(note => (
  <li>{note.title}</li>
))}
```

✓ **WITH Key (Correct)**

```
{notes.map(note => (
  <li key={note.id}>{note.title}</li>
```

Why Keys Matter

Keys help React:

- Identify which items changed
- Optimize re-renders
- Maintain component state

Event Handling in React

Event handling in React is similar to traditional JavaScript but with key differences: camelCase naming and no need for addEventListener.

React Event Handling

```
function App() {
  const handleClick = () => {
    console.log('Button clicked!');
  };

  return (
    <button onClick={handleClick}>
      Click Me
    </button>
  );
}
```

Common Events

- onClick
- onChange
- onSubmit
- onMouseEnter
- onMouseLeave
- onFocus
- onBlur
- onKeyDown

Form Handling

In React, form elements maintain their state in component state, giving you full control over form behavior (controlled components).

Controlled Component Example

```
function LoginForm() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();  // Prevent page reload
    console.log({ email, password });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input value={email} onChange={e => setEmail(e.target.value)} />
      <input value={password} onChange={e => setPassword(e.target.value)} />
      <button type='submit'>Login</button>
    </form>
  );
```

Conditional Rendering

Render different content based on conditions using JavaScript logical operators and ternary expressions.

Logical AND (&&) Operator

```
return (
  <div>
    {isAuthenticated &&
      <span>
        Hello {user.name}
      </span>
    }
  </div>
);
```

Ternary Operator

```
return (
  <div>
    {isAuthenticated ? (
      <span>Hello {user.name}</span>
    ) : (
      <span>Please Login</span>
    )}
  </div>
);
```

Essential React Commands

Key commands for creating, running, and building React applications.

`npx create-react-app <appname>`

Create a new React application with all dependencies

`npm start`

Start the development server (typically on localhost:3000)

`npm run build`

Build optimized production bundle for deployment

`npm test`

Run test suite in interactive watch mode

`npm install <package>`

Install additional packages or dependencies

React Key Takeaways

- Component-based architecture for reusable UI
- JSX combines JavaScript and HTML-like syntax
- State and props manage data flow
- Virtual DOM optimizes rendering performance
- Hooks enable modern functional components
- Understanding lifecycle is crucial for interviews