1. Getting Started with React 19

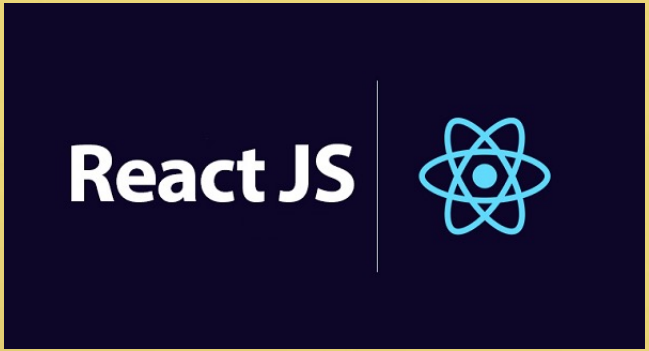
DEVFORGE
CODE CRAFTERS

Contents

DEVFORGE
CODE CRAFTERS

Lesson 1:

Introduction to React 19

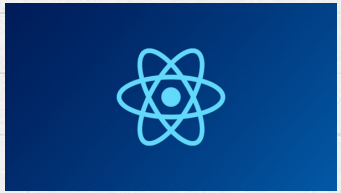


DEVFORGE
CODE CRAFTERS

Learning Objectives

By the end of this section, you will:

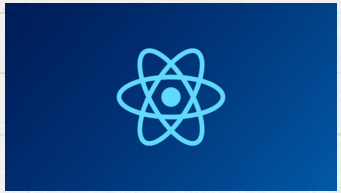
- Understand key new features in React 19.
- Learn how React Server Components (RSC) work.
- Explore Actions and Form Optimizations.
- Identify how these features improve performance and developer experience.



DEVFORGE
CODE CRAFTERS

What's New in React 19

- React 19 introduces several improvements that enhance performance, developer experience, and application maintainability. The major updates include:
- Let's have a closer look at some of them



1. React Server Components (RSC)

React Server Components allow developers to execute components on the server, reducing client-side JavaScript bundle size.

Key Benefits:

✅ Faster initial page loads

✅ Reduced client-side JavaScript execution

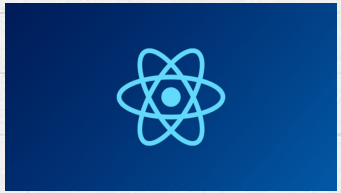
✅ Seamless integration with Next.js

How It works:

✅ Server Components run only on the server and never reach the client.

✅ They can fetch data directly from a database or API.

✅ Server-side rendering (SSR) and static site generation (SSG) become more efficient.



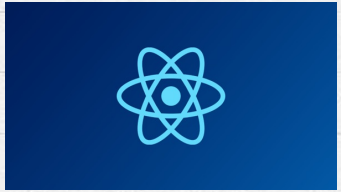
DEVFORGE
- CODE CRAFTERS -

Example:

Basic Server Components

```jsx
// app/components/UserList.server.jsx
import db from "@/lib/db"; // Simulated database import

export default async function UserList() {
const users = await db.getUsers(); // Fetch users from the database

return (
<ul>
{users.map((user) => (
<li key={user.id}>{user.name}</li>
))}
</ul>
);
}
```

1.
React Server Components (RSC)



Actions: A New Way to Handle Data Mutations

Actions simplify form handling, API calls, and state updates by eliminating the need for `useState` and `useEffect` in many cases.

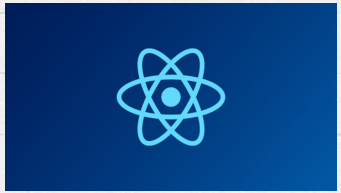
Key Benefits:

✅ Reduce boilerplate code

✅ Improve performance with built-in handling of async state updates

✅ Integrated with React Server Components

2.
Actions



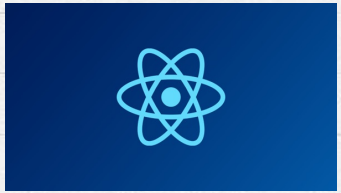
DEVFORGE
- CODE CRAFTERS -

Example:

Form Submission with Actions

```
"use server";

export async function createTask(formData) {
const task = { title: formData.get("title") };
await db.addTask(task);
}
```

2.
Actions



3. Form Optimizations

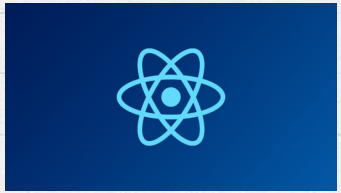
React 19 improves how forms are handled, making submissions more efficient.

Key Benefits:

✅ Built-in `pending` state to show loading indicators

✅ Automatic optimistic updates

✅ Seamless integration with Actions

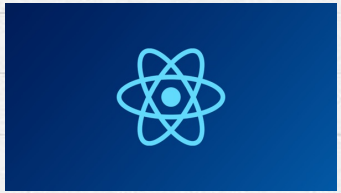


Example:

Optimized Form Submission

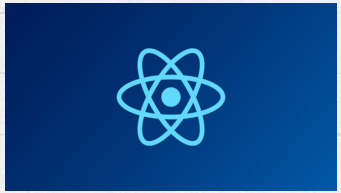
```
import { useFormState } from "react";

export default function TaskForm() {
const [state, formAction] = useFormState(createTask, null);

return (
<form action={formAction}>
<input name="title" placeholder="New Task" />
<button type="submit" disabled={state?.pending}>Add Task</button>
</form>
);
}
```

3. Form Optimizations



Lesson Summary

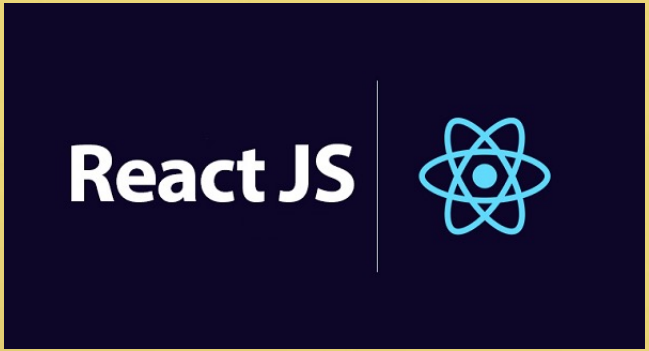
- React Server Component move execution to the server, reducing client-side overhead.
- Actions simplify data mutations, eliminating unnecessary API calls.
- Form Optimizations improve user experience with built-in state management.



DEVFORGE
CODE CRAFTERS

Lesson 2:

Setting Up a Project with Vite

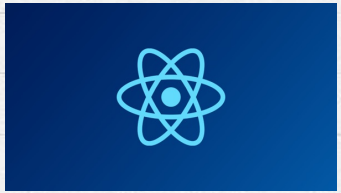


DEVFORGE
- CODE CRAFTERS -

Learning Objectives

By the end of this lesson, you will:

- ✅ Set up a React 19 project using Vite.
- ✅ Understand the project structure and key files.
- ✅ Configure ESLint and Prettier for code consistency.
- ✅ Run the development server and verify the setup.



DEVFORGE
- CODE CRAFTERS -

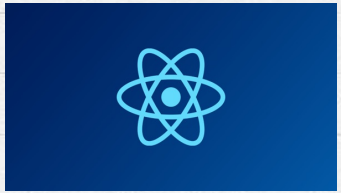
Why Use Vite for React 19

Vite is a modern frontend build tool that offers:

- ✅ **Faster startup times** with instant server boot
- ✅ **Hot Module Replacement (HMR)** for quick development iterations.
- ✅ **Optimized builds** with smaller bundle sizes

- Unlike `create-react-app` (CRA), Vite uses ES modules (ESM) and Rollup for faster performance

- CRA has now been deprecated



DEVFORGE
- CODE CRAFTERS -

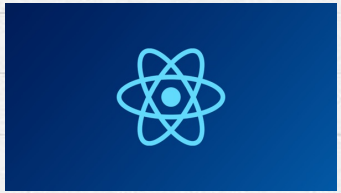
Step 1: Install Node.js (If not Installed)

```
# Confirm that NodeJS is installed (v18+):
node -v

# To install NodeJS, visit the following site:
# https://nodejs.org
```

Step 2: Create a New React 19 Project

```
# Open a Terminal or Prompt to any directory
# Run the following command to create a React 19 project with Vite:
npm create vite@latest my-react19-app --template react
cd my-react19-app

# Replace with your preferred project name
```

Step-by-Step Setup



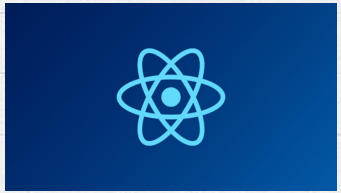
Step 3: Install Dependencies

```
# Run the following command inside your project folder:
npm run dev

# This installs React, React DOM, and other required dependencies.
```

Step 4: Start the Development Server

```
# Run the following command inside your project folder:
npm create vite@latest my-react19-app --template react
cd my-react19-app

# You should see output like this:
VITE v4.0 ready in 300ms
➜ Local: http://localhost:5173/

# Open the link in your browser
```

Step-by-Step Setup



DEVFORGE
- CODE CRAFTERS -

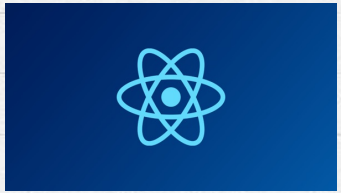
Understanding Project Structure

Your new React 19 project will have the following structure:

```
my-react19-app
├── public/              # Static assets (favicon, images)
├── src/                 # Source code
│   ├── App.jsx          # Main app component
│   ├── main.jsx         # Entry point
│   ├── assets/          # Static files (CSS, images)
│   ├── components/      # Reusable UI components
│   └── styles/          # Stylesheets (optional)
├── index.html           # Root HTML file
├── package.json         # Project metadata & scripts
├── vite.config.js       # Vite configuration
└── README.md            # Project documentation
```

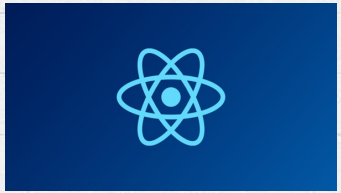
VSCode Plugins:

- Prettier
- ESLint



Lesson Summary

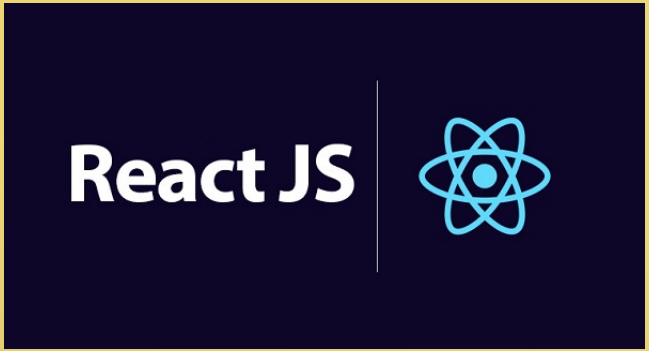
- Installed React 19 with Vite
- Explored project structure
- Configured ESLint and Prettier for better code quality
- Ran the development server successfully



DEVFORGE
CODE CRAFTERS

Lesson 3:

JSX and Virtual DOM Deep Dive

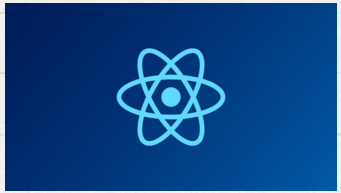


DEVFORGE
CODE CRAFTERS

Learning Objectives

By the end of this lesson, you will:

- ✅ Understand the basics of JSX (JavaScript XML) and its syntax.
- ✅ Learn how JSX gets compiled into JavaScript.
- ✅ Explore the Virtual DOM and how React efficiently updates the UI.
- ✅ Compare the Virtual DOM with the real DOM.

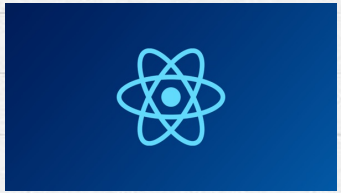


DEVFORGE
- CODE CRAFTERS -

What is JSX?

- JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code inside JavaScript files
- JSX makes React code more readable and declarative
- It is NOT valid JavaScript but it gets **compiled** into JavaScript

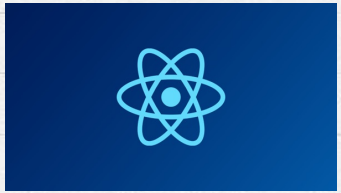
```
# Example JSX:
# Looks like HTML, but it's JavaScript!
const element = <h1>Hello, React 19!</h1>;

# Compiled Version (JavaScript):
const element = React.createElement("h1", null, "Hello, React 19!");
```



1. JSX Must Have One Parent Element:

```
# This will throw an error:
return (
  <h1>Hello</h1>
  <p>React</p>
);


# Instead, wrap elements inside a single parent:
return (
  <div>
    <h1>Hello</h1>
    <p>React</p>
  </div>
);

# You can also use React Fragment <>...</> instead of <div>
```

JSX Syntax Rules



DEVFORGE
CODE CRAFTERS

JSX Syntax Rules

2. JSX Allows Expressions Inside {}:

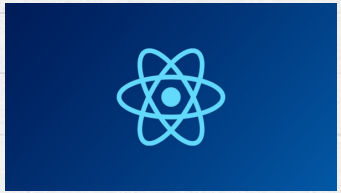
```
# You can embed JavaScript expressions inside JSX using {}:
const name = "Ebinisa";
const greeting = <h1>Hello, {name}!</h1>;
```

3. JSX Uses `classname` Instead of `class`:

```
# Since class is a reserved JavaScript keyword, JSX uses className:
return <h1 className="title">Hello React</h1>;
```

4. Self-Closiing Tags are Required:

```
# In JSX, self-closing tags MUST include /:
return <img src="logo.png" alt="React Logo" />;
```

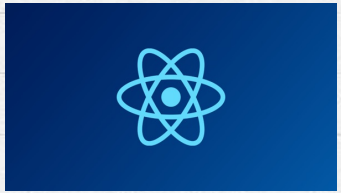


DEVFORGE
- CODE CRAFTERS -

What is the Virtual DOM?

1. The Problem with the Real DOM:

- The real DOM (Document Object Model) is slow because:
  - Every UI update modifies the actual webpage structure
  - Large UI updates cause performance issues

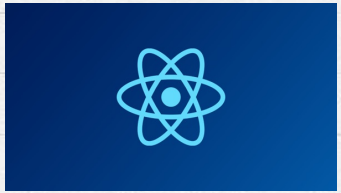


2. React's Solution: The Virtual DOM

- The Virtual DOM (VDOM) is a lightweight, in-memory copy of the real DOM

- How React Uses the Virtual DOM:
  - React creates a virtual representation of the UI
  - When data changes, React updates the Virtual DOM first
  - React compares the new Virtual DOM with the previous one (diffing)
  - React updates only the necessary parts of the real DOM (reconciliation)

What is the Virtual DOM?

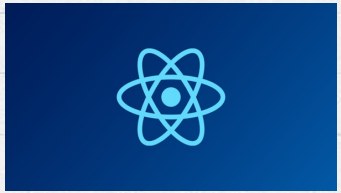


2. React's Solution: The Virtual DOM

- Illustration:

```
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      Clicked {count} times
    </button>
  );
}

# Instead of re-rendering the entire page, React updates only the button's text.
```

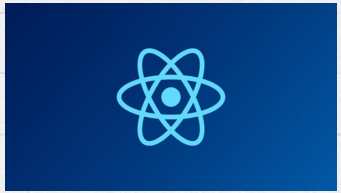
What is the Virtual DOM?



Comparing Virtual DOM vs Real DOM

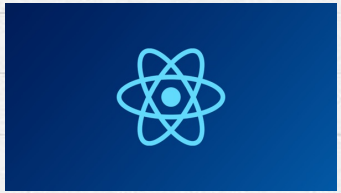
Feature	Real DOM	Virtual DOM
Performance	Slower	Faster
Updates	Directly modifies the UI	Updates only necessary parts
Re-rendering	Full-page refresh	Only updates changed elements
Efficiency	Less optimized	Highly optimized

Comparing Virtual DOM vs Real DOM



Lesson Summary

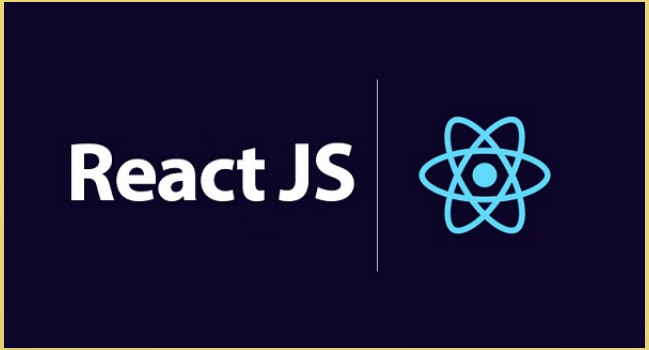
- JSX is a syntax extension for JavaScript that looks like HTML but compiles to JavaScript
- JSX must have one parent element, use `className` instead of `class`, and use `{}` for expressions.
- The Virtual DOM improves performance by updating only the necessary parts of the real DOM



DEVFORGE
CODE CRAFTERS

Lesson 4:

Functional Components & Hooks Overview

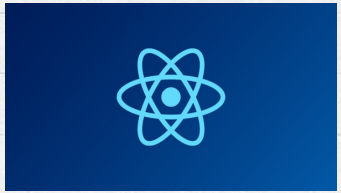


DEVFORGE
CODE CRAFTERS

Learning Objectives

By the end of this lesson, you will:

- ✅ Understand what functional components are and why they are preferred over class components.
- ✅ Learn how to create and use functional components in React 19.
- ✅ Get an overview of React Hooks (`useState`, `useEffect`, `useRef`).
- ✅ Understand the lifecycle of a functional component



DEVFORGE
- CODE CRAFTERS -

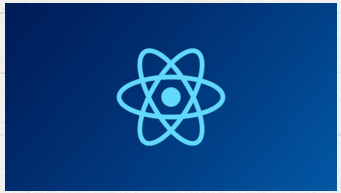
What are Functional Components?

Components, in React, are reusable building blocks of the UI

- Functional Components are JavaScript functions that return JSX
- Class Components were used before React Hooks, but they are now largely deprecated

```
# Basic Functional Component Example:
# This function returns JSX
function Greeting() {
  return <h1>Hello, React 19!</h1>;
}
```

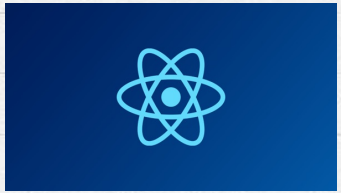
- Functional Components are:
  - Simpler and easier to read
  - Stateless by default but can manage state using Hooks
  - More efficient than class components



Before React 16.8, only Class Components could manage state.

```
# Old Class Component (Before Hooks):
# Note: Hooks provide a cleaner alternative

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <button onClick={this.increment}>
        Clicked {this.state.count} times
      </button>
    );
  }
}
```

Stateful Functional Components with Hooks

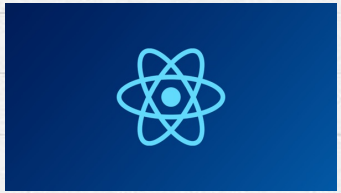


Introducing React Hooks.

- Hooks enable functional components to have state and lifecycle features

```
# Basic useState Hook Example (Modern Approach):

import { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0); // Declare state

  return (
    <button onClick={() => setCount(count + 1)}>
      Clicked {count} times
    </button>
  );
}
```

Stateful Functional Components with Hooks



DEVFORGE
CODE CRAFTERS

Overview of Commonly Used Hooks

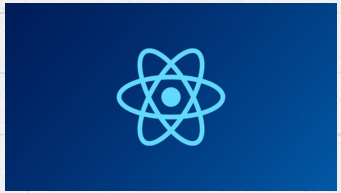
1. `useState` – Managing Component State

```
# Used to store and update values inside a component

const [name, setName] = useState("Ebinisa");
setName("John"); // Updates the name
```

2. `useEffect` – Handling Side Effects

```
# Runs code after the component renders (e.g., fetching data, updating DOM).

useEffect(() => {
  console.log("Component mounted or updated");
}, []);
e
```

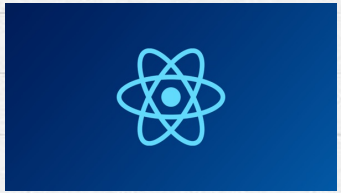
- Runs once when `[]` (empty array) is passed
- Runs on every render if no dependency array is provided
- Runs only when specific values changes if `[dependencies]` are provided



3. useRef – ccessing DOM Elements or Persisting Values

```
# Used for direct DOM manipulation or storing values that don't trigger
# re-renders

onst inputRef = useRef(null);

function focusInput() {
inputRef.current.focus();
}

return <input ref={inputRef} />;
```

Overview of Commonly Used Hooks



DEVFORGE
CODE CRAFTERS

Thanks!

Any questions?