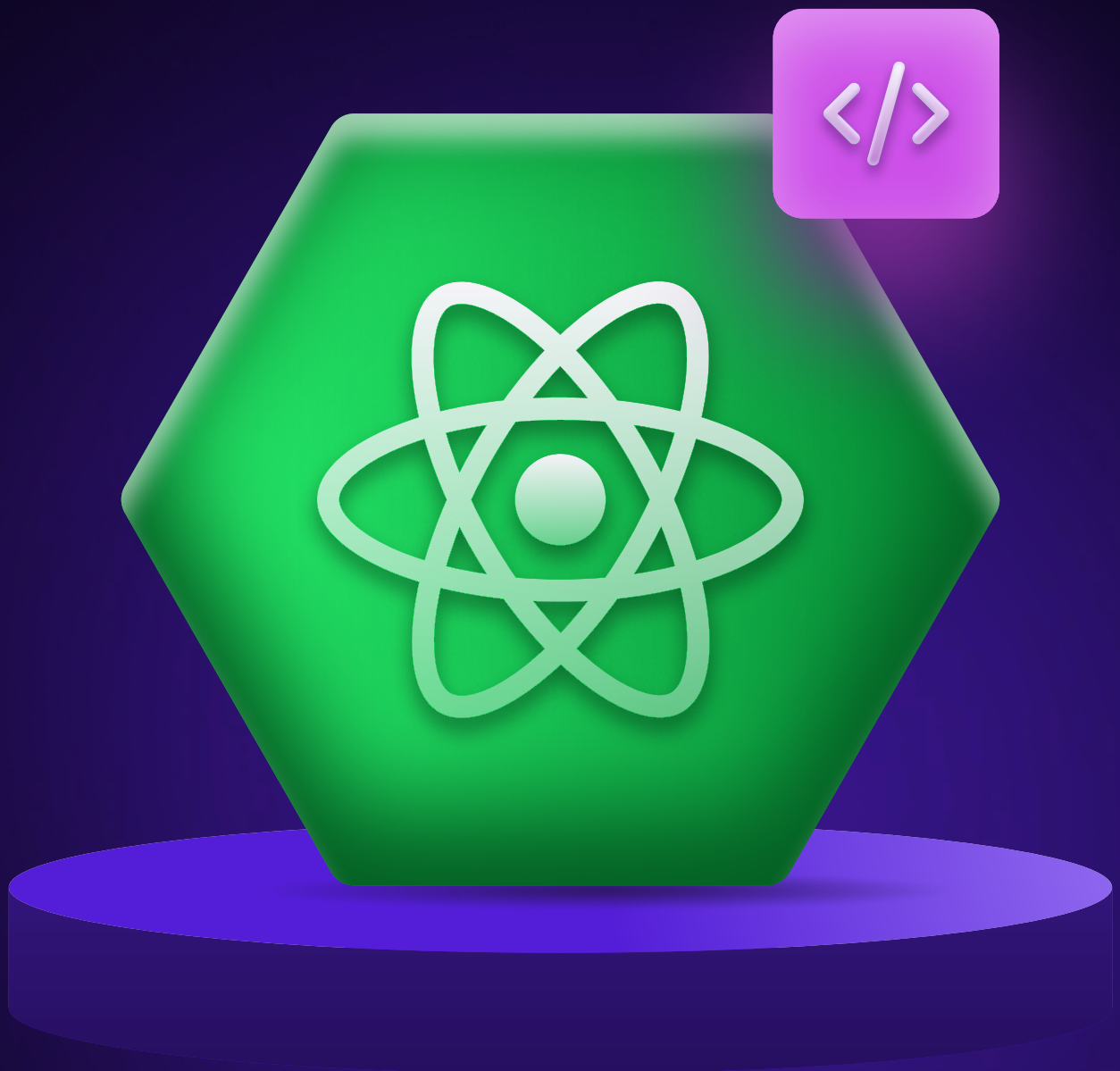
GUVI | HCL

Skill Up. Level Up

React Ebook

Your Go-To Guide for FullStack Development Journey



GUVI | HCL
Skill Up. Level Up

React

React is one of the most popular JavaScript libraries for building dynamic, user-friendly web applications. Whether you're a seasoned developer or just beginning your journey, learning React can open the door to creating modern, interactive user interfaces with ease. This ebook is designed to be your quick and straightforward guide to understanding the fundamentals of React, making it perfect for beginners who want to get started without feeling overwhelmed.

In this ebook, we'll cover the key concepts of React, including its component-based architecture, state and props, JSX syntax, and how to manage application data effectively. You'll also learn how to build reusable UI components, work with hooks, and understand the basics of managing application workflows. By the end, you'll have a solid foundation to create simple yet powerful applications, setting you up for more advanced React projects. Let's dive in!

GUVI | HCL
Skill Up. Level Up

Table of Contents

1. Introduction to React

What is React, and why use it?

React is a JavaScript library created by Facebook for building user interfaces (UIs), particularly single-page applications (SPAs). It allows developers to create reusable UI components that can efficiently update and render as the underlying data changes. React focuses on the view layer (the "V" in MVC) of an application and is widely used due to its performance, flexibility, and developer-friendly features.

Key features

- **Declarative UI, Component-based Architecture, Virtual DOM.**

    - **Component-Based Architecture:** UIs are built using reusable, self-contained components.
    - **Declarative Syntax:** Makes code predictable and easier to debug by describing what the UI should look like for a given state.
    - **Virtual DOM:** Enhances performance by minimizing real DOM manipulations and applying updates efficiently.
    - **One-Way Data Binding:** Ensures predictable application behavior by flowing data in a single direction.
    - **Rich Ecosystem:** React integrates seamlessly with libraries like React Router, Redux, and more.

Applications of React

- **Single-Page Applications (SPAs)** – React enables fast, dynamic SPAs where only content updates without full page reloads (e.g., Facebook,

- Gmail).

- **Interactive Dashboards** – Used for real-time data visualization in admin panels and analytics tools (e.g., stock market apps, CRM dashboards).

- **E-Commerce Websites** – Helps build dynamic product pages, shopping carts, and checkout systems (e.g., Amazon, Shopify).

- **Social Media & Messaging Apps** – React powers real-time feeds, notifications, and chat applications (e.g., Instagram, WhatsApp Web).

- **Content Management Systems (CMS)** – Used in headless CMS and blog platforms for smooth content editing (e.g., WordPress Gutenberg, Ghost).

2. Setting Up the Environment

Installing Node.js and npm.

Step 1: Download Node.js

1. Open your browser and go to the official Node.js website: https://nodejs.org
2. You will see two versions:
   - LTS (Long Term Support) – Recommended for most users.
   - Current – The latest version with new features.
3. Choose LTS for stability and click the download button.

Step 2: Install Node.js

1. Run the Installer
   - Locate the downloaded file (.msi for Windows, .pkg for macOS, .tar.gz for Linux).
   - Double-click to start the installation.

2. Follow the Setup Wizard
   - Click Next.
   - Accept the License Agreement.
   - Click Next to install Node.js in the default location.

3. Ensure npm is selected
   - Node.js automatically installs npm (Node Package Manager).

4. Complete Installation
   - Click Install.
   - Wait for the setup to complete.
   - Click Finish.

Step 3: Verify Installation

After installation, verify that Node.js and npm are installed correctly.

1. Open Terminal or Command Prompt
   - On Windows: Open Command Prompt (cmd) or PowerShell.
   - On macOS/Linux: Open Terminal.

2. Check Node.js Version
   node -v
   If installed correctly, you will see a version number like: **v18.16.0**

3. Check npm Version
   npm -v
   If installed, you will see an npm version number like: **9.5.1**

Step 4: Update npm (Optional)

If npm is outdated, update it using:
npm install -g npm

3. JSX Basics

What is JSX, and how does it work?

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. It is used in React to describe the structure of the user interface. JSX makes the code more readable and allows developers to write UI elements in a syntax similar to HTML while leveraging the power of JavaScript.

JSX gets compiled into React function calls by tools like Babel, which convert JSX into plain JavaScript objects that the browser can understand.

Embedding JavaScript in JSX

JSX allows you to embed JavaScript expressions using curly braces {}. This enables dynamic content rendering and makes the code more interactive.

```
const name = "John";
const element = <h1>Hello, {name}!</h1>;
// Output: Hello, John!
```

You can also use JavaScript expressions for calculations, conditional rendering, and invoking functions:

```
const isLoggedIn = true;
const greeting = (
  <div>
    <h1>Welcome {isLoggedIn ? "back!" : "to the site!"}</h1>
  </div>
);
```

Basic syntax rules and differences from HTML

Although JSX looks similar to HTML, there are some key differences:

1. Use of className Instead of class

In JSX, class is a reserved keyword in JavaScript, so you must use className for applying CSS classes.

```
<div className="container">Hello, World!</div>
```

2. Self-Closing Tags

JSX requires self-closing tags for void elements (like <img />,
, <input />).

```
<img src="logo.png" alt="Logo" />
```

3. CamelCase for Attributes

Attributes in JSX follow camelCase notation, unlike HTML.

```
<button onClick={handleClick}>Click Me</button>
```

4. Components

Functional components vs. class components (focus on functional for simplicity)

React components are the building blocks of any React application. In the past, class components were widely used, but functional components are now the standard due to their simplicity and compatibility with hooks.

Functional Components are plain JavaScript functions that accept props as arguments and return JSX. They are easier to read, write, and maintain compared to class components. Functional components also use hooks like useState and useEffect to handle state and lifecycle methods, making them more powerful.

Example of a Functional Component:

```
function Greeting() {
  return <h1>Hello, World!</h1>;
}
```

Class Components, on the other hand, use ES6 classes and require a render() method to return JSX. While they were previously necessary for managing state and lifecycle methods, functional components with hooks have replaced their need.

Example of a Class Component:

```
class Greeting extends React.Component {
```

```
render() {
    return <h1>Hello, World!</h1>;
  }
}
```

Functional components are now the preferred approach as they are less verbose and work seamlessly with React's modern features.

Props: passing data to components

Props (short for properties) are used to pass data from a parent component to its child components in React. They are read-only, meaning a child component cannot modify the props it receives. Props allow you to create reusable and dynamic components by supplying different data for different use cases.

For example, imagine a Welcome component that greets a user by their name. The name can be passed as a prop:

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

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

In this case, Alice is passed as a name prop to the Welcome component, and the output will be:

Hello, Alice!

State and basic usage with useState

State in React allows components to hold and manage dynamic data. Functional components use the useState hook to add and update state. Unlike props, which are read-only, state is mutable and controlled within the component itself.

The useState hook takes an initial value and returns an array containing two elements:

1. The current state value.
2. A function to update the state.

Here's a simple example of a counter component using useState:

```
import React, { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0); // Initialize state with 0
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
export default Counter;
```

- **useState(0)** initializes the count state to 0.
- **setCount** is the function used to update the count value.
- Clicking the "Increment" button triggers an update, and React re-renders the component with the new value.

Output

Count: 0

[Button: Increment]

Each click on the button increases the count by 1, and the UI updates dynamically.

This is a powerful feature of React that allows components to respond to user interactions and reflect changes in real time.

5. Handling Events

Adding event handlers in JSX

In React, event handling is very similar to how it is done in plain HTML, but there are a few key differences. Events in React are written in camelCase (e.g., onClick, onChange) instead of lowercase (e.g., onclick, onchange), and instead of passing a string of JavaScript code, you pass a function as the event handler.

Event handlers are used to make components interactive by responding to user actions, such as clicks, typing, or submitting forms.

Example: Handling button clicks

You can handle button clicks in React by adding the onClick event to a button element. The event handler is a function that gets executed when the button is clicked.

Here's an example:

```
import React, { useState } from "react";
function ClickCounter() {
  const [count, setCount] = useState(0); // Initialize count to 0
  const handleClick = () => {
    setCount(count + 1); // Update count when button is clicked
  };
  return (
    <div>
```

```
    <p>You clicked {count} times.</p>
        <button onClick={handleClick}>Click Me</button>
      </div>
    );
  }
  export default ClickCounter;
```

Explanation:

1. The onClick attribute is used to attach the handleClick function to the button.
2. Each time the button is clicked, the handleClick function is executed, and the setCount function updates the count state.
3. React re-renders the component to show the updated count.

Example: Handling button clicks

Handling form inputs in React is a bit different because you typically make the input a **controlled component.** This means the input's value is controlled by React state.

Here’s an example of a simple input form where a user can type their name:

```
import React, { useState } from "react";
function NameForm() {
  const [name, setName] = useState(""); // Initialize name to an empty
string
  const handleChange = (event) => {
    setName(event.target.value); // Update state when input changes
  };
  const handleSubmit = (event) => {
```

```
  event.preventDefault(); // Prevent the page from reloading
    alert(`Hello, ${name}!`);
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Enter your name:
        <input type="text" value={name} onChange={handleChange} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
export default NameForm;
```

Explanation:

1. The onChange event is used to capture user input. Whenever the user types in the input field, the handleChange function updates the name state.
2. The value attribute of the input field is tied to the name state, making it a controlled component.
3. When the form is submitted, the handleSubmit function prevents the default behavior (page reload) and displays an alert with the entered name.

6. Rendering Lists and Conditional Rendering

Using .map() to render lists

In React, the .map() function is used to iterate over an array and render a list of elements dynamically. Each child element in the list must have a unique key prop to help React identify changes efficiently.

```
function NameList() {
  const names = ["Alice", "Bob", "Charlie"];
  return (
    <ul>
      {names.map((name, index) => (
        <li key={index}>{name}</li> // `key` prop added for uniqueness
      ))}
    </ul>
  );
}
export default NameList;
```

- The .map() function generates a <li> for each name in the names array.
- The key prop ensures React can efficiently manage updates to the list.

Conditional rendering with if statements, ternary operators, and short-circuiting

Conditional rendering allows React to decide what to display based on conditions. You can achieve this using if statements, ternary operators, or

short-circuiting.

1. Using if Statements

This is useful for rendering different components or elements:

```
function Greeting({ isLoggedIn }) {
  if (isLoggedIn) {
    return <h1>Welcome back!</h1>;
  }
  return <h1>Please log in.</h1>;
}
```

2. Using Ternary Operators

A concise way to render elements based on a condition:

```
function Greeting({ isLoggedIn }) {
  return <h1>{isLoggedIn ? "Welcome back!" : "Please log in."}</h1>;
}
```

3. Using Short-Circuiting

For simpler conditions where only one element needs to be rendered:

```
function Notification({ hasMessage }) {
  return <>{hasMessage && <p>You have new messages!</p>}</>; // Render
only if true
}
```

7. Basic Hooks

Introduction to hooks

Hooks are special functions introduced in React 16.8 that allow you to use state and other React features in functional components. Before hooks, state and lifecycle methods were only available in class components, but hooks made functional components more powerful and the preferred approach in modern React development.

Why Use Hooks?

- **Simplified Code:** Hooks eliminate the need for class components, reducing boilerplate and making components easier to read and maintain.
- **Reusability:** Custom hooks can encapsulate and reuse logic across components.
- **Flexibility:** Hooks like useState, useEffect, useContext, and more provide solutions for state management, side effects, and context handling.
- **Detailed focus on useState and useEffect.**

useState

useState is a hook used to add state to functional components. It initializes a state variable and provides a function to update it.

Syntax

const [state, setState] = useState(initialValue);
where,
state: The current state value.
setState: A function to update the state.
initialValue: The initial value of the state.

useEffect

useEffect is a hook used to handle side effects in functional components. Side effects include data fetching, updating the DOM, and subscriptions. It runs after the component renders and can also clean up effects when the component unmounts.

Syntax

```
useEffect(() => {
  // Effect logic here
  return () => {
    // Cleanup logic here (optional)
  };
}, [dependencies]);
```

Effect logic: Code to be executed after rendering.

Cleanup logic: Optional function to clean up effects (like removing event listeners).

[dependencies]: Array of variables that trigger the effect when they change. If empty, the effect runs only once (like componentDidMount).

8. Conclusion and Next Steps

Recap of what was covered

In this ebook, we have tried to cover all of the fundamental concepts of React that you should know as a beginner. We have talked about React is a powerful JavaScript library for building dynamic and interactive user interfaces, especially useful in the MERN stack for seamless integration with back-end APIs.

It uses **JSX** (JavaScript XML) to write HTML-like syntax within JavaScript, enabling developers to embed JavaScript logic directly in the UI. Modern React development favors **functional components** over class components due to their simplicity and compatibility with **hooks** like useState and useEffect. Props allow data to be passed from parent to child components, while state, managed with useState, handles dynamic data within components.

React simplifies rendering dynamic content with .map() for lists and conditional rendering using if statements, ternary operators, or short-circuiting (&&). The useEffect hook is used for side effects like fetching data or subscriptions. With event handlers like onClick and onChange, React makes user interactions intuitive. Its modular, reusable component-based architecture and rich ecosystem make it an essential tool for modern web development.

Now, these are just basics that we have talked about in this ebook, there are indeed advanced concepts like **React Router, Context API, and third-party libraries** which you can explore in the course offered by GUVI.

Check out this course on React which talks about the major pointers you should know to get started with React and work on top projects.

GUVI | HCL
Skill Up. Level Up

About GUVI

GUVI (Grab Ur Vernacular Imprint), an IIT-Madras Incubated Company is First Vernacular Ed-Tech Learning Platform. Introduced by Ex-PayPal Employees Mr. Arun Prakash (CEO) and Mr. SP Balamurugan, and late Sridevi Arun Prakash (co-founders) and driven by leading Industry Experts, GUVI empowers students to master on-demand technologies, tools, and programming skills in the comfort of their native languages like Hindi, Tamil, Telugu, English etc.

Personalized Solutions

End-to-end personalized solutions in online learning, upskilling, and recruitment.

Empowering Learners

Empowering learners with tech skills in native languages.

Gamified Learning

Gamified, bite-sized videos for an interactive learning experience.

Accreditations & Partnerships

Google for Education Partner

Want to know more about GUVI? Visit our website today!

About Zen Class

Zen Class is GUVI's specially curated learning platform that incorporates all the Advanced Tech Career Courses like Full Stack Web Development, Data Science, Automation Testing, Big Data & Cloud Analytics, UI/UX Designing, and more. Zen Class provides the best industry-led curriculum programs with assured placement guidance

Zen Class mentors are experts from leading companies like **Google, Microsoft, Flipkart, Zoho & Freshworks.** Partnered with 600+ tech companies, your chances of getting a high-paying tech job at top companies increases.

Why choose Zen Class?

- Assured Placement Guidance
- IIT-M Pravartak Certification for Advanced Programming
- Ease of Learning in native languages such as Hindi & Tamil
- A rich portfolio of real-world Projects
- Self-paced Online Classes
- 600+ Hiring Partners
- Excellent Mentor Support
- Easy EMI options

Do check our Zen Class - Full Stack Development Program which talks about the fundamentals of JavaScript, MongoDB, ExpressJS, NodeJS, ReactJS, AWS, and the core tools and technologies you must know. You'll also learn working on real-world projects, and get placement guidance.

GUVI | HCL
Skill Up. Level Up

Thank You

www.guvi.in