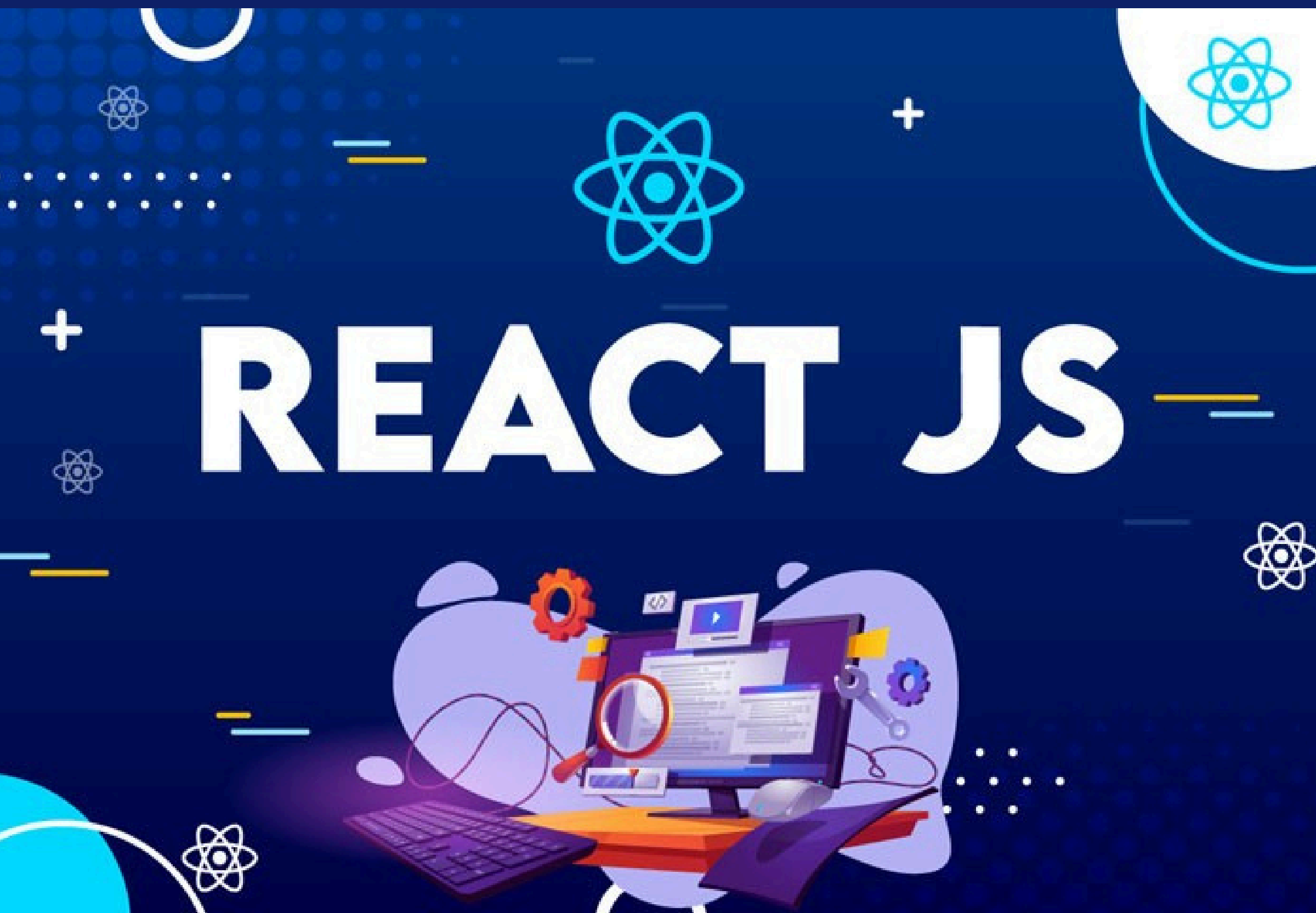
Day 1

What Is REACT JS



What is React.js?

React.js is a popular JavaScript library for building user interfaces, particularly single-page applications (SPAs) where content dynamically updates without needing a full page reload.

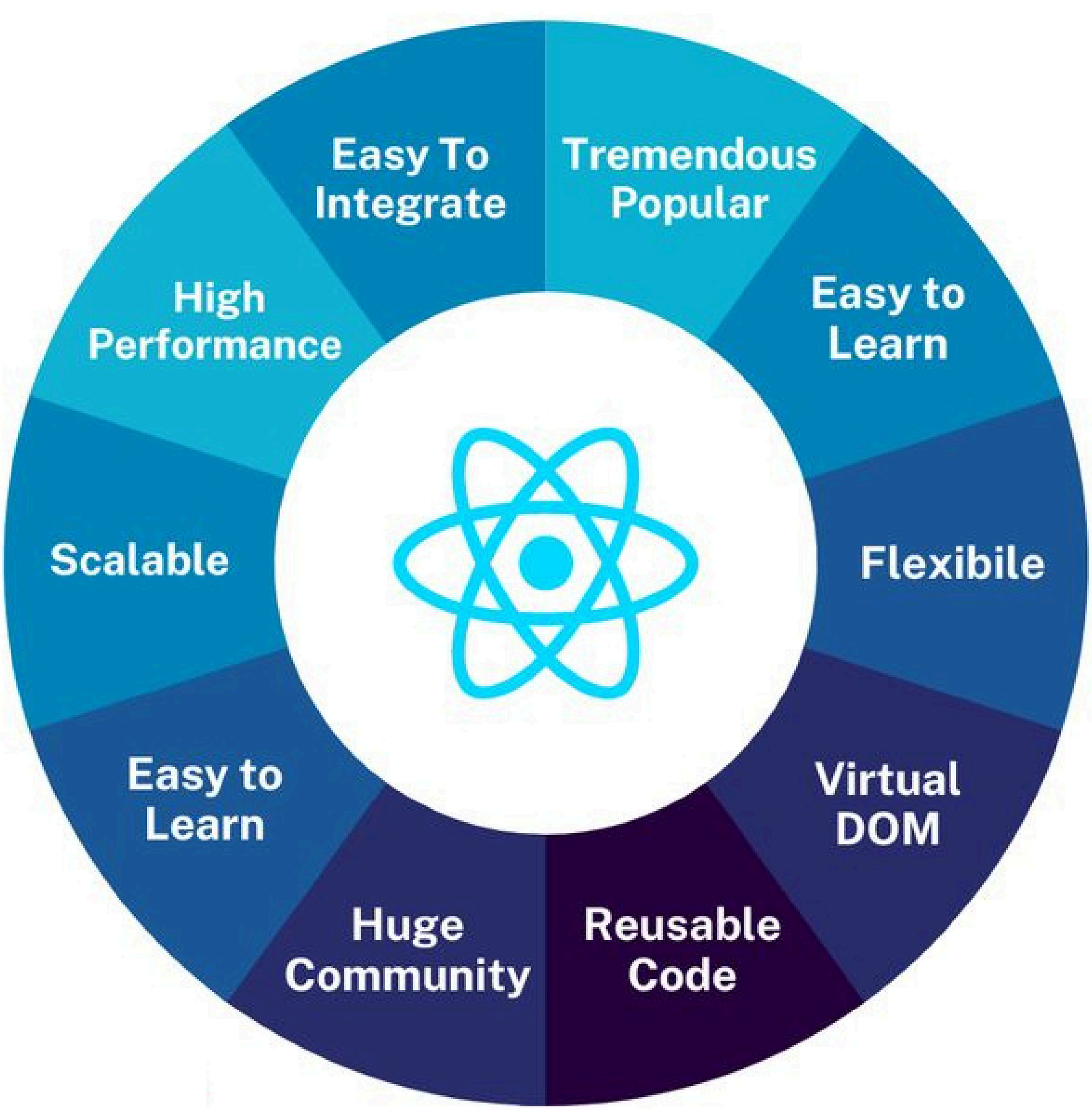
Developed and maintained by **Facebook**, React is known for its efficiency, flexibility, and ease of use.

React.js is an open-source JavaScript library used for building fast, interactive user interfaces for web applications.

It allows developers to create large web applications that can update and render efficiently with changing data.

Why Use React.js?

REACT.JS IS MARKET'S FAVORITE !



1. Component-Based Architecture

Think of React components like building blocks. You create small pieces (components) that you can use again and again throughout your app.

React follows a **component-based architecture**, allowing you to build encapsulated components that manage their own state and compose them to create complex UIs.

Components can be reused across different parts of the application, reducing redundancy and ensuring consistency.

For example, a button you use on one page can be reused on other pages without having to rewrite the code.

```jsx
// Button.js
import React from 'react';

function Button({ label, onClick }) {
  return <button onClick={onClick}>{label}</button>;
}

export default Button;

// App.js
import React from 'react';
import Button from './Button'; // Button component is imported

function App() {
  const handleClick = () => alert('Button clicked!');

  return (
    <div>
      <Button label="Click Me!" onClick={handleClick}/>
      <Button label="Submit" onClick={handleClick} />
    </div>
  );
}

export default App;
```

Here, the **Button component** is reused with different properties, demonstrating how components in React can be modular and reusable.

2. Virtual DOM for Performance Optimization

React uses a special technology called the Virtual DOM to keep your app fast.

The Virtual DOM is a lightweight copy of the **real DOM**, to optimize rendering. When the state of an object changes, React first updates the Virtual DOM, then efficiently updates the real DOM by applying only the necessary changes.

This approach minimizes the number of updates to the real DOM, significantly boosting performance.

For Example:

```jsx
import React, { useState } from 'react';

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

In this example, only the part of the DOM that contains the count value is updated when the button is clicked, thanks to React's Virtual DOM.

3. Unidirectional Data Flow

React makes it easy to follow how data moves through your app.

React follows a **unidirectional data flow**, meaning data flows in **one direction**, from parent to child components via **props**.

This makes the app easier to debug and understand, as changes follow a predictable pattern.

With a single **source of truth (all the state or data for your application is centralized in a single location)**, debugging becomes straightforward, and you can trace changes through the entire application.

For Example;

```jsx
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  const message = 'Hello from Parent!';

// ChildComponent now has access to the value in the 'message' variable
  return <ChildComponent message={message} />;
}

export default ParentComponent;

// ChildComponent.js
import React from 'react';

function ChildComponent({ message }) {
  return <p>{message}</p>;
}

export default ChildComponent;
```

message is passed as a prop to the ChildComponent

message is then used as a jsx value

Here, data flows from **ParentComponent** to **ChildComponent** via the message prop, illustrating unidirectional data flow.

4. Easy Integration with Other Libraries and Frameworks

React is designed to be flexible and can be easily integrated with other libraries and frameworks, such as Redux for state management, React Router for navigation, or even non-React libraries.

React can be embedded in an existing project or used to create new projects, making it versatile and suitable for a wide range of applications.

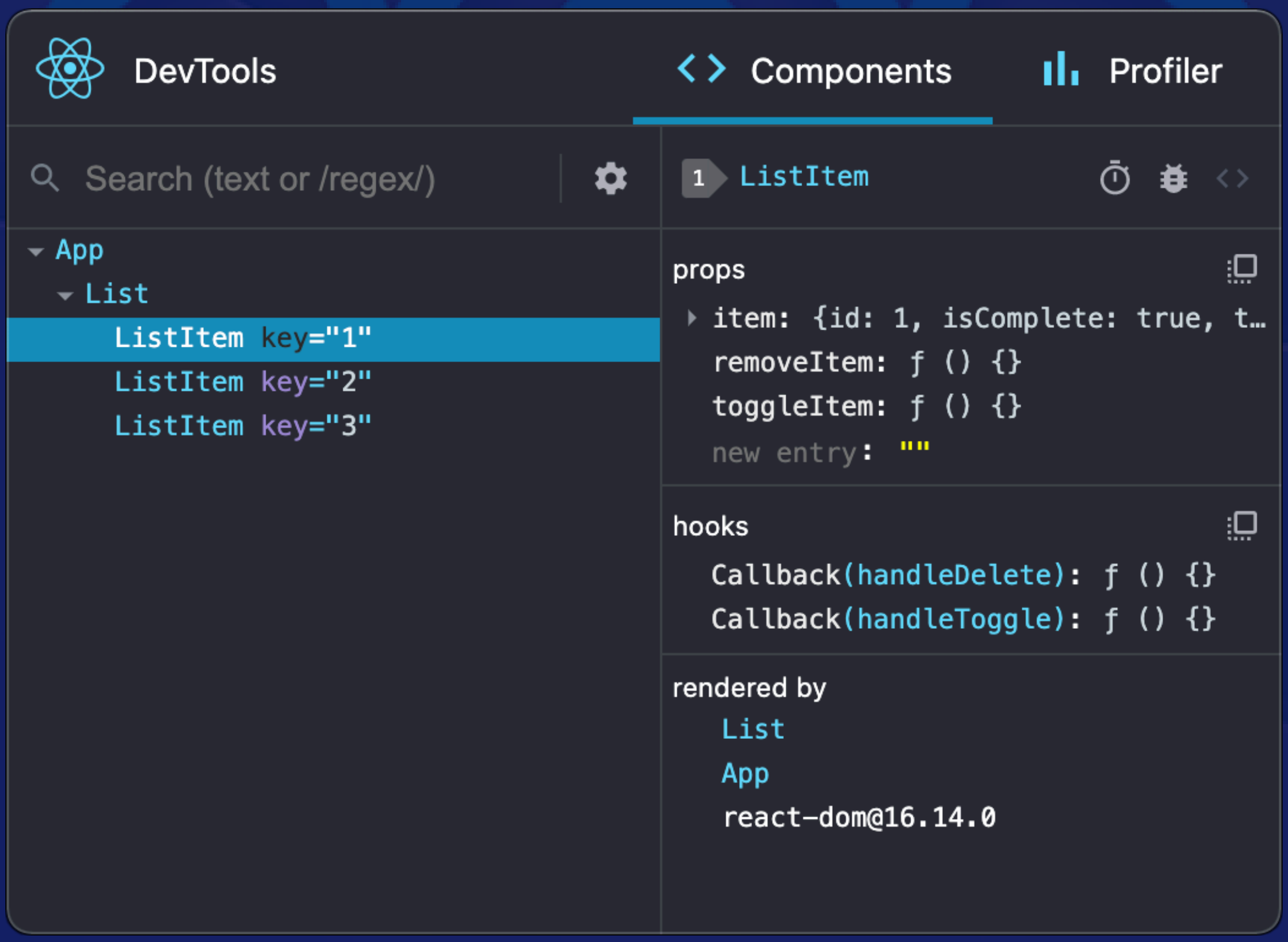
```jsx
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Route path="/" exact component={Home} />
      <Route path="/about" component={About} />
    </Router>
  );
}
```

This example shows how to add navigation between different pages using the ‘react-router-dom’ library

5. Easy Debugging with Developer Tools

React has powerful developer tools that help you see how your app is working, making debugging easier.

React has excellent developer tools, such as **React Developer Tools** for Chrome and Firefox, allowing developers to inspect component hierarchies, view props, and state, and debug more effectively.



6. Support for Server-Side Rendering (SSR)

React supports **Server-Side Rendering (SSR)**, which helps **improve SEO** and **performance** by rendering components on the server before sending them to the client.

SSR allows search engines to index pages more effectively, and users experience faster initial loading times.

7. Code Reusability and Maintenance

With **React's modular approach**, components can be reused across different parts of the application, reducing redundancy and making code more maintainable.

This approach makes it easier to update and maintain applications, reducing the cost and time of development.

9. JSX for Readable Syntax

React uses **JSX (JavaScript XML)**, an extension that allows writing **HTML-like** syntax within JavaScript. JSX makes the code more readable and easier to debug. It looks like HTML but has the power of JavaScript.

Syntax for JSX:

```
1 const element = <h1>Hello, World!</h1>;
```

Overview of React.js Ecosystem

The React ecosystem is a collection of **tools**, **libraries**, and **frameworks** that enhance the core functionalities of React, making it easier to build complex, efficient, and scalable web applications.

Some of the Most Import part of the ecosystem are:

1. State Management: State management is a critical concept in React.js that deals with how data flows through a React application.

In React, **state** refers to a JavaScript object that holds information about the component. This data determines how the component renders and behaves.

State is managed within the component itself and can change over time. For example, in a to-do list application, the state might contain an array of tasks. When a user adds a new task, the state is updated, and the UI re-renders to reflect the new task.

Types of State Management in React

- **Local State:** Managed within a single component using the useState hook.
- **Global State:** Managed across multiple components and requires external libraries like Redux, MobX, or Context API.
- **Server State:** Data that comes from an external server that must be integrated with the app's local state.
- **URL State:** State that exists on URLs, including the pathname and query parameters.

2. Routing: React Router is a popular library for managing navigation and routing in React applications. It allows developers to handle dynamic routing, nested routes, and complex route configurations easily.

3. Styling: Styling in React.js can be done in various ways, providing flexibility and control over how you want your application to look and feel.

Styling can be done with:

- Styled Components: A library for writing CSS in JavaScript with a focus on component-level styling.
- Emotion: A performant and flexible CSS-in-JS library that provides powerful and flexible styling capabilities, similar to Styled Components.
- CSS Modules: A CSS file in which all class names and animation names are scoped locally by default, helping to avoid style conflicts.
- Inline Styling: Inline styling in React allows you to apply styles directly to elements using the style attribute.
- CSS Stylesheets: You can use traditional CSS stylesheets to style your React components. This method involves creating **.css** files and importing them into your React component.

4. Data Fetching: Allows you to get data from external sources like APIs and servers to display dynamic content.

Data can be fetched in React using libraries like Axios, SWR,React Query, Javascript Promise, etc...

5. Build Tools and Frameworks: React alone doesn’t provide a complete set of tools for building complex applications, that’s where various frameworks and libraries come in. They extend the capabilities of React and help developers create robust, scalable, and efficient applications more easily.

Examples of these tools are:

- **Next.js:** A React framework that provides server-side rendering (SSR), static site generation (SSG), and many other features for building fast and scalable web applications.
- **Vite:** A fast build tool that serves as an alternative to Create React App, providing faster development and optimized production builds.

Learned well about what React.js is? or Do you have questions or clarifications about any concept learned today?

Engage me in the comments, I will be happy to help further.

Don’t worry if you are seeing some of these terminologies or snippets for the first time, I am pretty sure that before the end of these 30 days of Learning Reactjs, you would have known better.

I hope you found this material useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be helpful to someone 👌

kodemaven-portfolio.vercel.app

Hi There!

Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi

kodemaven-portfolio.vercel.app