React

**1. react**

* **Core React library** for building user interfaces.
* Handles **component-based UI**, **state**, **props**, **hooks**, etc.
* Works in the **browser** and is required by all React-based projects.

🧠 Think of this as the brain of React.

**2. react-dom**

* Bridges **React (the brain)** with the **actual DOM in the browser**.
* Provides methods like:
* Only used in **web applications**, not mobile apps.

ReactDOM.createRoot(document.getElementById("root")).render(<App />);

🧠 Think of this as the hands that control what users see in the browser.

**3. react-native**

* Used to build **native mobile apps** (iOS + Android) using React.
* Instead of rendering HTML (<div>, <span>), it renders **native components** like <View>, <Text>.
* Doesn't use react-dom.

🧠 Think of this as React dressed up for mobile platforms.

**✅ web-vitals — What It Is and Why It Matters**

web-vitals is a **small JavaScript library** from Google designed to help developers **measure key performance metrics** for user experience in web applications.

It focuses on **Core Web Vitals**, which are essential for monitoring and improving **real-world performance**.

**📦 Why is it in your React project?**

When you create a React app using **Create React App (npx create-react-app)**, it includes web-vitals by default so you can easily track performance.

**📊 What Metrics Does web-vitals Measure?**

| **Metric** | **What it measures** | **Importance** |
| --- | --- | --- |
| **LCP** (Largest Contentful Paint) | Loading speed | How fast main content appears |
| **FID** (First Input Delay) | Interactivity | Time between user action and browser response |
| **CLS** (Cumulative Layout Shift) | Visual stability | Prevents layout jumps |
| **INP** (Interaction to Next Paint) | Responsiveness | Successor to FID |
| **TTFB** (Time to First Byte) | Backend response time | Time taken to receive the first byte |

**🛠️ Example Usage in React**

In index.js or main.jsx, you’ll often see:

import reportWebVitals from './reportWebVitals';

reportWebVitals(console.log); // Logs metrics to console

**✅ What is react-scripts?**

react-scripts is a core part of **Create React App (CRA)** — it contains the **configuration and scripts** needed to build, run, and test a React project **without manually setting up Webpack, Babel, ESLint, etc.**

**🔧 What does it do?**

It provides the commands that you typically run in a CRA project:

| **Command** | **What it does** |
| --- | --- |
| npm start | Runs the dev server with hot-reload |
| npm run build | Creates an optimized production build |
| npm test | Runs tests using Jest |
| npm run eject | Exposes the underlying config so you can customize it (irreversible) |

These commands internally call scripts from the react-scripts package.

**📁 Where is it located?**

In your package.json, you'll see:

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

}

So when you type npm start, you're actually running react-scripts start.

**🧠 Why is it useful?**

* 🏗 **Zero config** — no need to mess with Webpack or Babel yourself.
* 🧼 Handles linting, compiling, bundling, HMR (Hot Module Replacement).
* 🚀 Good for fast prototyping and learning React.

**❗ Can I remove it?**

Only if:

* You **eject** using npm run eject (not reversible), or
* You build your own setup with tools like **Vite**, **Webpack**, or **Next.js**.

Let me know if you’re planning to move beyond react-scripts (e.g., Vite or Next.js), and I can guide you through that transition.

**✅ What is eslintConfig in package.json?**

The eslintConfig field in your package.json defines the **configuration for ESLint**, which is a popular **JavaScript linter** used to:

* Catch errors early (like undefined variables, unused imports)
* Enforce coding style and best practices
* Improve code readability and maintainability

**🔍 Where it's found**

{

"eslintConfig": {

"extends": ["react-app", "react-app/jest"]

}

}

This config tells ESLint **what set of rules** to follow. In this case:

* "react-app": Adds rules for regular React code.
* "react-app/jest": Adds rules specifically for test files using Jest.

**📁 Alternative: Using a separate .eslintrc file**

You can also put ESLint config in its own file, like .eslintrc.json, .eslintrc.js, or .eslintrc.yml. Example:

// .eslintrc.json

{

"extends": ["eslint:recommended", "plugin:react/recommended"],

"rules": {

"no-console": "warn",

"semi": ["error", "always"]

}

}

**🛠 What You Can Define in eslintConfig**

| **Key** | **Purpose** |
| --- | --- |
| extends | Base rules to inherit (like react-app or eslint:recommended) |
| rules | Custom rules to enable/disable or adjust |
| env | Environment settings (e.g. browser, node, jest) |
| parserOptions | What syntax features ESLint should understand (like JSX or ES6 modules) |
| plugins | Third-party rule sets (e.g., react, jsx-a11y) |

**📌 Summary**

eslintConfig is just a convenient place to tell ESLint:

* **How to lint your code**
* **What rules to apply**
* **What environment you're working in (like React or Jest)**

Let me know if you'd like help customizing ESLint rules or integrating it with tools like Prettier!