**Remove-Item -Recurse -Force node\_modules; Remove-Item -Force package-lock.json**

**1.Imp topics to start react and what is react**

Ans: Js fundamentals, array, arrow fun implicit return, map, filter, reduce, object, mutable and immutable and DEstructuring

**React** is a **JavaScript library** for building **fast and interactive** user interfaces.

**Key Points:**

* Used for **Single Page Applications (SPA)**.
* Allows **reusable UI components**.
* Uses **Virtual DOM** for better performance.
* Follows a **component-based architecture**.

**2: What is Emmet?**

A: Emmet is a tool that helps web developers to write HTML and CSS code faster. It lets you type short codes, and it automatically expands them into full code.

3. **What is CDN? Why do we use it?**

A **Content Delivery Network (CDN)** is a network of **geographically distributed servers** that work together to deliver internet content quickly.

**Why do we use it?**

* Improves **website speed and performance**.
* Reduces **server load** by caching content.
* Enhances **availability and reliability**.
* Provides **better security** against DDoS attacks.

.

**4. Why is React known as React?**

React is called **"React"** because it **reacts** efficiently to UI changes by updating components quickly. It ensures smooth and fast updates on web pages. React was developed by **Facebook**.

**5. What is crossorigin in script tag?**

A. The crossorigin attribute in the <script> tag is used for loading scripts from another domain while controlling security and caching.

Values:

* anonymous: Requests script without cookies or authentication.
* use-credentials: Requests script with cookies and authentication (server must allow it).

This helps manage cross-origin security policies when fetching external scripts

<script crossorigin="anonymous|use-credentials">

**6: What is difference between React and ReactDOM?**

* **React**: A **JavaScript library** for building **user interfaces**. It provides core features like React.createElement(), React.Component, and React.Children.
* **ReactDOM**: A **library that connects React to the DOM**. It provides methods like ReactDOM.render() for rendering components and ReactDOMServer for **server-side rendering**.

**7. What is difference between react.development.js and react.production.js files via CDN?**

* **react.development.js**: This is the development version. It includes helpful warnings, error messages, and debugging tools to help developers. However, it is larger in size and slower.
* **react.production.js**: This is the optimized version for production. It removes extra debugging features, making it **smaller** and **faster** for better performance in real applications.

Use **development** while coding and **production** when deploying your app. 🚀

**8. The async and defer attributes in the <script> tag control how the browser loads and executes JavaScript files.**

1. **async (Asynchronous Loading)**
   * **Loads while HTML is being processed.**
   * **Executes immediately once downloaded.**
   * **Best for independent scripts.**
   * **<script src="script.js" async></script>**
2. **defer (Deferred Loading)**
   * **Loads while HTML is being processed.**
   * **Executes after the full HTML is loaded.**
   * **Best for scripts that depend on the page structure.**
   * **<script src="script.js" defer></script>**

**🔹 async: Loads and runs as soon as possible (may block rendering).  
🔹 defer: Loads in the background and runs after HTML is fully loaded.**

**9.Difference Between Library and Framework**

* **Library**: A collection of **pre-written code** for specific tasks. Developers **call** library functions as needed. Example: **React**.
* **Framework**: A structured environment that dictates the architecture of an application. It **calls** the developer’s code. Example: **Angular**.

🔹 **Library = You control the flow.**  
🔹 **Framework = It controls the flow.**

.

**10. createRoot and CreateElement and root.render**

createElement lets you create a React element. It serves as an alternative to writing [JSX.](https://react.dev/learn/writing-markup-with-jsx) const element = createElement(type, props, ...children)

createRoot lets you create a root to display React components inside a browser DOM node. const root = createRoot(domNode, options?)

render=Call root.render to display a piece of [JSX](https://react.dev/learn/writing-markup-with-jsx) (“React node”) into the React root’s browser DOM node.

**11 What Does "React Creates a Virtual DOM in Memory" Mean?**

The **Virtual DOM (VDOM)** is a lightweight copy of the real **DOM** that React creates and updates in memory. Instead of changing the actual webpage (real DOM) directly, React updates this **virtual version first**, making updates faster and more efficient.

**How It Works:**

1. **React creates a Virtual DOM** as a copy of the real DOM.
2. **Changes are made in the Virtual DOM first**, not directly in the real page.
3. **React compares the Virtual DOM with the previous version** (using a process called **"diffing"**).
4. **Only the changed parts are updated** in the real DOM (not the entire page).

**Npm install -D parcel what is -D**

There are two types of dependencies / packages we can install

1 is the dev dep – is used for development

1 is the normal – is used for production

..

What is package.json = is a configuration for are npm

Node module its like a data base its content the actual data

**12. transitive dependencies**

Ans: In React (or any Node.js project), **transitive dependencies** are the packages that your direct dependencies rely on.

For example:

* You install **React** (npm install react).
* React depends on other libraries like **scheduler** and **react-reconciler**.
* These dependencies are installed automatically, even though you didn't add them manually.

So, **transitive dependencies** are the hidden dependencies that come with the libraries you install.

**13 if a won’t put node\_modules in git only I put package.json and pacage-lock.json**

A : If you don’t add node\_modules to Git but keep package.json and package-lock.json, you can reinstall everything using npm install.

* **npm** → Installs packages
* **npx** → Runs packages without installing them globally (e.g., npx parcel index.html).

# \_Episode 02 - Igniting Our App\_

**14.what is NPM**

Ans: npm (Node Package Manager) is the world's largest software registry, used to install, manage, and share JavaScript packages.

* It is the default package manager for Node.js.
* The npm registry hosts open-source packages for web apps, mobile apps, and more.
* Alternative: **Yarn**.

**How to initialize npm?**

* npm init → Starts package setup.
* npm init -y → Skips setup, auto-creates package.json.

**15: What is Parcel/Webpack? Why do we need it?**

**A:** Parcel and Webpack are **bundlers** used in JavaScript projects. They process and optimize code for better performance.

**Why do we need them?**

* **Bundle multiple files** into one.
* **Optimize code** (minify, compress, tree shaking).
* **Support modern JavaScript** features (Babel, TypeScript).
* **Handle assets** (CSS, images, etc.).

**Parcel vs Webpack**

* **Parcel** → Zero-config, fast, simpler for beginners.
* **Webpack** → Highly customizable, powerful but requires more setup.

**16.What is the .parcel-cache folder?**

The **.parcel-cache** folder is created by **Parcel** to store cached build files.

**Why is it needed?**

* **Speeds up builds** by reusing previous results.
* **Reduces compilation time** by avoiding unnecessary reprocessing.
* **Stores optimized assets** like minified JS, CSS, and images.

You can delete .parcel-cache if needed; Parcel will regenerate it on the next build. 🚀

**17. what is this minify, compress, tree shaking?**

Ans: **1. Minify**

* Removes unnecessary spaces, comments, and formatting from code.

**2. Compress**

* Reduces file size by optimizing the code further (e.g., renaming variables, removing redundant code).

**3. Tree Shaking**

* Removes **unused** (dead) code to make the final bundle smaller.

**18: What is npx?**

**A:** npx (Node Package eXecute) is a command that runs **Node.js packages** without installing them globally.

**Why use npx?**

* Runs **one-time commands** without global installation.
* Ensures you're using the **latest** package version.
* Useful for tools like npx create-react-app or npx parcel index.html.

**19. What is difference between dependencies vs devDependencies?**

A: dependencies : Packages required by your application in production. devDependencies : Packages that are only needed for local development and testing.

**20: What is Hot Module Replacement (HMR)?**

**A:** **Hot Module Replacement (HMR)** is a feature in bundlers like **Parcel** and **Webpack** that **updates changes in your code without refreshing the entire page**.

**Why is HMR useful?**

* **Faster development** → No full reload needed.
* **Preserves state** → Keeps app data intact during updates.
* **Efficient updates** → Only modified modules reload, saving time.

**Example:**

When editing a React component, HMR updates only that component instead of reloading the whole page. 🚀

**21: List your favorite 5 superpowers of Parcel and describe any 3.**

**Top 5 Superpowers of Parcel:**

1. **Zero Configuration**
2. **HMR (Hot Module Replacement)**
3. **Fast Bundling with Caching**
4. **Code Splitting**
5. **Tree Shaking**

**Describing 3 Superpowers:**

1️⃣ **Zero Configuration** 🛠️

* Unlike Webpack, which requires a complex setup, Parcel works **out of the box** without any config files. Just run npx parcel index.html, and it automatically detects and bundles everything.

2️⃣ **HMR (Hot Module Replacement)** 🔥

* HMR updates only the changed modules **without refreshing** the whole page. This keeps the app’s state intact and speeds up development.

3️⃣ **Fast Bundling with Caching** ⚡

* Parcel **caches** previously built files, so on subsequent builds, it **only processes changed files**, making it **super fast** compared to other bundlers. 🚀

and some of more cool features of Parcel are:

- File Watching Algorithm - written in C++

- Caching - Faster Builds

- Image Optimization

- Minification

- Bundling

- Compress

- Consistent Hashing

- Code Splitting

- Differential Bundling - support older browsers

- Diagnostic

- Error Handling

- HTTPs

- Tree Shaking - remove unused code

- Different Build for dev and prod bundles

**22: What is .gitignore?**

**A:** .gitignore is a file in a Git repository that tells Git **which files or folders to ignore** (not track). This helps keep the repository clean by avoiding unnecessary or sensitive files.

**What Should You Add to .gitignore? ✅**

* node\_modules/ → Installed dependencies (can be reinstalled).
* .parcel-cache/ → Parcel's build cache (not needed in Git).
* .env → Environment variables (API keys, secrets).
* dist/ or build/ → Compiled output files.
* \*.log → Log files.

**23: What is the difference between package.json and package-lock.json?**

**package.json** is the main configuration file for a Node.js project. It defines the project's metadata, dependencies, scripts, and versioning. It allows developers to specify which packages the project needs, but it often uses version ranges (e.g., ^1.0.0), meaning different environments might install slightly different versions of dependencies.

**package-lock.json**, on the other hand, is automatically generated when you install packages. It records the exact versions of each dependency and sub-dependency installed, ensuring that all team members or deployments use the same versions. This prevents unexpected bugs caused by different package updates.

In short, package.json defines what your project needs, while package-lock.json locks the exact versions to maintain consistency. Both files should be committed to Git for a stable and predictable development environment. 🚀

**24: Why should I not modify package-lock.json?**

You should not manually modify package-lock.json because it is automatically generated by **npm** to ensure consistency in package versions across different environments.

If you edit it manually, you might:

1. **Break Dependency Resolution** – npm may not recognize the correct versions when installing packages.
2. **Cause Inconsistencies** – Other developers or environments might install different versions, leading to unexpected bugs.
3. **Lose Integrity Hashes** – The file contains integrity checks for security; modifying it can break these validations.

Instead of editing it, always use commands like npm install or npm update to modify dependencies properly. 🚀

**25: What is node\_modules?**

**node\_modules** is a folder created when you run npm install. It stores all the installed dependencies required for your project.

**Why is node\_modules important?**

1. **Contains all installed packages** – Any dependencies listed in package.json get downloaded here.
2. **Includes transitive dependencies** – It also installs packages that your dependencies rely on.
3. **Helps run the project** – Without this folder, your JavaScript project won’t work properly.

**Should node\_modules be added to Git?**

❌ **No!** It is too large and can be regenerated anytime using npm install, so it's usually added to .gitignore. 🚀

**26: What is the dist folder?**

The **dist (distribution)** folder is the output directory where **bundled, minified, and optimized** files are stored after building a project. It contains the final version of your code, ready for deployment.

**Why is the dist folder important?**

1. **Optimized for production** – The files inside are compressed and performance-optimized.
2. **Contains compiled code** – It may include bundled JavaScript, CSS, and assets.
3. **Ready for deployment** – This is the folder you upload to a web server or use in production.

**Should the dist folder be added to Git?**

❌ **No!** Since it can be regenerated using a build tool (npm run build), it’s usually added to .gitignore. 🚀

**26: What is browserslist?**

**browserslist** is a configuration used in **front-end tools** like **Babel, Autoprefixer, and Parcel** to specify which browsers your project should support.

# Episode 03 - Laying the Foundation

**26: What is JSX?**

**JSX (JavaScript XML)** is a syntax extension for JavaScript that allows you to write **HTML-like code inside JavaScript**. It is commonly used in **React** to define UI components.

**Why is JSX useful?**

1. **Easier to write UI** – Looks like HTML but works inside JavaScript.
2. **More readable** – Makes component structures clear and intuitive.
3. **Compiles to JavaScript** – Browsers don’t understand JSX, so Babel converts it to standard JavaScript (React.createElement).

**27: Role of the type attribute in the <script> tag? What options can I use?**

The **type attribute** in the <script> tag specifies the **type of script** being used. It tells the browser how to interpret the script inside the tag.

**1: type="text/javascript"** (Default):  Modern browsers assume JavaScript by default, so it's optional.

**2: type="module"** (ES6 Modules)

* Enables ES6 module support, allowing import and export.
* Runs the script in strict mode by default.

**3**:**type="application/json"**

* Defines a JSON script that JavaScript can fetch and parse.

**4 . type="text/plain"**

* **Used when you don't want the browser to execute the script.**

**28: {TitleComponent} vs {<TitleComponent/>} vs {<TitleComponent></TitleComponent>} in JSX.**

* Ans: {TitleComponent} → Refers to the component function itself (not rendered).
* {<TitleComponent/>} → Calls & renders the component (self-closing).
* {<TitleComponent></TitleComponent>} → Same as above but used for components with children.

🚀 **Use {<Component/>} unless you need to pass children!**

**Episode 04 - Talk is Cheap, show me the code**

**29: Is JSX mandatory for React?**

**No, JSX is not mandatory for React,** but it makes writing UI components easier and more readable. React can work without JSX by using **React.createElement()** directly, but JSX is preferred because it simplifies syntax.

**30**.**Is ES6 mandatory for React?**

**No, ES6 is not mandatory for React**, but it is highly recommended because modern React code heavily relies on ES6 features. React can be written in ES5, but it becomes verbose and harder to maintain.

**Key ES6 Features Used in React:**

✅ **Arrow Functions (()=>{})** – Used for functional components.  
✅ **const and let** – For declaring variables.  
✅ **Destructuring** – Makes props and state handling easier.  
✅ **Spread and Rest Operators (... )** – Helps in props and state management.  
✅ **Modules (import/export)** – Used for organizing components.

While React **can** work without ES6, using it makes the code cleaner, more efficient, and modern

**31. Q: How can I write comments in JSX?**

Ans: {/\* single line JSX comment \*/}

{/\* Multi line

JSX comment \*/}

**32.What is <React.Fragment></React.Fragment> and <></>?**

In React, **<React.Fragment>** and **<>...</>** are used to wrap multiple elements **without adding extra HTML elements** to the DOM.

**1<React.Fragment></React.Fragment>**

✅ A built-in React component that **does not** create an extra DOM node.  
✅ Useful when returning multiple elements inside a component.

(Rendered HTML (No extra div!):

**2 Short Syntax: <></> (Shorthand for React.Fragment)**

✅ Works exactly like <React.Fragment>, but is **shorter and cleaner**.  
✅ Cannot accept key or other attributes.

**33: What is Reconciliation in React?**

**Reconciliation** is the process React uses to efficiently update the **DOM** by comparing the new Virtual DOM with the previous one. It ensures that only the necessary changes are made, improving performance.

**How Reconciliation Works:**

1. React **creates a Virtual DOM** – A lightweight copy of the actual DOM.
2. When state or props change, React **creates a new Virtual DOM**.
3. React **compares (diffs) the new Virtual DOM with the previous one**.
4. React updates **only the changed elements** in the actual DOM instead of re-rendering everything.

**Key Concepts in Reconciliation:**

✅ **Virtual DOM** – A memory-efficient representation of the UI.  
✅ **Diffing Algorithm** – React compares old and new Virtual DOMs to find changes.  
✅ **Efficient Updates** – Only modified parts of the DOM get updated, making React fast.

**34: What is React Fiber?**

**React Fiber** is the **new reconciliation algorithm** introduced in React 16. It improves React’s rendering performance by making updates faster and smoother, especially for complex UIs.

**Why was React Fiber introduced?**

Before Fiber, React used a **stack-based reconciliation** process, which was **synchronous** and could cause UI blocking. Fiber introduced an **asynchronous, incremental rendering approach**, making React more efficient.

**Key Features of React Fiber:**

✅ **Asynchronous Rendering** – Allows React to **pause and resume work**, preventing UI freezes.  
✅ **Prioritization of Updates** – Important updates (like user interactions) are processed **before** less important ones.  
✅ **Concurrency** – React can **split rendering work into chunks** and process them efficiently.  
✅ **Better Animation & Gesture Handling** – Makes animations and transitions smoother.

**How React Fiber Works:**

1. **Splits rendering into small units** (Fiber nodes).
2. **Processes high-priority updates first**, delaying less important updates.
3. **Can pause and resume rendering**, ensuring a smooth user experience.

**35: Why do we need keys in React?**

**Keys** are special attributes in React that help identify which items in a list have changed, been added, or removed. They improve **performance** by allowing React to efficiently update and re-render lists.

**Why are keys important?**

✅ **Optimized Reconciliation** – Helps React identify which items changed.  
✅ **Prevents Unnecessary Re-renders** – Avoids re-creating elements unnecessarily.  
✅ **Better Performance** – React updates only the modified items, not the whole list.

✅ Using a **unique key** (like id from a database) is better than index.

**What Happens Without Keys?**

* React **re-renders all list items**, even unchanged ones.
* **Incorrect animations or UI bugs** may occur.

**Best Practices for Keys:**

✔ Use **unique IDs** from data (not index).  
✔ Avoid using **random values** as keys (they change on every render).  
✔ Ensure **keys are stable and unique** within the list.

**Keys make React lists efficient and smooth! 🚀**

**36: What are props in React?**

**Props (short for "properties")** are used to pass data from a **parent component** to a **child component** in React. Props make components **reusable and dynamic** by allowing them to receive different values.

**Ways to Use Props in React**

**1. Passing Props to a Component**

You can pass props like HTML attributes when calling a component.

const Greeting = (props) => {

return <h1>Hello, {props.name}!</h1>;

};

const App = () => {

return <Greeting name="Alice" />;

};

🔹 Here, "Alice" is passed as a name prop to the Greeting component.

**2. Destructuring Props (Cleaner Syntax)**

Instead of using props.name, you can **destructure** props:

const Greeting = ({ name }) => <h1>Hello, {name}!</h1>;

✔ Makes code more readable and avoids props. repetition.

**3. Default Props (Setting Default Values)**

If a prop is not passed, you can define a **default value**.

const Greeting = ({ name = "Guest" }) => <h1>Hello, {name}!</h1>;

✔ If no name prop is passed, it defaults to "Guest".

**4. Passing Multiple Props**

const UserCard = ({ name, age }) => (

<p>{name} is {age} years old.</p>

);

const App = () => <UserCard name="Bob" age={25} />;

✔ Pass multiple props inside {}.

**5. Passing Objects as Props**

You can pass an entire object as a prop.

const user = { name: "Charlie", age: 30 };

const UserCard = ({ user }) => (

<p>{user.name} is {user.age} years old.</p>

);

const App = () => <UserCard user={user} />;

✔ Helps when passing multiple related values.

**6. Passing Functions as Props (Callback Props)**

Used when a child needs to **communicate with the parent**.

const Button = ({ handleClick }) => (

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

);

const App = () => {

const showMessage = () => alert("Button Clicked!");

return <Button handleClick={showMessage} />;

};

✔ Allows child components to trigger actions in the parent.

**Key Takeaways:**

✔ **Props are read-only** – Cannot be modified inside the child.  
✔ **Props help make components reusable**.  
✔ **Use destructuring** for cleaner code.  
✔ **Can pass strings, numbers, objects, arrays, or functions**.

🚀 **Props help build dynamic and reusable React components!**

**37: What is Config Driven UI?**

**Config Driven UI** is an approach where the UI is dynamically generated based on a **configuration object** rather than hardcoded components. This makes the UI **flexible, scalable, and reusable** without modifying the actual code.

**38: Difference Between Virtual DOM and Real DOM?**

The Virtual DOM (VDOM) and Real DOM are two different ways React handles UI updates. The Virtual DOM makes React faster by reducing unnecessary updates to the Real DOM.

**1️⃣ What is the Real DOM?**

DOM (Document Object Model) represents the UI as a tree structure. When updates happen:  
🔴 The entire UI may be re-rendered.  
🔴 Frequent updates cause performance issues.  
🔴 Directly manipulating the DOM is slow.

Example:

document.getElementById("btn").innerText = "Clicked!";

👉 Directly updating the Real DOM is expensive.

**2️⃣ What is the Virtual DOM?**

The Virtual DOM is a lightweight copy of the Real DOM stored in memory. React updates the Virtual DOM first, compares it with the previous version (diffing), and then applies only the necessary changes to the Real DOM (reconciliation).

🔵 Updates are faster and optimized.  
🔵 React minimizes direct Real DOM manipulations.  
🔵 Efficient batch updates improve performance.

Example: Virtual DOM in Action

const App = () => {

const [count, setCount] = React.useState(0);

return (

<div>

<h1>Count: {count}</h1> {/\* Only this updates, not the entire UI \*/}

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

};

👉 In Real DOM, updating the count would re-render the whole UI.  
👉 In Virtual DOM, React only updates the <h1> tag efficiently.

Why is Virtual DOM Better?

✔ Faster updates – React avoids unnecessary re-renders.  
✔ Better performance – UI feels smooth and responsive.  
✔ Efficient rendering – React only updates what has changed.

🚀 Virtual DOM makes React one of the fastest front-end libraries!

**38.React(or JavaScript in general), there are three main types of exports:**

1. **Named Export (export {} & import {})**
   * You can export multiple things from a file.
   * You must use the exact name when importing.
   * Example:
   * export const hello = () => console.log("Hello");
   * export const goodbye = () => console.log("Goodbye");
   * import { hello, goodbye } from "./file";
2. **Default Export (export default & import without {})**
   * Only one default export per file.
   * You can import it with any name.
   * Example:
   * export default function greet() {
   * console.log("Hello");}
   * import greet from "./file"; // Can be any name
3. **\* as Export (export \* as & import \* as )**
   * Exports everything as a single object.
   * You access items with dot (.) notation.
   * Example:
   * export const hello = () => console.log("Hello");
   * export const goodbye = () => console.log("Goodbye");
   * import \* as messages from "./file";
   * messages.hello();
   * messages.goodbye();

**39: What are React Hooks?**

**React Hooks (Simple Explanation)**

React Hooks are functions that let you **use state and other React features** in functional components **without writing a class**.

**Common Hooks:**

1. **useState** – Manages state in a functional component
2. **useEffect** – Runs side effects (e.g., fetching data, updating the DOM).
3. **useContext** – Accesses global state without prop drilling.

**40.Why do we need the useState Hook?**

In React, **functional components** used to be **stateless**. Before Hooks, if we wanted a component to have state, we had to use a **class component**.

The useState Hook allows us to **add state to functional components**, making them more powerful and easier to manage.

**Why useState is needed?**  
✅ Allows functional components to have **state**  
✅ Triggers **re-renders** when state changes  
✅ Makes the component **interactive**

**41 what is microservices?**

**Ans: A Microservice is a small, independent service that performs a specific function in an application. In a Microservices Architecture, an application is broken into multiple such services that communicate via APIs.**

For example, in a food delivery app, User Management, Orders, Payments, and Notifications can be separate microservices. This makes development, scaling, and maintenance easier since each service works independently.

**42: What is Monolithic Architecture?**

A: Monolithic Architecture is a traditional approach where the entire application is built as a single unit with all components (UI, business logic, and database) tightly integrated.

Example: A food delivery app in a monolith setup would have user management, orders, payments, and notifications all in one codebase and deployed together.

**43: What are the key differences between Monolithic and Microservices Architecture?**

| **Feature** | **Monolithic Architecture** | **Microservices Architecture** |
| --- | --- | --- |
| **Structure** | Single codebase & unit | Multiple independent services |
| **Scalability** | Hard to scale specific parts | Easily scalable per service |
| **Deployment** | Entire app deployed together | Each service deployed separately |
| **Fault Tolerance** | One failure can crash the app | Failure in one service doesn’t affect others |
| **Technology** | Uses a single tech stack | Can use different technologies for each service |
| **Speed of Development** | Faster for small apps | Better for large, complex apps |

**44: Why do we need the useEffect Hook?**

**A:** The useEffect hook is used in React functional components to handle **side effects**, such as:  
1️⃣ Fetching data from an API.  
2️⃣ Updating the DOM (e.g., changing the title).  
3️⃣ Managing event listeners and subscriptions.  
4️⃣ Handling timers and intervals.  
5️⃣ Cleaning up resources when a component unmounts.

👉 Without useEffect, side effects would run on every render, causing performance issues.  
👉 It replaces lifecycle methods like componentDidMount and componentWillUnmount in class components.

**45: What is Optional Chaining (?.) in JavaScript?**

**A:** **Optional Chaining (?.)** is a feature in JavaScript that **prevents errors** when accessing deeply nested properties in objects **that might be null or undefined**.

🔹 If profile exists, it returns name.  
🔹 If profile is undefined, it **stops execution** and returns undefined instead of throwing an error.

**46: What is Shimmer UI?**

**A:** **Shimmer UI** is a **loading placeholder effect** used in web applications to improve user experience while fetching data. Instead of showing a blank screen or a spinner, a **gray animated skeleton** (shimmer effect) is displayed to indicate that content is loading.

**47: What is the difference between JS expression and JS statement? and how will use in react**

An **expression** is a piece of code that **evaluates to a value**. It can be used inside variables, function calls, or JSX in React.

A **statement** is an instruction that **performs an action** but does not necessarily return a value. Statements include conditionals, loops, and function declarations.

48. **Q: What is Conditional Rendering in React?**

**Conditional Rendering** in React refers to the process of dynamically displaying or hiding components based on certain conditions. Instead of rendering everything at once, React allows us to control what gets displayed based on logic.

For example, a website might show a **login button** if a user is not logged in and a **logout button** if the user is logged in.

49. **Q: What is CORS?**

**CORS (Cross-Origin Resource Sharing)** is a security feature in web browsers that controls how resources (like APIs) can be accessed from a different domain than the one that served the webpage. It prevents unauthorized cross-origin requests for security reasons.

**Why is CORS Needed?**

By default, browsers follow the **Same-Origin Policy (SOP)**, which blocks requests from different origins to prevent security risks like **cross-site request forgery (CSRF)**. CORS allows safe exceptions by enabling servers to specify which origins can access their resources.

**How CORS Works?**

When a frontend app (e.g., example.com) requests data from a backend API (e.g., api.example.com), the server must send a **CORS header** like

**50: What is async and await in JavaScript?**

**async and await** are used in JavaScript to handle asynchronous operations in a cleaner and more readable way compared to Promises and callbacks.

* **async**: Used to declare a function that always returns a Promise.
* **await**: Pauses the execution inside an async function until the Promise resolves.

51. **Q: What is the use of const json = await data.json(); in getRestaurants()?**

When calling an API in JavaScript, we receive a **Response object**. The line:

**Converts the API response into a usable JSON format.**

**Step-by-Step Explanation:**

1️⃣ data holds the **raw response** from the API.  
2️⃣ .json() is a method that **parses** the response into JavaScript objects.  
3️⃣ await ensures we **wait** for the conversion to complete before moving forward.

| **52. What are various ways to add images into our react project? Explain simple answer with code examples.** |
| --- |

|  |  |
| --- | --- |
| **Import method** | Jab image **src folder** ke andar ho |

|  |  |
| --- | --- |
| **Public folder method** | Jab image **public folder** me ho |

|  |  |
| --- | --- |
| **External URL** | Jab image **online ya API se aaye** |

|  |  |
| --- | --- |
| **CSS Background** | Jab image **background me lagani ho** |

|  |  |
| --- | --- |
| **Dynamic Image** | Jab image **state ya props se change karni ho** |

**53.What would happen if we do console.log(useState())?**

A: If we do console.log(useState()), we get an array [undefined, function] where first item in an array is state is undefined and the second item in an array is setState function is bound dispatchSetState.

**54**. **Q: How will useEffect behave if we don't add a dependency array?**

Ans Syntax of useEffect is:

useEffect(() => {}, []);

Case 1 : When the dependency array is not included in the arguments of useEffect() hook, the callback function will be executed every time the component is rendered and re-rendered.

Case 2 : When the dependency array is empty in the arguments of useEffect() hook, the callback function will be executed only one time during the initial render of the component.

Case 3 : When the dependency array contains a condition, the callback function will be executed one time during the initial render of the component and also re-render if there is a change in the condition.

**55: What is SPA?**

A: Single Page Application (SPA) is a web application that dynamically updates the webpage with data from web server without reloading/refreshing the entire page. All the HTML, CSS, JS are retrieved in the initial load and other data/resources can be loaded dynamically whenever required. An SPA is sometimes referred to as a single-page interface (SPI).

**56**. **Client-Side Routing vs Server-Side Routing**

**1️ Server-Side Routing (SSR):**  
In **server-side routing**, every time a user navigates to a new page, a request is sent to the server. The server processes this request, fetches the required data, generates a new HTML page, and sends it back to the browser. This approach ensures that each page is fully loaded from the server, which improves SEO but results in slower navigation due to full-page reloads. It is commonly used in traditional websites built with PHP, Django, or ASP.NET.

**2️ Client-Side Routing (CSR):**  
In **client-side routing**, navigation happens within the browser using JavaScript without requesting a new page from the server. Instead, JavaScript dynamically updates the content of the page while keeping the URL in sync. This approach makes navigation much faster and provides a smoother user experience. However, it requires additional work for SEO optimization since search engines may struggle to index JavaScript-rendered content. CSR is widely used in modern JavaScript frameworks like React, Angular, and Vue.

**Key Differences:**

* SSR is **slower** because it reloads the full page, while CSR is **faster** as it only updates necessary content.
* SSR is **better for SEO**, whereas CSR requires **extra efforts** like server-side rendering (e.g., Next.js) to improve SEO.
* CSR provides a **better user experience** with smooth transitions, whereas SSR may cause flickering due to full-page reloads.

56. **What is the order of life cycle method calls in Class Based Components?**

A: Following is the order of lifecycle methods calls in Class Based Components:

1. constructor()
2. render ()
3. componentDidMount()
4. componentDidUpdate()
5. componentWillUnmount()

For more reference [React-Lifecycle-methods-Diagram](https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)

**57. Why do we use componentDidMount?**

**Why Do We Use componentDidMount()?**

componentDidMount() is a lifecycle method in **class-based components** in React. It runs **only once**, after the component has been rendered (mounted) to the DOM.

**Uses of componentDidMount()**

1. **Fetching Data from APIs** – It is commonly used to make API calls when the component loads.
2. **Adding Event Listeners** – Used to attach event listeners like window.addEventListener().
3. **Setting up Subscriptions** – Useful for setting up WebSockets, timers, or intervals.
4. **Manipulating the DOM** – Can be used to access and modify the DOM after the component is mounted.

**58: Why do we use componentWillUnmount? Show with example.**

**Why Do We Use componentWillUnmount()?**

componentWillUnmount() is a lifecycle method in **class-based components** in React. It is called **just before a component is removed (unmounted) from the DOM**.

**Uses of componentWillUnmount()**

1. **Cleaning up event listeners** – Removing event listeners to prevent memory leaks.
2. **Clearing timers or intervals** – Stopping setTimeout or setInterval functions.
3. **Unsubscribing from API calls or WebSockets** – Preventing unwanted network requests.
4. **Removing side effects** – Any side effect that should not persist after the component is gone.

**59. Why do we use super(props) in constructor?**

**Why Do We Use super(props) in Constructor?**

In **class-based components** in React, we use super(props) inside the constructor to correctly initialize the **this keyword** and access props inside the class.

**1️⃣ Key Reasons for Using super(props)**

1. **To Call the Parent Class Constructor (React.Component)**
   * In JavaScript, when a class extends another class, we must call super() before using this.
   * super(props) calls the constructor of React.Component, ensuring the component is initialized properly.
2. **To Allow Access to this.props Inside the Constructor**
   * Without super(props), this.props will be undefined inside the constructor.

**60. Why can't we have the callback function of useEffect async?**

❌ **Don't use async directly in useEffect** because it returns a Promise.  
✅ **Use an inner async function inside useEffect** or an **IIFE**.  
✅ **For cleanup, use the return function correctly**.

**Redux steps**

**🔁 What is Redux?**

Redux is a **state management library** for React apps. It's especially useful when many components need to share or update the same state.

**🔧 Why Redux?**

To avoid **prop drilling** (passing data through multiple nested components), Redux provides a **centralized store** so that **any component** can access or modify state easily.

**🧱 Core Concepts in Redux**

| **Term** | **Meaning** |
| --- | --- |
| **Action** | An object that describes what happened (can also carry data). |
| **Store** | A container that holds the whole app's state and connects everything. |
| **Reducer** | A function that updates the state based on the action. |
| **Slice** | A collection of reducer logic and initial state for a specific feature. |
| **State** | The actual data your app works with. |

**🔄 Redux Flow (How It Works)**

1. **UI triggers an event** (e.g., button click)
2. A **function dispatches an action**
3. The **action reaches the reducer via the store**
4. The **reducer updates the state**
5. The **UI re-renders** with the updated state

Button click -> handleFunc() -> dispatch(action) -> reducer -> state updated -> UI re-renders

**🛠️ Redux Setup in React App (Step-by-Step)**

**✅ Step 1: Create a Redux Store**

* Create a folder called redux
* Inside it, make a file store.js

// redux/store.js

import { configureStore } from '@reduxjs/toolkit';

import counterReducer from '../features/counterSlice';

const store = configureStore({

reducer: {

counter: counterReducer

},

});

export default store;

**✅ Step 2: Wrap Your App with <Provider>**

* In index.js, wrap <App /> with <Provider> from react-redux and pass the store

// index.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import { Provider } from 'react-redux';

import store from './redux/store';

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>,

document.getElementById('root')

);

**✅ Step 3: Create a Slice**

* Inside a features folder, create a file like counterSlice.js

// features/counterSlice.js

import { createSlice } from '@reduxjs/toolkit';

const counterSlice = createSlice({

name: 'counter',

initialState: {

value: 0

},

reducers: {

increment: (state) => {

state.value += 1;

},

decrement: (state) => {

state.value -= 1;

},

incrementByAmount: (state, action) => {

state.value += action.payload;

}

}

});

export const { increment, decrement, incrementByAmount } = counterSlice.actions;

export default counterSlice.reducer;

**✅ Step 4: Use Redux in Your Components**

// CounterComponent.js

import React from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { increment, decrement, incrementByAmount } from './features/counterSlice';

const Counter = () => {

const count = useSelector((state) => state.counter.value);

const dispatch = useDispatch();

return (

<div>

<h2>{count}</h2>

<button onClick={() => dispatch(increment())}>+</button>

<button onClick={() => dispatch(decrement())}>-</button>

<button onClick={() => dispatch(incrementByAmount(5))}>+5</button>

</div>

);

};

export default Counter;

**⚒️ Key Hooks:**

* useDispatch() – to send actions
* useSelector() – to access state from store
* **🔚 Summary**

| **Step** | **Description** |
| --- | --- |
| 1 | Create a store using configureStore() |
| 2 | Wrap <App /> in <Provider> and pass store |
| 3 | Create slice using createSlice() |
| 4 | Add reducers inside the slice |
| 5 | Use useSelector() and useDispatch() in components |

**Q: Advantages of using Redux Toolkit over Redux**

**Definition**: Redux Toolkit (RTK) is a modern approach to Redux that simplifies state management by reducing boilerplate code.

* **Simplifies Setup**: Reduces boilerplate with built-in best practices.
* **Less Code**: No need for manual action types, creators, or reducers.
* **Immutable Updates**: Uses immer for easy state management.
* **Integrated DevTools**: Pre-configured Redux DevTools support.
* **Modern Approach**: Promotes slices for modular state management.
* **Thunk Support**: Built-in createAsyncThunk for handling async logic.

**Q: Explain Dispatcher**

**Definition**: A dispatcher is a function in Redux that sends actions to the store to update the state.

* **Manages Actions**: Sends actions to update state in Redux.
* **Dispatch Function**: store.dispatch(action) triggers state changes.
* **Works with Reducers**: Passes actions to reducers for processing.
* **Ensures Predictability**: Centralized action flow for a structured state update.

**Q: Explain Reducer**

**Definition**: A reducer is a pure function in Redux that determines state changes based on actions.

* **Pure Function**: Takes current state + action → returns new state.
* **Handles Actions**: Uses switch case to update state.
* **Ensures Immutability**: Returns a new state instead of modifying existing.
* **Manages Specific State**: Can be combined with others using combineReducers.

**Q: Explain Slice**

**Definition**: A slice in Redux Toolkit is a collection of reducers, actions, and initial state for a specific feature.

* **Encapsulates Redux Logic**: Groups state, reducers, and actions together.
* **Uses createSlice**: Automatically generates actions and reducers.
* **Modular Approach**: Makes state management more organized.
* **Supports Mutability**: Uses immer to allow direct state modifications.

**Q: Explain Selector**

**Definition**: A selector is a function that retrieves specific data from the Redux store.

* **Extracts Data**: Fetches specific parts of the state.
* **Improves Performance**: Uses reselect for memoization.
* **Encapsulates Logic**: Reduces component dependency on state structure.

**Q: Explain createSlice and its Configuration**

**Definition**: createSlice is a function in Redux Toolkit that simplifies state management by auto-generating reducers and actions.

* **name**: Identifier for the slice.
* **initialState**: Defines the starting state.
* **reducers**: Contains functions to update state.
* **extraReducers**: Handles external actions.
* **Returns**: A reducer and auto-generated action creators.