


REACT JS INTERVIEW QUESTION AND ANSWERS

[Document subtitle]

INDEX

	 Question	
I.	What is react	3
II.	What are the major features of react	3
III.	How to create components in react	4
IV.	What is the difference between state and props	6
V.	What is reconciliation	7
VI.	What are the recommended ways for static types checking	8
VII.	Why react useclassname over class attribute	11
VIII.	Do hooks replace render and higher order components	11
IX.	How do you access the redux store of react components	11
X.	How do you access the redux store outside of react components in short	12
XI.	What is the use effect hook,and when it is used	13
XII.	How can you create a react application	13
XIII.	What is jsx	14
XIV.	What is the difference between function and class components in react	14
XV.	What is react memo function	15
XVI.	How events are different in react	16
XVII.	Here's concise summary of how events work differently in react	18
XVIII.	How to enable production mode in react	19
XIX.	Why you can't update props in react	19
XX.	What are the pointer events supported in react	19
XXI.	Dose the lazy function support name exports	20

1. what is react ?

- React is a JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications. It enables developers to create reusable UI components, manage the state of those components efficiently, and build complex user interfaces with a more declarative approach.

Key features of React include:

1. **Component-Based Architecture:** React lets you break down the UI into small, reusable components, each managing its own state and logic
2. **Virtual DOM:** React uses a virtual DOM to efficiently update and render components by comparing it with the real DOM and applying only the necessary changes.
3. **Declarative Syntax:** React's declarative approach makes it easier to describe what the UI should look like based on the state, simplifying the process of building and managing interactive UIs.
4. **JSX (JavaScript XML):** React uses JSX, a syntax extension that allows you to write HTML elements and components within JavaScript code, making it easier to visualize the UI structure
5. **Unidirectional Data Flow:** Data in React flows in one direction, from parent to child components, making it easier to understand how data changes affect the UI.

2. what are the major features of react ?

- React offers several major features that make it a powerful and popular library for building user interfaces:

1. **Component-Based Architecture:** React structures the UI into small, reusable components. Each component encapsulates its own logic, state, and rendering, which helps in building complex UIs from simple building blocks.
2. **Virtual DOM:** React maintains a virtual representation of the DOM to optimize updates. When the state of a component changes, React updates the virtual DOM first, compares it with the real DOM, and only applies the changes necessary. This improves performance and efficiency.
3. **Declarative Syntax:** React's declarative approach allows developers to describe what the UI should look like based on the state. This makes the code more predictable and easier to debug compared to imperative approaches.
4. **JSX (JavaScript XML):** JSX is a syntax extension that combines JavaScript and HTML. It allows developers to write HTML-like code within JavaScript files, which makes it easier to visualize and construct the UI.
5. **Unidirectional Data Flow:** React enforces a unidirectional data flow, meaning data flows from parent components to child components. This makes it easier to understand how data changes affect the application, reducing bugs and improving maintainability.
6. **State Management:** React provides mechanisms to manage component state, allowing components to maintain their own state and re-render when the state changes. For more complex state management, React can be used with libraries like Redux or Context API.
7. **Lifecycle Methods:** React components have lifecycle methods that allow developers to hook into different stages of a component's life (e.g., mounting, updating, unmounting). This provides control over how components behave at different points.
8. **Hooks:** Introduced in React 16.8, hooks allow functional components to manage state and side effects without needing class components. Common hooks include `useState`, `useEffect`, and `useContext`, which simplify component logic and state management.
9. **Context API:** The Context API allows you to manage and share global state across components without having to pass props down manually through every level of the component tree. It's useful for themes, authentication, and other global settings.
10. **React Router:** Although not a core feature, React Router is commonly used with React to handle client-side routing and navigation, enabling the creation of single-page applications with

3. how to create components in react

- Creating components in React involves defining reusable pieces of UI that can be used throughout your application. There are two main types of components in React: **function components** and **class components**. Here's a guide on how to create each type:

1. Function Components

Function components are simpler and are the recommended way to create components in modern React, especially with the introduction of hooks.

Basic Example:

Jsx

```
import React from 'react';

// Function component
const MyComponent = () => {
  return (
    <div>
      <h1>Hello, React!</h1>
      <p>This is a function component.</p>
    </div>
  );
};

export default MyComponent;
```

With Props:

jsx

```
import React from 'react';

// Function component with props
const Greeting = ({ name }) => {
  return <h1>Hello, {name}!</h1>;
};

export default Greeting;
```

Using Hooks:

jsx

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

// Function component with state
const Counter = () => {
  const [count, setCount] = useState(0);

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

export default Counter;
```

2. Class Components

Class components are more complex but still used in older codebases. They allow you to manage state and lifecycle methods.

Basic Example:

jsx

```
import React, { Component } from 'react';

// Class component
class MyClassComponent extends Component {
  render() {
    return (
      <div>
        <h1>Hello, React!</h1>
        <p>This is a class component.</p>
      </div>
    );
  }
}

export default MyClassComponent;
```

With Props:

jsx

```
import React, { Component } from 'react';

// Class component with props
class Greeting extends Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}

export default Greeting;
```

With State and Lifecycle Methods:

jsx

```
import React, { Component } from 'react';

// Class component with state and lifecycle methods
class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  componentDidMount() {
    console.log('Component mounted');
  }

  componentDidUpdate(prevProps, prevState) {
    console.log('Component updated');
  }

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

  render() {
    return (
      <div>
        <p>Current count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}

export default Counter;
```

Key Points

- **Function Components:** Use function components for simpler, stateless components, and leverage hooks for managing state and side effects.
- **Class Components:** Use class components for more complex components that require lifecycle methods or when working with legacy codebases.

Both types of components can be used interchangeably, but function components with hooks are more modern and offer a more concise syntax.

4. what is the difference between state and props?

- In React, **state** and **props** are two fundamental concepts used to manage and pass data within components. Here's a comparison to highlight their differences:

Props

1. **Definition:** Props (short for "properties") are read-only attributes passed from a parent component to a child component. They are used to pass data and event handlers down the component tree.
2. **Usage:** Props are used to configure a component and control its behavior from outside. For example, you might pass data or callback functions to a component via props.
3. **Mutability:** Props are immutable; a component cannot change its own props. They are set by the parent component and cannot be modified by the child component.
4. **Setting Props:** Props are set by the parent component when rendering the child component.
5. For example:

jsx

```
<ChildComponent name="Alice" age={30} />
```

6. **Accessing Props:** Props are accessed within a component via `this.props` in class components or directly from the function arguments in function components:

jsx

```
// Function component
const ChildComponent = ({ name, age }) => {
  return <p>Name: {name}, Age: {age}</p>;
};
jsx
Copy code
// Class component
class ChildComponent extends React.Component {
  render() {
    return <p>Name: {this.props.name}, Age: {this.props.age}</p>;
  }
}
```

State

1. **Definition:** State is a local data storage that is managed within a component. It can be used to store and manage data that affects the component’s behavior or rendering.
2. **Usage:** State is used to handle dynamic data that can change over time, such as user inputs, form submissions, or interactive elements. It is essential for handling components that need to respond to user interactions or other events.
3. **Mutability:** State is mutable; components can change their own state using methods like `setState` in class components or the `useState` hook in function components.
4. **Setting State:** State is typically initialized in a component’s constructor (for class components) or directly within the function component (for function components). For example:

jsx

// Function component
const [count, setCount] = useState(0);
jsx
Copy code
// Class component
class Counter extends React.Component {
 constructor(props) {
 super(props);
 this.state = { count: 0 };
 }
 ...
}

5. **Updating State:** State can be updated using:
- **setState** method in class components:

```
this.setState({ count: this.state.count + 1 });
```

- **State updater function** from the `useState` hook in function components:

jsx

setCount(count + 1);

6. **Accessing State:**
- **Function components:** Access directly from the state variable.

jsx

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

- **Class components:** Access using `this.state`.

jsx

this.state.count

Summary

- **Props** are for passing data and functions from parent to child components, and they are immutable within the child component.
- **State** is used for managing internal data and is mutable; it is local to the component and can be changed by the component itself.

Understanding the difference between props and state helps in designing React applications where data flows effectively and components manage their own data appropriately.

5.what is reconciliation?

- In React, **reconciliation** is the process of updating the user interface in response to changes in the application’s state or props. It involves comparing the current state of the user interface (UI) with a new state and then applying the necessary updates to the DOM efficiently.

Key Aspects of Reconciliation

- 1.Virtual DOM:** React uses a virtual DOM, which is a lightweight copy of the actual DOM. When a component’s state or props change, React first updates the virtual DOM. It then compares this updated virtual DOM with the previous version using a process called **diffing**.
- 2.Diffing Algorithm:** The diffing algorithm is a core part of React's reconciliation process. It determines the minimal set of changes required to update the actual DOM based on differences between the new virtual DOM and the previous virtual DOM. This algorithm optimizes performance by reducing the number of direct DOM manipulations.

- 3.Updating the DOM:** After identifying the differences, React updates the real DOM with only the necessary changes. This process is more efficient than re-rendering the entire DOM, which helps improve performance and user experience.
- 4.Component Keys:** In lists of elements, React uses keys to identify which items have changed, been added, or been removed. Keys help React to maintain element identity across re-renders and enhance the efficiency of the reconciliation process.
- 5.Lifecycle Methods and Hooks:** React provides lifecycle methods in class components (like `componentDidUpdate`, `componentDidMount`, etc.) and hooks in function components (like `useEffect`) that allow developers to perform actions during different phases of the reconciliation process. This helps in managing side effects and other logic related to component updates.

How Reconciliation Works

- 1. **Render Phase:** When a component’s state or props change, React starts the render phase. It generates a new virtual DOM tree based on the updated state or props.
- 2. **Diffing:** React compares the new virtual DOM tree with the previous virtual DOM tree. It identifies which parts of the virtual DOM have changed.
- 3. **Update Phase:** React calculates the minimal set of changes needed to update the real DOM to match the new virtual DOM. It then performs these updates in a batch to optimize performance.
- 4. **Commit Phase:** React applies the calculated changes to the actual DOM. It also invokes any lifecycle methods or hooks related to the update.

Example

Consider a simple example where a component displays a list of items. If an item is added, removed, or updated:

- 1. **Initial Render:** The component renders a list of items to the virtual DOM.
- 2. **State Update:** When an item is added or removed, the component’s state changes.
- 3. **Reconciliation:**
 - React creates a new virtual DOM tree with the updated list.
 - React compares this new virtual DOM with the previous one.
 - React identifies which items were added or removed.
 - React updates the real DOM to reflect these changes efficiently.

Summary

Reconciliation is a critical part of React’s performance optimization strategy. By using a virtual DOM and efficient diffing algorithms, React minimizes the number of direct DOM manipulations, ensuring that UI updates are fast and efficient. This approach contributes to a smoother and more responsive user experience.

6.what are the recommended ways for static type checking ?

- Static type checking is a technique to ensure the correctness of code by analyzing the types of variables and expressions at compile time. In JavaScript and TypeScript, several tools and practices are recommended for static type checking:

1. TypeScript

TypeScript is a superset of JavaScript that adds static type checking. It is one of the most popular and robust tools for static type checking in JavaScript projects.

- **Features:**
 - **Static Type Checking:** Enforces type constraints during development.
 - **Type Inference:** Infers types based on code context to reduce explicit type declarations.
 - **Rich Type System:** Supports interfaces, generics, enums, and more.
 - **Tooling Support:** Excellent support in popular editors (e.g., VSCode) and build tools.
- **Example:**

typescript

```
// TypeScript example
function greet(name: string): string {
  return `Hello, ${name}!`;
}
```



```
    }  
    const greeting = greet("Alice");
```

2. Flow

Flow is a static type checker developed by Facebook for JavaScript. It offers a way to add type annotations to JavaScript code and provides type checking.

- **Features:**
 - **Gradual Typing:** Allows you to opt-in types in JavaScript code.
 - **Type Inference:** Infers types in the absence of explicit type annotations.
 - **Integration:** Works with various editors and build systems.
- **Example:**

javascript
<pre>// Flow example // @flow function greet(name: string): string { return `Hello, \${name}!`; } const greeting = greet("Alice");</pre>

3. PropTypes (React)

PropTypes is a library for runtime type checking of React component props. While it's not a static type checker, it helps ensure that the props passed to React components conform to expected types.

- **Features:**
 - **Runtime Validation:** Checks types at runtime rather than compile-time.
 - **Descriptive Errors:** Provides useful error messages for invalid prop types.
- **Example:**

javascript
<pre>// PropTypes example in React import PropTypes from 'prop-types'; function MyComponent({ name, age }) { return <div>{`Name: \${name}, Age: \${age}`}</div>; } MyComponent.propTypes = { name: PropTypes.string.isRequired, age: PropTypes.number.isRequired };</pre>

4. JSDoc with Type Annotations

JSDoc can be used to provide type annotations in JavaScript files. Tools like **Google Closure Compiler** or **TypeScript** can use these annotations for static type checking.

- **Features:**
 - **Type Annotations:** Adds type information using JSDoc comments.
 - **Tool Integration:** Some IDEs and tools can infer types from JSDoc annotations.

6. **Example:**

javascript

```
// JSDoc example
/**
 * @param {string} name
 * @param {number} age
 * @returns {string}
 */
function greet(name, age) {

    return `Hello, ${name}. You are ${age} years old.`;
}

const greeting = greet("Alice", 30);
```

5. Type Checking with ESLint

ESLint is a linter tool for JavaScript that, with the help of plugins, can also be configured for type checking.

- **Features:**
 - **TypeScript Plugin:** Integrates with TypeScript for linting and type checking.
 - **Flow Plugin:** Provides type checking for Flow annotations.

7. **Example:**

javascript

```
// Using ESLint with TypeScript
// .eslintrc.js
module.exports = {
  parser: '@typescript-eslint/parser',
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/recommended',
  ],
  plugins: ['@typescript-eslint'],
};
```

Summary

- **TypeScript:** The most comprehensive solution for static type checking in JavaScript projects.
- **Flow:** An alternative to TypeScript for adding type annotations to JavaScript.
- **PropTypes:** Useful for runtime prop validation in React components.
- **JSDoc:** Allows type annotations in JavaScript files and can be used with certain tools for type checking.
- **ESLint:** Configurable for type checking with the help of TypeScript or Flow plugins.

Choosing the right tool depends on your project requirements, team preferences, and existing development environment. TypeScript is widely recommended for its strong type system and extensive community support.

7. Why react uses className over class attribute ?

- React uses `className` instead of `class` for defining CSS classes on elements because `class` is a reserved keyword in JavaScript.

In JavaScript, `class` is used to define classes (as in the object-oriented programming concept), so using it as an attribute name in JSX could lead to conflicts or confusion. By using `className`, React avoids this issue and makes it clear that this attribute is specifically for defining CSS classes, not JavaScript classes.

Here's a quick comparison:

- **In HTML:** You use `class` to specify CSS classes, like this:

Html
<code><div class="my-class"></div></code>

- **In JSX (used by React):** You use `className` to achieve the same effect:

jsx
<code><div className="my-class"></div></code>

This approach aligns with the need to avoid conflicts with the reserved `class` keyword while maintaining the ability to define and apply CSS classes in a way that is familiar to those coming from traditional HTML and CSS backgrounds.

8. Do hooks replace render props and higher order components ?

- **hooks** provide an alternative to **render props** and **higher-order components (HOCs)**, offering a simpler and more flexible way to manage state and side effects in functional components. While they don't strictly "replace" these patterns, hooks often make code easier to read and maintain compared to the more complex and sometimes cumbersome render props and HOCs patterns.

To summarize:

- **Hooks** simplify state and side effect management within functional components, making code more readable and reusable.
- **Render Props** and **HOCs** are still useful and applicable in certain scenarios, but hooks are generally preferred for their simplicity and ease of use.

9. How do you access the Redux store outside of React components?

- To access the Redux store outside of React components, you can import the store directly from the file where it's created and then interact with it using its methods. Here's a brief overview:

1. **Export the Store:** Ensure the store is exported from its configuration file.

jsx
<pre>// store.js import { createStore } from 'redux'; import rootReducer from './reducers'; const store = createStore(rootReducer); export default store;</pre>

2. **Import and Use the Store:** Import the store where needed and use methods like `getState()` to access the current state or `dispatch()` to dispatch actions.

jsx

```
// outsideComponent.js
import store from './store';

// Get current state
const state = store.getState();
console.log(state);

// Dispatch an action
store.dispatch({ type: 'ACTION_TYPE', payload: 'data' });
```

This method allows you to interact with the Redux store from outside React components, such as in utility functions or middleware.

10. How do you access the Redux store outside of React components in short ?

- We can access the Redux store outside of React components by importing the store instance directly and using its methods. For example:

1. **Export the store** from its configuration file:

jsx

```
// store.js
import { createStore } from 'redux';
import rootReducer from './reducers';

const store = createStore(rootReducer);
export default store;
```

2. **Import and use the store** where needed:

jsx

```
// outsideComponent.js
import store from './store';

// Access state
const state = store.getState();

// Dispatch an action
store.dispatch({ type: 'ACTION_TYPE', payload: 'data' });
```

This allows you to interact with the store directly for purposes such as logging or utility functions.

11. what is the use effect hook , and when it is used ?

- The `useEffect` hook in React is used to perform side effects in functional components. It allows you to run code that interacts with the DOM, fetch data, set up subscriptions, or manually trigger side effects after the component renders.

Key Points:

- **Usage:** `useEffect` takes two arguments: a function containing the side effect code, and an optional dependency array. The effect function runs after the component renders and can clean up after itself if necessary.

jsx

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

const Example = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Fetch data or perform side effects
    fetch('/api/data')
      .then(response => response.json())
      .then(data => setData(data));

    // Optional cleanup function
    return () => {
      // Cleanup code here
    };
  }, []); // Empty dependency array means it runs once on mount

  return <div>{data ? data.message : 'Loading...'}</div>;
};
```

- **When to Use:**
 - To fetch data from an API.
 - To subscribe to external services (like WebSocket connections).
 - To perform manual DOM manipulations.
 - To set up and clean up timers or intervals.

Summary: `useEffect` manages side effects in functional components, ensuring code runs after rendering and can clean up resources when the component unmounts or dependencies change.

12. how can you create a react application ?

- To create a React application quickly, use the `create-react-app` CLI tool. Here’s a short step-by-step guide:

1. **Install `create-react-app`** (if not already installed):

bash

```
npx create-react-app my-app
```

2 Navigate to the project directory:

bash
cd my-app

3 Start the development server:

bash
npm start

This command sets up a new React project with a basic structure, including build scripts, a development server, and a sample app. You can start building your React application right away.

13. What is jsx ?

- JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code directly within JavaScript. It simplifies creating and embedding React elements and components by mixing HTML with JavaScript logic.

Example:

jsx
const element = <h1>Hello, world!</h1>;

JSX is transpiled into plain JavaScript, making it easy to define the UI structure in a declarative way.

14. What is the difference between functional and class components in React?

- **Functional Components:** Simple functions that accept props and return React elements. They use hooks (useState, useEffect, etc.) for state and side effects. Preferred for their simplicity and conciseness.

jsx
function MyComponent(props) { const [count, setCount] = React.useState(0); React.useEffect(() => { /* side effect */ }, []); return <div>{count}</div>; }

Class Components: ES6 classes that extend `React.Component` and include a `render` method. They manage state with `this.state` and lifecycle methods (`componentDidMount`, `componentDidUpdate`, etc.).

jsx
class MyComponent extends React.Component { constructor(props) { super(props); this.state = { count: 0 }; } componentDidMount() { /* side effect */ } render() { return <div>{this.state.count}</div>; } }

Summary: Functional components are simpler and use hooks for state and side effects, while class components use state and lifecycle methods with a more complex syntax.

15. What is react.memo functions ?

- `React.memo` is a higher-order component that optimizes functional components by memoizing them. It prevents unnecessary re-renders by only re-rendering the component when its props change.

Usage:

Jsx
<pre>const MyComponent = React.memo((props) => { return <div>{props.value}</div>; });</pre>

Key Points:

- **Purpose:** Improves performance by avoiding re-renders when props haven't changed.
- **Comparison:** By default, performs a shallow comparison of props.

Summary: `React.memo` helps optimize functional components by memoizing them, thus reducing unnecessary re-renders when the component's props remain the same.

16. HOW EVENTS ARE DIFFERENT IN REACT ?

➤ In React, events are handled differently compared to traditional HTML/JavaScript event handling. Here are the main differences:

A. Synthetic Events:

- React uses a synthetic event system that wraps the browser's native events. This is known as SyntheticEvent. It is a cross-browser wrapper around the native event to ensure consistent behavior across different browsers.
- Synthetic events have the same interface as native events but are normalized to be consistent.

B. Event Delegation:

- React uses event delegation for performance optimization. Instead of attaching event handlers to every individual DOM element, React attaches a single event listener to the root of the document and delegates events to the appropriate components. This approach improves performance and reduces the number of event handlers.

C. Event Naming:

- In React, event names are written in camelCase. For example, the HTML `onclick` becomes `onClick` in React, `onchange` becomes `onChange`, and so on.

D. Handling Events:

- Event handlers in React are written as functions, and you pass these functions as props to the React elements. You often use arrow functions or bind methods to ensure the correct `this` context.

JSX-

```
function MyComponent() {  
  
  const handleClick = (event) => {  
  
    console.log('Button clicked!', event);  
  
  };  
  
  return <button onClick={handleClick}>Click me</button>;  
  
}
```

E. Event Handling with this:

When using class components you need to bind the event handler methods to the instance of the class. This is usually done in the constructor or by using public class fields syntax

JSX-

```
class MyComponent extends React.Component {  
  
  constructor(props) {  
  
    super(props);  
  
    this.handleClick = this.handleClick.bind(this);  
  
  }  
  
  handleClick(event) {
```



```
    console.log('Button clicked!');
  }
  render() {
    return <button onClick={this.handleClick}>Click me</button>;
  }
}
```

: Alternatively, using public class fields syntax

```
class MyComponent extends React.Component {
  handleClick = (event) => {
    console.log('Button clicked!');
  };

  render() {
    return <button onClick={this.handleClick}>Click me</button>;
  }
}
```

F. Event Pooling:

- Synthetic events are pooled to improve performance. This means that the event object is reused and may be nullified after the event callback has been invoked. If you need to access event properties asynchronously, you should call `event.persist()` to opt out of pooling.

```
jsx-

function MyComponent() {
  const handleClick = (event) => {
    event.persist(); // Prevents the event from being
    pooled
    setTimeout(() => {
      console.log(event.target); // Safe to access event
      properties
    }, 1000);
  };

  return <button onClick={handleClick}>Click me</button>;
}
```

G. Default Behavior:

- In React, you can prevent default behavior using `event.preventDefault()` just as you would in traditional JavaScript, but the handling is often done in a more declarative way.

❖ Here's a concise summary of how events work differently in React:

1. **Synthetic Events:** React uses a cross-browser wrapper called `SyntheticEvent` to ensure consistent behavior across different browsers.
2. **Event Delegation:** React attaches a single event listener to the root document and delegates events to the appropriate components for better performance.
3. **Event Naming:** Event names are written in camelCase in React (e.g., `onClick`, `onChange`).
4. **Event Handling:** Pass event handler functions as props. Use arrow functions or bind methods to handle `this` context correctly.
5. **Binding in Class Components:** Bind event handler methods in the constructor or use public class fields to avoid binding issues.
6. **Event Pooling:** React reuses event objects for performance. Call `event.persist()` if you need to access event properties asynchronously.
7. **Prevent Default Behavior:** Use `event.preventDefault()` to stop default actions, just as in standard JavaScript.

17. HOW TO ENABLE PRODUCTION MODE IN REACT ?

➤ To enable production mode in React, you need to ensure that your React application is built using the production configuration. Here’s a concise guide:

1. Use **NODE_ENV** Variable:
 - Set the `NODE_ENV` environment variable to `production`. This is typically handled automatically by build tools like Webpack or Create React App (CRA) during the build process.
2. **Build the App:**
 - For Create React App, run:

bash

npm run build

or

bash

yarn build

- This command sets `NODE_ENV` to `production` and optimizes the build for production.
3. **Deploy the Build:**
 - Deploy the contents of the `build` directory to your web server or hosting service.

18.WHY YOU CAN’T UPDATE PROPS IN REACT ?

➤ In React, you can't directly update props because they are immutable. Props are intended to be set by the parent component and remain constant throughout the lifecycle of the child component. If you need to change the data, you should:

1. **Update State:** Use state in the parent component to manage data and pass it down as props.
2. **Lift State Up:** Manage shared state in a common parent component and pass it down as props to children.

This ensures a unidirectional data flow and maintains a predictable component hierarchy.

19. WHAT ARE THE POINTER EVENTS SUPPORTED IN REACTS ?

➤ In React, pointer events can be handled using the standard React event handlers, which are based on the Pointer Events API. Here are the pointer events supported in React:

1. **onPointerDown:** Fired when a pointer (mouse, touch, stylus) is pressed down on an element.
2. **onPointerUp:** Fired when a pointer is released from an element.
3. **onPointerMove:** Fired when a pointer is moved over an element.
4. **onPointerEnter:** Fired when a pointer enters an element's boundary.
5. **onPointerLeave:** Fired when a pointer leaves an element's boundary.
6. **onPointerOver:** Fired when a pointer moves over an element or one of its children.
7. **onPointerOut:** Fired when a pointer moves out of an element or one of its children.
8. **onPointerCancel:** Fired when a pointer event is canceled, often due to interruption or an unexpected change.

These events work similarly to other React events but are designed to handle input from various types of pointers.

20.DOES THE LAZY FUNCTON SUPPORT NAMED EXPORTS ?

- No, React's `React.lazy` function does not directly support named exports. `React.lazy` is used to dynamically import a default export from a module, so it expects the module to have a default export.

If you need to use named exports, you can work around this limitation by creating a wrapper module that re-exports the named exports as a default export. Here’s how you can do it:

1. **Create a Wrapper Module:**
 - Create a new file that imports the named export and then re-exports it as a default export.

```
jsx

// MyComponentWrapper.js
import { MyComponent } from './MyComponent';
export default MyComponent;
```

2. **Use `React.lazy` with the Wrapper:**
 - Now you can use `React.lazy` to import the default export from the wrapper module.

```
Jsx-

// App.js
import React, { Suspense } from 'react';
const LazyComponent = React.lazy(() =>
import('./MyComponentWrapper'));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}

export default App;
```

This approach allows you to leverage `React.lazy` even when your original module uses named exports.

OR

No, `React.lazy` only supports default exports. To use named exports, create a wrapper module that re-exports the named export as a default export, and then use `React.lazy` with that wrapper.

