Here are **very long, clear, exam-oriented answers** for **all 10 questions**, written exactly in a **Web Technologies / React** exam style (LPU / MCA friendly). You can safely write these for **10–15 marks**.

---

1. What is the Virtual DOM?

The **Virtual DOM (VDOM)** is a lightweight, in-memory representation of the real DOM used by React. It is a JavaScript object that mirrors the structure of the actual DOM tree.

Direct manipulation of the real DOM is slow because it causes reflow and repaint operations in the browser. React avoids this problem by first updating the Virtual DOM instead of the real DOM.

When a change occurs in a component:

1. React creates a new Virtual DOM tree.
2. It compares the new tree with the previous one using a process called **diffing**.
3. React calculates the minimum number of changes required.
4. Only those changes are applied to the real DOM.

This approach improves performance and makes UI updates faster and more efficient.

---

2. What is the difference between Pure React and React?

React:

React is a JavaScript library used to build user interfaces using components. By default, React re-renders a component whenever its parent re-renders or its state/props change.

Pure React (PureComponent):

Pure React refers to using **React.PureComponent**, which automatically implements shouldComponentUpdate() with a shallow comparison of props and state.

Key Differences:

React Component	PureComponent
Always re-renders on update	Re-renders only if props/state change
No automatic comparison	Shallow comparison
Slightly slower	Better performance
Manual optimization needed	Built-in optimization

Conclusion:

Pure React improves performance by avoiding unnecessary re-renders, but it should be used carefully when dealing with complex data structures.

---

3. What is JSX and how does it work?

JSX (JavaScript XML) is a syntax extension for JavaScript used in React to describe UI elements. It looks similar to HTML but is actually written inside JavaScript.

Example:

```
const element = <h1>Hello React</h1>;
```

How JSX works:

- JSX is **not understood by browsers directly**
- It is converted (transpiled) into JavaScript using tools like Babel
- JSX is transformed into React.createElement() calls

Transpiled version:

```
React.createElement("h1", null, "Hello React");
```

Benefits of JSX:

- Makes code more readable
- Combines UI and logic
- Reduces coding errors
- Improves developer productivity

---

4. What are the differences between functional and class components?

Functional Components:

- Simple JavaScript functions
- Use Hooks for state and lifecycle
- No this keyword
- Less code and more readable

Example:

```
function Hello() {
  return <h1>Hello</h1>;
}
```

Class Components:

- ES6 classes
- Use this keyword
- Use lifecycle methods
- More boilerplate code

Example:

```
class Hello extends React.Component {
  render() {
    return <h1>Hello</h1>;
  }
}
```

Differences Table:

Functional	**Class**
Hooks	Lifecycle methods
No this	Uses this
Faster	Slightly slower
Preferred	Older approach

5. What is the Virtual DOM? How does React use it to render the UI?

The **Virtual DOM** is a JavaScript-based representation of the UI.

Rendering process in React:

1. Initial render creates a Virtual DOM tree.
2. State or props change triggers a new Virtual DOM.
3. React compares old and new Virtual DOM trees.
4. Differences are identified (diffing).
5. Only changed elements are updated in the real DOM.

This process is called **reconciliation**.

Advantages:

- High performance
- Efficient updates
- Smooth UI rendering
- Better user experience

---

6. What are the differences between controlled and uncontrolled components?

Controlled Components:

- Form data is controlled by React state
- Uses value and onChange

- Better validation and control

Example:

```
<input value={name} onChange={e => setName(e.target.value)} />
```

Uncontrolled Components:

- Form data handled by DOM itself
- Uses ref
- Less React control

Example:

```
<input ref={inputRef} />
```

Differences:

Controlled	**Uncontrolled**
React state	DOM state
Predictable	Less predictable
Validation easy	Validation harder
Recommended	Limited use

7. Explain about types of side effects in React component

Side effects are operations that affect something outside the component's render process.

Common side effects:

1. Fetching data from APIs
2. Updating the DOM manually
3. Setting timers (setTimeout, setInterval)
4. Subscribing to events
5. Logging

In functional components, side effects are handled using **useEffect()**.

Example:

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

Types:

- **Mounting side effects** – run once
- **Updating side effects** – run on state/props change

- **Cleanup side effects** – cleanup resources

---

8. What are the rendering characteristics? Explain.

Rendering characteristics define **how and when React components re-render**.

Key characteristics:

1. **Re-render on state change**
2. **Re-render on props change**
3. **Parent re-render affects child**
4. **Virtual DOM diffing**
5. **Batching of updates**

Important points:

- React renders efficiently
- Only changed parts update
- Avoids full page reloads

These characteristics make React fast and scalable.

---

9. What are Web APIs? Explain in detail.

Web APIs are interfaces provided by browsers that allow JavaScript to interact with browser features and external systems.

Common Web APIs:

- DOM API
- Fetch API
- Geolocation API
- Storage API (localStorage, sessionStorage)
- Timer API (setTimeout)
- Canvas API

Example:

```
fetch("https://api.example.com/data")
  .then(response => response.json())
  .then(data => console.log(data));
```

Role in React:

- Used for data fetching

- Handling browser storage
- Accessing device features

Web APIs extend JavaScript capabilities beyond core language features.

---

10. What is a framework and how is it used in React?

A **framework** is a complete structure that provides rules, tools, and libraries to build applications.

React and framework:

- React itself is a **library**, not a framework
- When combined with tools like:
    - React Router
    - Redux
    - Webpack
    - Babel
      it behaves like a framework

Usage in React:

- Component-based structure
- Routing
- State management
- Build optimization

Conclusion:

React becomes framework-like when used with supporting tools, allowing scalable and maintainable application development.
