React JS Frame work

1. **What is React.js?**
    - React.js is a JavaScript library for building user interfaces. It allows developers to create reusable UI components.
2. **Explain the Virtual DOM in React and how it improves performance.**
    - The Virtual DOM is a lightweight representation of the actual DOM in memory. React uses it to perform updates efficiently by comparing the current Virtual DOM with the previous one, minimizing actual DOM manipulations.
    - Example:

```
jsxCopy code
// Initial render
const element = <h1>Hello, World!</h1>;
ReactDOM.render(element, document.getElementById('root'));

// Subsequent update
const updatedElement = <h1>Hello, Universe!</h1>;
ReactDOM.render(updatedElement, document.getElementById('root'));
```

3. **What are the key features of React.js?**
    - Components and JSX
    - Virtual DOM
    - One-way data binding
    - React Router (for handling routing)
    - Flux/Redux (for state management)
4. **What is JSX? Provide an example.**
    - JSX stands for JavaScript XML and allows you to write HTML-like code directly within JavaScript.
    - Example:

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

5. **Explain React components and their types.**
    - Components are the building blocks of React applications. They can be either functional or class-based.
    - Example of a functional component:

```
jsxCopy code
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

6. **What are class components in React?**
    - Class components are ES6 classes that extend `React.Component`. They have a `render()` method to return the component's UI.

```
jsxCopy code
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

7. **What's the difference between functional and class components?**
    - Class components can have state and lifecycle methods, while functional components can't.
    - Class components use `this.props`, while functional components pass props as an argument.
8. **Explain the concept of props in React.**
    - Props (short for properties) are inputs to React components, allowing data to be passed from parent components to child components.

```
jsxCopy code
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Usage of the Greeting component with props
const element = <Greeting name="John" />;
```

9. **How do you handle events in React?**
    - In React, events are camel-cased and passed as props to components.

```
jsxCopy code
class Button extends React.Component {
  handleClick() {
    console.log('Button clicked!');
  }

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

10. **What is the state in React?**
    - State is a built-in feature of React components used for managing mutable data. When state changes, React re-renders the component.

```
jsxCopy code
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Increment
        </button>
      </div>
    );
```

```
  }
}
```

11. **Explain the concept of lifting state up in React.**
    - Lifting state up is a pattern in React where you move the state from a child component to its parent to share the state among multiple children.
12. **What is the significance of keys in React lists?**
    - Keys are used to help React identify which items have changed, been added, or removed in a list, enabling efficient rendering and updating of components.

```
jsxCopy code
const todos = ['Learn React', 'Build an app', 'Deploy to production'];

const TodoList = () => {
  return (
    <ul>
      {todos.map((todo, index) => (
        <li key={index}>{todo}</li>
      ))}
    </ul>
  );
};
```

13. **What is React Router, and how do you use it for navigation?**
    - React Router is a library that allows you to handle routing in a React application. It provides components like `BrowserRouter` and `Link` for navigation.

```
jsxCopy code
import { BrowserRouter, Route, Link } from 'react-router-dom';

const Home = () => <h1>Home</h1>;
const About = () => <h1>About</h1>;

const App = () => {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
```

```
      <Route path="/" exact component={Home} />
      <Route path="/about" component={About} />
    </BrowserRouter>
  );
};
```

14. **What are controlled components in React?**

    - Controlled components are React components where form data is controlled by React state. The state is used to manage the input values and keep them in sync with the component's UI.

```
jsxCopy code
class Form extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: '' };
  }

  handleChange(event) {
    this.setState({ value: event.target.value });
  }

  handleSubmit(event) {
    event.preventDefault();
    console.log('Submitted value:', this.state.value);
  }

  render() {
    return (
      <form onSubmit={(event) => this.handleSubmit(event)}>
        <input
          type="text"
          value={this.state.value}
          onChange={(event) => this.handleChange(event)}
        />
        <button type="submit">Submit</button>
      </form>
    );
  }
}
```

15. **What are Higher-Order Components (HOCs)?**

- HOCs are functions that take a component and return a new component with extended or modified functionality.

```
jsxCopy code
function withLogger(WrappedComponent) {
  return class extends React.Component {
    componentDidMount() {
      console.log('Component is mounted!');
    }

    render() {
      return <WrappedComponent {...this.props} />;
    }
  };
}

const EnhancedComponent = withLogger(MyComponent);
```

16. **Explain React hooks and list some commonly used ones.**

- React hooks are functions that allow functional components to use state and other React features without using class components. Commonly used hooks are `useState`, `useEffect`, `useContext`, `useReducer`, and `useCallback`.

```
jsxCopy code
import { useState, useEffect } from 'react';

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

  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]);

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

17. **What is the significance of `useEffect` hook in React?**

- The `useEffect` hook is used to perform side effects in functional components, such as data fetching, subscriptions, or DOM manipulations.

18. **Explain the Context API in React and why you would use it.**
    - The Context API is used for managing global state in React applications, allowing data to be shared without having to pass props manually through all components.
19. **How do you perform data fetching in React?**
    - Data fetching can be done using various methods, such as `fetch`, Axios, or using GraphQL clients like Apollo or Relay.

```
jsxCopy code
import { useEffect, useState } from 'react';

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

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then((response) => response.json())
      .then((data) => setData(data));
  }, []);

  return (
    <ul>
      {data.map((item) => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
};
```

20. **What is the purpose of the `shouldComponentUpdate` method?**
    - The `shouldComponentUpdate` method is a lifecycle hook that allows you to control whether a component should re-render or not. It can be used for performance optimization.
21. **What are React Fragments, and why are they useful?**
    - React Fragments, or `<>...</>`, are used to group multiple elements without adding an extra node to the DOM. They are useful when you don't want to add

an unnecessary parent element.

22. **How do you handle forms in React?**
    - Form handling in React can be achieved by using controlled components or using form libraries like Formik.
23. **What are portals in React, and why are they used?**
    - Portals allow you to render components outside the parent DOM hierarchy. They are useful for modals, tooltips, and other scenarios where you need to render content at a different DOM location.
24. **Explain Error Boundaries in React and their purpose.**
    - Error Boundaries are special React components used to catch JavaScript errors during rendering, in lifecycle methods, or in constructors of whole component tree.

```
jsxCopy code
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // Log the error or send it to an error reporting service
  }

  render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}
```

25. **What are the different lifecycle methods in React class components?**
    - `componentDidMount`, `componentDidUpdate`, `componentWillUnmount`, `getDerivedStateFromProps`, `shouldComponentUpdate`, etc.

26. **Explain React.memo() and its use case.**

- `React.memo()` is a higher-order component that memoizes the result of the component rendering. It prevents unnecessary re-renders by comparing the previous and current props.

```
jsxCopy code
const MyComponent = React.memo((props) => {
  /* component implementation */
});
```

27. **What is the significance of the `key` prop in dynamic lists?**

- The `key` prop is used by React to identify elements uniquely in a list and to improve performance during updates.

28. **How can you conditionally render components in React?**

- Conditional rendering can be done using ternary operators, `&&` short-circuiting, or by using `if` statements within the `render()` method.

```
jsxCopy code
const ConditionalComponent = ({ show }) => {
  return show ? <div>Show me</div> : null;
};
```

29. **What is the purpose of `React.StrictMode`?**

- `React.StrictMode` is a wrapper component that helps identify potential issues in the application during development by enabling additional checks and warnings.

```
jsxCopy code
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

30. **How do you handle default props in React?**

- You can define default props for a component using the `defaultProps` property.

```
jsxCopy code
class Greeting extends React.Component {
  // ...
}

Greeting.defaultProps = {
  name: 'Guest',
};
```

31. **What is the significance of the `dangerouslySetInnerHTML` attribute?**

- `dangerouslySetInnerHTML` is used to inject HTML from a string directly into a React component. It should be used with caution as it might lead to XSS attacks if not handled properly.

```
jsxCopy code
const htmlString = '<span style="color: red;">Dangerous HTML</span>';

const MyComponent = () => {
  return <div dangerouslySetInnerHTML={{ __html: htmlString }} />;
};
```

32. **Explain the difference between `useCallback` and `useMemo`.**

- `useCallback` memoizes a function and returns the memoized version, while `useMemo` memoizes a value and returns the memoized value.

33. **What is the significance of `React.Children` utilities?**

- The `React.Children` utilities allow you to work with the `props.children` object, even if it's a single child or an array of children.

```
jsxCopy code
import React from 'react';

const ChildrenExample = ({ children }) => {
  const count = React.Children.count(children);
  return <div>Number of children: {count}</div>;
```

```
};
```

34. **How do you use React without JSX?**

- You can use `React.createElement()` to create React elements without JSX.

```
jsxCopy code
const element = React.createElement('h1', null, 'Hello, World!');
```

35. **What are hooks rules in React?**

- Hooks should only be called at the top level of functional components or other custom hooks, not inside loops, conditions, or nested functions.

36. **Explain the concept of code splitting in React.**

- Code splitting is a technique to break the application code into smaller chunks and load them on demand. It improves the initial loading time and performance.

37. **How do you implement lazy loading in React?**

- Lazy loading can be achieved using React's `lazy` function along with dynamic `import()`.

```
jsxCopy code
import React, { lazy, Suspense } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'));

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

38. **What is the significance of `React.Fragment`?**

- `React.Fragment` is an alternative to using the shorthand `<>...</>` syntax for wrapping multiple elements. It does not create an additional DOM node.

39. **How do you prevent the component from rendering in React?**
    - Components can prevent rendering by returning `null` from the `render()` method.
40. **Explain the `displayName` property in React components.**
    - The `displayName` property is used to give a name to a component. It helps with debugging by providing more readable component names in React Developer Tools.

```
jsxCopy code
const MyComponent = () => {
  // component implementation
};

MyComponent.displayName = 'MyComponent';
```

41. **What is React DevTools, and how does it help developers?**
    - React DevTools is a browser extension used to inspect React component hierarchies, state, and props. It aids in debugging and optimizing React applications.
42. **How do you optimize performance in React applications?**
    - Performance optimization techniques include using code splitting, memoization, avoiding unnecessary renders, and minimizing the use of inline functions.
43. **How can you prevent the default behavior of an event in React?**
    - You can call `event.preventDefault()` to prevent the default behavior of an event.

```
jsxCopy code
const handleClick = (event) => {
  event.preventDefault();
  // Your custom logic here
};
```

44. **Explain the concept of portals in React.**

- Portals allow you to render components outside of their parent DOM hierarchy. They are useful for scenarios like modals and tooltips.

```
jsxCopy code
import ReactDOM from 'react-dom';

const Modal = ({ children }) => {
  return ReactDOM.createPortal(
    <div className="modal">
      {children}
    </div>,
    document.getElementById('modal-root')
  );
};
```

45. **What is the purpose of `dangerouslySetInnerHTML` in React?**

- `dangerouslySetInnerHTML` is used to insert HTML content from a string directly into a component. It should be used with caution, as it can lead to XSS vulnerabilities if not handled properly.

```
jsxCopy code
const htmlString = '<span style="color: red;">Dangerous HTML</span>';

const MyComponent = () => {
  return <div dangerouslySetInnerHTML={{ __html: htmlString }} />;
};
```

46. **What is React Server-Side Rendering (SSR), and why is it important?**

- SSR is a technique where the initial HTML content is generated on the server and sent to the client, enabling search engines to index the page properly and improving performance for the first load.

47. **How do you handle forms in React?**

- Form handling can be done using controlled components, where form values are stored in state, or by using form libraries like Formik or React Hook Form.

```
jsxCopy code
const MyForm = () => {
```

```
  const [formData, setFormData] = useState({
    username: '',
    password: '',
  });

  const handleChange = (event) => {
    setFormData({
      ...formData,
      [event.target.name]: event.target.value,
    });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    // Handle form submission with formData
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="username"
        value={formData.username}
        onChange={handleChange}
      />
      <input
        type="password"
        name="password"
        value={formData.password}
        onChange={handleChange}
      />
      <button type="submit">Submit</button>
    </form>
  );
};
```

48. **How do you handle routing in React applications?**

- React Router is a popular library for handling routing in React applications. It provides components like `BrowserRouter`, `Route`, and `Link`.

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

const Home = () => <h1>Home</h1>;
const About = () => <h1>About</h1>;

const 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>
  );
};
```

49. **How do you handle state management in React?**
    - React offers built-in state management using the `useState` hook. For more complex state management, you can use libraries like Redux or Mobx.
50. **Explain the concept of hooks in React.**
    - Hooks are functions that allow you to use state and other React features in functional components without writing a class.