**3.Controlled and UnControlled Components ?**

Uncontrolled Components

Uncontrolled Components are the components that are not controlled by the React state and are handled by the DOM (Document Object Model). So in order to access any value that has been entered we take the help of refs. For instance, if we want to add a file as an input, this cannot be controlled as this depends on the browser so this is an example of an uncontrolled input.

Example: We are creating a simple form that comprises an input field with a label name and a submit button with an onSubmit function that triggers when we submit the form. We are accessing the name we have filled using useRef.

import React, { useRef } from "react";

import "./App.css";

function App() {

const inputRef = useRef(null);

function handleSubmit() {

alert(`Name: ${inputRef.current.value}`);

}

return (

<div className="App">

<h1 className="geeks">GeeksForGeeks</h1>

<h3>Uncontrolled Component</h3>

<form onSubmit={handleSubmit}>

<label>Name :</label>

<input

type="text"

name="name"

ref={inputRef}

/>

<button type="submit">Submit</button>

</form>

</div>

);

}

export default App;

Controlled Components

In React, Controlled Components are those in which form’s data is handled by the component’s state. It takes its current value through props and makes changes through callbacks like onClick, onChange, etc. A parent component manages its own state and passes the new values as props to the controlled component. Example: We are creating a state name that stores the value we enter into the input field using the useState hook. We are creating an onChange function in the input field that stores the data we are entering in the input field to our state. There is another function handleSumit that gets triggered when we submit the form and it shows the name we have entered on the alert box.

import { useState } from "react";

import "./App.css";

function App() {

const [name, setName] = useState("");

function handleSubmit() {

alert(`Name: ${name}`);

}

return (

<div className="App">

<h1 className="geeks">GeeksForGeeks</h1>

<h3>Controlled Component</h3>

<form onSubmit={handleSubmit}>

<label>Name:</label>

<input

name="name"

value={name}

onChange={(e) =>

setName(e.target.value)

}

/>

<button type="submit">Submit</button>

</form>

</div>

);

}

export default App;

Controlled Component -

The component is under control of the component’s state.

These components are predictable as are controlled by the state of the component.

Internal state is not maintained.

It accepts the current value as props

Does not maintain its internal state.

Controlled by the parent component.

Have better control on the form data and values

Uncontrolled Component-

Components are under the control of DOM.

Are Uncontrolled because during the life cycle methods the data may loss

Internal state is maintained

We access the values using refs

Maintains its internal state.

Controlled by the DOM itself.

Has very limited control over form values and data

**4.What is the use Super(props) in React?**

So, the simple answer to this question is that this thing basically allows accessing this.props in a Constructor function. In fact, what the super() function does is, calls the constructor of the parent class.

**5.What are synthetic events in ReactJS ?**

In order to work as a cross-browser application, React has created a wrapper same as the native browser in order to avoid creating multiple implementations for multiple methods for multiple browsers, creating common names for all events across browsers.

Synthetic events in React are cross-browser wrappers around the browser’s original event. Different browsers may have different names for the events. They create a uniform interface that React uses to ensure that events execute consistently across browsers. React normalizes this information to provide a consistent API for handling events regardless of the browser.

Syntax:

e.preventDefault() // prevents all the default behavior by the browser.

e.stopPropagation() // prevents the call to the parent component whenever a child component gets called.

Some of the attributes are:

bubbles: Return true or false indicates event is a bubbling event or not.

cancelable: Return true or false indicates if the event can be canceled or not.

currentTarget: Indicates the element to which the handler is attached

defaultPrevented: Return true or false, indicates whether the event has been canceled by preventDefault().

eventPhase: Returns number, indicates the phase

isTrusted: Return true when the event is generated by the user and false when by the browser/script

type: Returns string, it indicates the type of the event

**6.What is the difference between useState and useRef ?**

useRef is primarily used to access and manipulate the DOM or store mutable values without triggering re-renders. It provides a mutable reference that persists across component renders. On the other hand, useState is used for managing component state, triggering re-renders when the state updates.

**7.Why do we use forward ref in react js?**

ForwardRef() is a utility function in react that let you expose a child components DOM to a parent component with a ref. Usually the parent component passes the props and data to the child component. But in some instances like when working with input or where components need to respond with user interactions, the parent components needs direct access to the child components DOM, to achieve this you can use forward refs.

**8.What is Reconciliation?**

Reconciliation in React refers to the process of updating the DOM (Document Object Model) to match the most recent virtual representation of a component's UI. In simpler terms, it's React's way of ensuring that the user interface reflects the current application state accurately.

**9.How does the lifecycle of a React component work?**

A React component goes through different phases in its lifecycle. These phases are:

* Mounting: This phase occurs when the component is first created and inserted into the DOM.
* Updating: This phase occurs when the component's state or props change.
* Unmounting: This phase occurs when the component is removed from the DOM.

Each phase has its own set of lifecycle methods that can be used to perform specific tasks. These methods are called automatically by React at the appropriate time.

**10.Props And State**

Props

Props are known as properties it can be used to pass data from one component to another. Props cannot be modified, read-only, and Immutable.

The Data is passed from one component to another.

Props can be used with state and functional components.

Props are read-only.

State

The state represents parts of an Application that can change. Each component can have its State. The state is Mutable and It is local to the component only.

The Data is passed within the component only.

It is Mutable ( can be modified).

The state can be used only with the state components/class component (Before 16.0).

The state is both read and write.

**11.What are the latest technology advancements in react ?**

Here are some of the latest technology advancements in React

Concurrent Mode:

Concurrent Mode is a new feature in React that allows developers to create components that can render and update asynchronously. This can improve the performance of applications by allowing React to work on multiple tasks simultaneously.

Suspense:

Suspense is a feature that allows developers to handle asynchronous data fetching and error handling in a more seamless way. This can make applications more responsive and user-friendly.

Server Components:

Server Components are a new way to build server-rendered React applications. They are designed to be more efficient and easier to use than traditional server-rendering methods.

Improved Accessibility:

React is constantly improving its accessibility features. This makes it easier for developers to create applications that can be used by everyone.

Enhanced Server-Side Rendering:

React is also improving its server-side rendering capabilities. This allows developers to render their applications on the server rather than the client, which can improve performance and SEO.

Improved Developer Tooling:

The React developer tooling is constantly being improved. This makes it easier for developers to debug and develop React applications.

**12.What is error handling in react js?**

Error boundaries are React components that **catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI** instead of the component tree that crashed. Only Class Component can be Error Boundary.

**13.What are react portals?**

React Portals are a powerful feature in React that allows you to render components outside the current React tree hierarchy. With portals, you can easily render elements into different parts of the DOM, such as modals, tooltips, or any other component that needs to break out of the component’s container.

**14.Functional Component and Class based Component ?**

Functional Components

A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX). There is no render method used in functional components.Functional components run from top to bottom and once the function is returned it can’t be kept alive.Also known as Stateless components as they simply accept data and display them in some form, they are mainly responsible for rendering UI.React lifecycle methods (for example, componentDidMount) cannot be used in functional components.Hooks can be easily used in functional components to make them Stateful.Constructors are not used.

Example:

const [name,SetName]= React.useState(' ')

Class Based Components

A class component requires you to extend from React. Component and create a render function that returns a React element.It must have the render() method returning JSX (which is syntactically similar to HTML).The class component is instantiated and different life cycle method is kept alive and is run and invoked depending on the phase of the class component.Also known as Stateful components because they implement logic and state.React lifecycle methods can be used inside class components (for example, componentDidMount).It requires different syntax inside a class component to implement hooks.Constructor is used as it needs to store state.

Example:

constructor(props) {

super(props);

this.state = {name: ' '}

}

**15.What is JSX?**

JSX provides you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

Why use JSX?

It is faster than regular JavaScript because it performs optimization while translating the code to JavaScript.

Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both. We will learn components in a further section.

It is type-safe, and most of the errors can be found at compilation time.

It makes easier to create templates.

**16.How will you optimise components in react ?**

Optimizing components in React involves several techniques to improve performance and efficiency. Here are some strategies you can employ:

Functional Components and React.memo: Use functional components whenever possible, as they are generally more lightweight. Additionally, wrap components with React.memo to memoize the result, preventing unnecessary re-renders if the props haven't changed.

import React from 'react';

const MyComponent = React.memo(function MyComponent({ prop }) {

// Component logic

});

PureComponent and shouldComponentUpdate: If you're using class components, extend React.PureComponent instead of React.Component. This provides a shallow prop and state comparison for shouldComponentUpdate, preventing re-renders if props and state haven't changed.

import React, { PureComponent } from 'react';

class MyComponent extends PureComponent {

// Component logic

}

Use Key Prop: When rendering lists in React, ensure each child component has a unique key prop. This helps React efficiently update the DOM when the list changes.

<ul>

{items.map(item => (

<li key={item.id}>{item.name}</li>

))}

</ul>

Avoid Inline Functions: Minimize inline function creation within render methods, especially when passing them down as props, as this can cause unnecessary re-renders. Instead, define functions outside the render method and bind them in the constructor or use arrow functions.

class MyComponent extends React.Component {

handleClick = () => {

// Handle click

};

render() {

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

}

}

Use React.lazy and Suspense: For code-splitting, use React.lazy to dynamically import components. This helps reduce the initial bundle size and improves performance. Wrap lazy-loaded components with Suspense to display fallback content while the component is loading.

javascript

Copy code

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

function App() {

return (

<Suspense fallback={<div>Loading...</div>}>

<LazyComponent />

</Suspense>

);

}

Memoize Expensive Computations: If a component relies on expensive computations or data fetching, memoize the results using techniques like useMemo or caching libraries like memoize-one. This prevents redundant calculations on re-renders.

import { useMemo } from 'react';

function MyComponent({ data }) {

const processedData = useMemo(() => expensiveProcess(data), [data]);

return <div>{processedData}</div>;

}

**17.Can you explain the concept of higher-order components?**

Higher-order components (HOCs) are a powerful and flexible pattern in React for enhancing and extending the functionality of components. The concept is derived from functional programming, where functions can take other functions as arguments or return functions.

In React, a higher-order component is a function that takes a component as its argument and returns a new component with enhanced capabilities. This pattern allows you to encapsulate common functionality and share it across multiple components without repeating code.

Here's a basic example to illustrate the concept:

import React from 'react';

// Higher-order component

const withLogger = (WrappedComponent) => {

return class extends React.Component {

componentDidMount() {

console.log('Component is mounted');

}

render() {

return <WrappedComponent {...this.props} />;

}

};

};

// Usage

class MyComponent extends React.Component {

render() {

return <div>My Component</div>;

}

}

const EnhancedComponent = withLogger(MyComponent);

export default EnhancedComponent;

In this example:

withLogger is a higher-order component that takes a component (WrappedComponent) as its argument.

Inside withLogger, a new component is defined, which logs a message when it's mounted and renders the WrappedComponent.

MyComponent is a regular component that renders a simple <div>.

EnhancedComponent is created by passing MyComponent to withLogger, resulting in a new component with logging functionality added.

Higher-order components offer several benefits:

Reusability: HOCs allow you to encapsulate common logic and behavior, making it reusable across different components.

Composability: You can compose multiple HOCs together to create more complex behavior, providing a flexible way to extend component functionality.

Separation of Concerns: HOCs promote separation of concerns by separating the core functionality of a component from additional concerns such as logging, authentication, data fetching, etc.

Enhanced Testing: Since HOCs are regular JavaScript functions, they can be easily tested in isolation, promoting better testability of your codebase.

However, it's essential to use HOCs judiciously and be mindful of potential pitfalls, such as prop drilling and nesting too many HOCs, which can make the codebase harder to understand. Additionally, with the advent of React hooks, many patterns that previously required HOCs can now be achieved more elegantly with custom hooks.

**18.How do React keys work in a list, and why are they important?**

In React, when you render a list of elements using the map() function or any similar method, you need to assign a unique key prop to each element in the list. React uses these keys to efficiently update the DOM when the list changes, ensuring that the correct components are added, removed, or reordered without re-rendering unnecessary components.

Here's how keys work in a list:

Uniqueness: Each key in a list must be unique among its siblings. React uses keys to differentiate between components and their corresponding DOM elements.

Stability: Keys should be stable, meaning they should not change between renders unless the identity of the component itself changes. This ensures that React can correctly identify which components have been added, removed, or updated.

Efficient Reconciliation: When updating a list, React compares the new list of keys with the previous list to determine the minimal set of changes required to update the DOM. Without keys, React would have to rely solely on the order of elements, leading to potentially inefficient DOM updates and rendering.

Performance: Using keys allows React to optimize the rendering process by minimizing DOM manipulations. React can efficiently detect when elements are added, removed, or reordered, resulting in better performance, especially for large lists.

Here's an example of rendering a list with keys:

const items = [

{ id: 1, text: 'Item 1' },

{ id: 2, text: 'Item 2' },

{ id: 3, text: 'Item 3' }

];

const ListComponent = () => (

<ul>

{items.map(item => (

<li key={item.id}>{item.text}</li>

))}

</ul>

);

In this example:

Each li element in the list is assigned a unique key based on the id of the corresponding item.

When the list changes (e.g., items are added, removed, or reordered), React uses the keys to efficiently update the DOM without re-rendering all list items.

In summary, keys are essential in React when rendering lists because they enable efficient reconciliation and help React identify and track individual components within a list, leading to better performance and a smoother user experience.

**19.What is Coupling ?**

Coupling refers to the degree of interdependence between different components or modules within a system. In software engineering, coupling describes how closely connected or dependent two or more modules are on each other. It is a measure of how much one module relies on the internal workings of another module.

There are different types of coupling:

Low Coupling: In low coupling, modules are independent of each other, meaning changes to one module have minimal impact on others. This promotes modularity and reusability, as modules can be modified, replaced, or extended without affecting the rest of the system.

High Coupling: High coupling occurs when modules are closely interconnected and dependent on each other's internal details. Changes to one module may require modifications to multiple other modules, increasing the risk of unintended side effects and making the system more difficult to maintain and evolve.

Tight Coupling: Tight coupling refers to a strong relationship between modules where they directly reference each other's internal implementation details. This makes it challenging to isolate and test individual modules, as changes in one module can have cascading effects on others.

Loose Coupling: Loose coupling describes a weaker relationship between modules, where communication occurs through well-defined interfaces or contracts. Modules interact with each other through defined APIs, reducing direct dependencies and promoting flexibility and maintainability.

**20. HOw to implement lazy loading in react routers with router dom library?**

Lazy loading in React Router with React Router DOM is a technique used to dynamically import components only when they are needed, thus improving the performance of your application by reducing the initial bundle size. React Router provides a React.lazy() function that allows you to lazily load components. Here's how you can implement lazy loading in React Router with React Router DOM:

const LoadingFallback = () => <div>Loading...</div>;

const App = () => {

return (

<BrowserRouter>

<Suspense fallback={<LoadingFallback />}>

<Switch>

<Route exact path="/" component={Home} />

<Route path="/about" component={About} />

<Route path="/lazy" component={LazyComponent} />

</Switch>

</Suspense>

</BrowserRouter>

);

};

export default App;

Make sure you handle code splitting properly with your build tool (e.g., webpack) so that the components are bundled into separate chunks for lazy loading to work effectively. Also, note that lazy loading is typically more effective in larger applications with many components or complex UIs. In smaller applications, the benefits may be negligible.

**21.Why we can use className instead of class in React js**

In React.js, you use className instead of class to define CSS classes for elements. This is because class is a reserved keyword in JavaScript, used for defining classes in object-oriented programming. React uses JSX, which is a syntax extension for JavaScript, to describe the structure of UI components. JSX closely resembles HTML, but it's ultimately transpiled into JavaScript.

Since class is reserved in JavaScript, using it directly in JSX would result in a syntax error. Therefore, React developers opted to use className instead to assign CSS classes to elements. This naming convention allows React to differentiate between regular JavaScript classes and CSS classes within JSX.

**22.Error Boundaries in ReactJS ?**

In React.js, an error boundary is a React component that catches JavaScript errors anywhere in its child component tree, logs those errors, and displays a fallback UI instead of crashing the entire React application. Error boundaries are a useful feature for building robust and resilient React applications because they help prevent the "blank screen of death" that occurs when an unhandled JavaScript error crashes the entire UI.

Here's how you can create an error boundary in React:

Create an Error Boundary Component:

Define a new React component that will act as your error boundary. This component should implement at least one of the error boundary lifecycle methods: static getDerivedStateFromError() or componentDidCatch().

import React, { Component } from 'react';

class ErrorBoundary extends Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

// Update state so the next render will show the fallback UI.

return { hasError: true };

}

componentDidCatch(error, errorInfo) {

// You can also log the error to an error reporting service

console.error('Error caught by error boundary:', error, errorInfo);

}

render() {

if (this.state.hasError) {

// Fallback UI when an error occurs

return <h2>Something went wrong.</h2>;

}

return this.props.children;

}

}

export default ErrorBoundary;

Wrap Components with the Error Boundary:

Wrap the components you want to be covered by the error boundary with the <ErrorBoundary> component you've just created.

jsx

Copy code

import ErrorBoundary from './ErrorBoundary';

import MyComponent from './MyComponent';

const App = () => {

return (

<div>

<h1>My React App</h1>

<ErrorBoundary>

<MyComponent />

</ErrorBoundary>

</div>

);

};

export default App;

Now, if any error occurs within the MyComponent or its child components, the ErrorBoundary component will catch the error and display the fallback UI defined within its render() method. This prevents the error from propagating up and crashing the entire application.

Remember to use error boundaries strategically and sparingly in your application. They should wrap only the components that might fail unpredictably, not the entire application. Also, note that error boundaries only catch errors in components below them in the component tree. They do not catch errors within themselves, in event handlers, or during server-side rendering.

**23.Why we use super in constructor in React ?**

In React, when you define a constructor for a class-based component, you typically call super(props) as the first statement in the constructor. This is necessary because React components extend React.Component, which is a class provided by the React library. Calling super(props) ensures that the constructor of the base class (React.Component) is called before initializing the state or performing any other operations in the constructor of your component.

Here's why super(props) is used in React component constructors:

Accessing this.props: When you call super(props), you're essentially calling the constructor of the base class (React.Component) and passing the props object to it. This allows your component to access the props via this.props within its methods.

Initializing State: If your component has its own state, you may want to initialize it in the constructor. By calling super(props) first, you ensure that this.props is properly initialized, so you can use it to set the initial state based on props, if needed.

Binding Event Handlers: If you're using class methods as event handlers, you might need to bind them to the correct instance of the component. You can do this in the constructor. However, with modern JavaScript and the use of arrow functions for class methods, this is less common.

**24. What is Props Drilling ?**

"props drilling" or "component drilling," refers to the process of passing down props from a parent component to deeply nested child components through multiple layers of intermediate components. It occurs when a component needs to pass data to its child components, but those child components are not direct descendants, requiring the data to be passed through intermediary components in the component tree.

Here's how prop drilling typically works:

Parent Component: The data originates from a parent component that holds the state or receives the data from an external source (e.g., an API call).

Passing Props: The parent component passes down the necessary data or functions as props to its immediate child component.

Intermediate Components: If the child component needs to pass the received data further down to its own child components, it must pass the props it received from its parent to its children. This process continues recursively through multiple layers of components until the data reaches the component where it's needed.

Child Component: Finally, the data reaches the desired child component where it's used for rendering or other purposes.

**25.What are WebSockets ?**

WebSocket are a communication protocol that enable bidirectional communication between application between application . They are a great choice when two-way communication is needed such as chat and multiplayer collaboration . Additionally , websocket are well-suited when you need to push fresh data from the server as soon as its available-like live sports score updated on a package delivery or perhaps realtime chart data . Because they are full-duplex ,information can flow in both directions simultaneously, making websockets an attractive option for high throughput scenarios like an online multipliew game as well.

**26.What are React Portals ?**

React Portals are powerful features in React that allows you to remember components outside the current React tree hierarchy with portals, you can easily render elements into different

**27.Explain Redux ?**

Redux is a predictable state container for JavaScript apps, primarily used with frameworks like React. It helps manage the state of your application in a predictable way, making it easier to understand, debug, and maintain your application's state.

Key Concepts:

Store: The store holds the global state of your application. It's a JavaScript object that represents the state tree.

Actions: Actions are payloads of information that send data from your application to the store. They are plain JavaScript objects and must have a type property indicating the type of action being performed.

Reducers: Reducers specify how the application's state changes in response to actions. They are pure functions that take the current state and an action as arguments and return a new state.

Dispatch: Dispatch is the method used to send actions to the Redux store. It is the only way to update the state.

Workflow:

When something happens in the application (like a user interaction), an action is dispatched.

The reducers then respond to these actions by computing the new state based on the previous state and the action dispatched.

Finally, the updated state is stored in the Redux store, and any components subscribed to that state are notified of the change and re-rendered accordingly.

Benefits of Redux:

Predictability: Redux follows strict principles, making it predictable and easy to understand how data flows through your application.

Centralized State: Having a single source of truth (the store) makes it easier to manage and debug your application's state.

DevTools: Redux comes with helpful development tools that allow you to trace actions, track state changes, and debug your application efficiently.

Scalability: Redux scales well with larger applications due to its organized structure and predictable data flow.

When to Use Redux:

Use Redux when your application has complex state management requirements, such as sharing state across multiple components or handling deeply nested state.

It's especially useful when your application's state needs to be accessed by many components at different levels of the component tree.

Pros of redux :

Predictable state management : Easier to reason about application state . Improved Testability,scalability.

Cons of Redux :

Boilerplate code requires setting up stores,reducers,action and dispatch. Complexity Learning curve for managing state with reducers and actions.

**28.How Data flow in Redux?**

In Redux, the data flow follows a unidirectional pattern, which means data flows in a single direction throughout the application. Understanding this data flow is crucial for effectively managing the state of your application. Here's how the data flows in Redux:

Action Dispatch:

The data flow starts when an action is dispatched. An action is a plain JavaScript object that describes the intention to change the state.

Each action must have a type property, which is a string that describes the type of action being performed. Additional properties can be included to carry any necessary data.

Reducers Receive Actions:

Actions are sent to reducers. Reducers are pure functions that take the current state and an action as arguments, and return a new state based on that action.

Reducers are responsible for specifying how the application's state changes in response to the actions they receive.

State Update:

The reducer calculates the new state based on the current state and the action it received. It doesn't mutate the existing state but returns a new state object.

It's important to note that Redux enforces immutability, meaning state updates are handled by creating new state objects rather than modifying existing ones. This ensures predictability and easier debugging.

Store Update:

Once the new state is calculated, it replaces the old state in the Redux store.

The Redux store is a single JavaScript object that holds the entire state tree of your application.

Component Re-rendering:

Any components that are subscribed to the parts of the state that have changed are notified of the update.

React components connected to the Redux store via the connect function or hooks like useSelector will re-render to reflect the updated state.

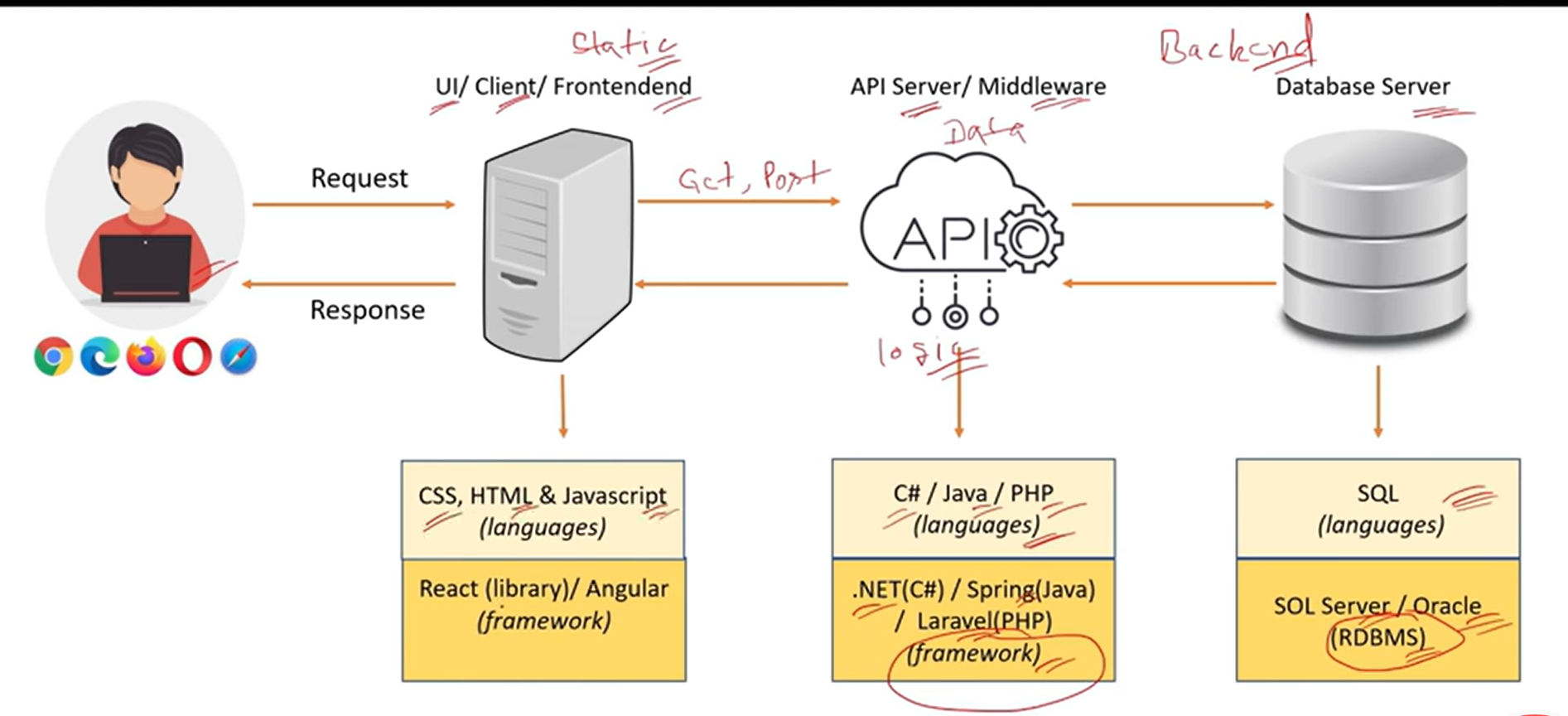
User Interaction:

The cycle repeats when a user interacts with the application again, triggering the dispatch of new actions and the subsequent update of the state.

By following this unidirectional data flow, Redux ensures a predictable and manageable way to handle application state. This pattern simplifies debugging, as you can trace state changes back to the actions that caused them, and facilitates the implementation of features like time-travel debugging and undo functionality.

**29.What is role of React in Software Development ?**

When the users opens the E-commerce application the request will hit to the UI/Client/Frontend sever . This server contains all static details of the web page but for the dynamic content like list UI server sends the request to the API servers . To create update or delete the data your API Servers will call the Data base server and the Based on API server the data will get added or deleted . Both API server and DataBase Servers are called as Backend Servers . When the data base server back with response API server will do some logical operation like sorting or fiterling then finally the data is send to the user in the response format.



RDBMS are used to simplify the complex data base operation and .net simplifies the serverside programming logic part. llly we need for frontend only html,css,js to simplify the complex UI building we need the ReactJS Libraray. React simplifies the creation of SPA by using reusable components.

**30.What is SPA ?**

A Single Page Application is a web application that have only one single web page.Whenever user do some action on the website, then in response content is dynamically updated without refreshing or loading a new page.

**31.Advantages of ReactJS?**

1.Simple to build Single Page Application by using components.

2.React is cross platform and open source(Free to use).

3.Lightweight and very fast (Virtual DOM).

4.Large Community and Ecosystem.

5.Testing is Easy.

**32.What is DOM?**

The DOM (Document Object Model) represents the web page as a tree-like structure which allows JS to dynamically access and manipulate the content and structure of a web page.

\* React uses a virtual DOM to efficiently update the UI without rerender the entire page,which helps to improve the performance and make application more responsive.

**33.What is NPM and role model of npm ?**

NPM is used to manage the dependencies for youe React Project,including the React Library itself.

**node\_modules** folder contains all the dependencies of the project,including the React Libraries.

**Public** folder contains static assets that are served directly to the user's browser,such as images,fonts and index.html file.

**Src -** In React , src folder is used to store all the source code of the application.

**Index.html -** Index.html file in a React Project is the entry point to the application.

**App.js -** App.js is the top/main/root level component in React application,and it is responsible for rendering all of thr other components.

**33.What is the use of Export default App?**

It is used to make this App component available for import using "import " statement in other files.

**34.What is the role of index.js file,ReactDOM and render method in React?**

ReactDOM is a JS library which convert your components(which are written in React syntax) to the actual browser DOM.React Dom acts like a bridge between React and browsers dom

Index.js file is the JS file which renders all the components and replace the root element of index.html file with the newly rendered root elements

**35.How React App load and display the components in Browser?**

Whenever the user hits the Request it will load to the index.html its connected to index.js which will internally call the App.js and send the response back to the index.html

**36.Advantages of JSX?**

Improve code reusability and writability.

Error checking in advance(Type safety).

Support JS expressions.

Improved Performance.

Code reusability.

**37.Can browser read a JSX FIle ? What is Babel ?**

NoBrowser cant read the jsx ,Browser can read only Regular Js object.Babel in React is used to transile JSX syntax into regular JS which browser can understand.

**38.What is Transpiler?**

A Transpiler is a tool that converts source code from one programming language to another .

**39.What are the types of conditional rendering in JSX?**

if/else statements,Ternary operator,&& operator ,Switch statement

**40.Three Principles of Redux?**

Single source of truth

The global state of your application is stored in an object tree within a single store.

State is read-only

The only way to change the state is to emit an action, an object describing what happened.

Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure reducers.