HTML

Answer 1 HTML:

<!DOCTYPE> declaration. The declaration is not an HTML tag. A DOCTYPE is nothing more than a document type declaration, which is a list of guidelines that must be attached to a certain html, xml, or sgxml online document. Because it is required for the HTML version, if the user application experiences any problems at that time, it can be quickly determined that the problem is due to incompatibility between the HTML version and the user's browser.

Answer 2 HTML :

Semantic HTML (also called semantic markup) is HTML code that uses HTML tags to effectively describe the purpose of page elements. Semantic HTML code communicates the meaning of its elements to both computers and humans, which helps web browsers, search engines, assistive technologies, and human developers understand the components of a web page.The semantic HTML tags help the search engines and other user devices to determine the importance and context of web pages.

The pages made with semantic elements are much easier to read.

It has greater accessibility. It offers a better user experience.

Answer 3 HTML :

HTML TAG

1.HTML tags are used to hold the HTML element.

2.HTML tag starts with < and ends with >

3.HTML tags are almost like keywords where every single tag has unique meaning.

HTML ELEMENTS

1.HTML element holds the content.

2.Whatever written within a HTML tag are HTML elements.

3.HTML elements specifies the general content.

Answer 4 HTML:

Github Link :- https://github.com/rahul2271/resume.git

Answer 5 HTML:

Answer 6 HTML:-

Advantages

* Multimedia support.

*Short and simple syntax.

* Improved security features.

* inlcude semantic tags.

* Cross-platform support.

Answer 7 HTML :

Github Link :- https://github.com/rahul2271/Music-Player.git

Answer 8 HTML :

<fig>

<figure> tag is a container tag.

This tag is inline element.

The figure tag contains default alignment and styling.

<img>

<img> tag is a void tag.

It is an inline element but when we specify width and height it becomes a block element.

The image tag does not contain any default alignment and styling.

Answer 9 HTML:

HTML TAG

HTML tags are used to hold the HTML element.

HTML tag starts with < and ends with >

HTML tags are almost like keywords where every single tag has unique meaning.

HTML ATTRIBUTE

HTML attributes are used to describe the characteristic of an HTML element in detail.

HTML attributes are found only in the starting tag.

HTML attributes specify various additional properties to the existing HTML element.

Answer 10 HTML :

NO IMAGE

CSS

Answer 1 CSS:

A CSS box model is a compartment that includes numerous assets, such as edge, border, padding and material. It is used to develop the design and structure of a web page. It can be used as a set of tools to personalize the layout of different components. According to the CSS box model, the web browser supplies each element as a square prism.

Properties:

The CSS box model contains the different properties in CSS. These are listed below.

Border

Margin

Padding

Content

Answer 2 CSS :

In CSS, pattern matching rules determine which style rules apply to elements in a document. These patterns, called selectors, may range from simple element types to rich contextual patterns. If all conditions in the pattern are true for a certain element, the selector matches the element.

TYPES:

The CSS element Selector

The CSS id Selector

The CSS class Selector

The CSS Universal Selector

The CSS Grouping Selector

ADVANTAGES:

It's faster than XPath.

It's much easier to learn and implement.

You have a high chance of finding your elements.

It's compatible with most browsers to date.

Answer 3 CSS :

1vh is equal to 1% of the viewport's height, with the viewport being the browser window. 1px is one physical pixel on the device's screen. Percent is the relative size compared to its parent element.

Answer 4 CSS:

display:inline

When an element is styled with display:inline, it will not start on a new line, will only take up as much width as the content it contains, and will not cause a line break after it. The HTML element is inline by default as well as several other elements listed here.

display:inline-block

The difference between an inline element and an inline-block element is that an inline-block element can take up specified width and height. But, it will also not start on a new line within its parent or cause a line break after it. The inline-block element does not start on a new line, but it takes up the specified width and height.

display:block

Any element styled with display: block is the polar opposite of display:inline. A block element starts on a new line and occupies the available width of its parent element or its specified width. The HTML [p], [div], as well as other elements listed here are block elements by default.

Answer 5 CSS:

Border-box :

The width and height properties include the content, padding, and border, but do not include the margin. Note that padding and border will be inside of the box.

For example: .box {width: 350px; border: 10px solid black;} renders a box that is 350px wide. The content box can't be negative and is floored to 0, making it impossible to use border-box to make the element disappear.

Here the dimensions of the element are calculated as: width = border + padding + width of the content, and height = border + padding + height of the content.

Content-box :

This is the initial and default value as specified by the CSS standard. The width and height properties include the content, but does not include the padding, border, or margin.

For example : .box {width: 350px; border: 10px solid black;} renders a box that is 370px wide.

Here, the dimensions of the element are calculated as: width = width of the content, and height = height of the content. (Borders and padding are not included in the calculation.)

Answer 6 CSS:

A z-index is a CSS property that controls overlapping HTML elements' vertical stacking order, which appears closer to the viewer.

The property is called "z-index" because it sets the order of elements along the z-axis.

use:

Elements can overlap for several reasons, such as when relative positioning nudges it, covering something else; a negative margin has pulled the element over another, or absolutely positioned elements overlap.

Answer 6 CSS:

GRID

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

FLEX

The flex property in CSS is shorthand for flex-grow, flex-shrink, and flex-basis. It only works on the flex-items, so if the container's item is not a flex-item, the flex property will not affect the corresponding item.

DIFFERENCE:

GRID:

Two – Dimensional

Can flex combination of items through space-occupying Features

Layout First

FLEX:

One – Dimensional

Can push content element to extreme alignment

Content First

Answer 7 CSS:

Answer 8 CSS:

Github Link :- https://github.com/rahul2271/Periodic-Table.git

Answer 9 CSS:

Github Link : https://github.com/rahul2271/Grid-Flex-CSS.git

Answer 10 CSS:

Github Link :- https://github.com/rahul2271/Responsive.git

Answer 11 CSS:

Github Link:- https://github.com/rahul2271/ineuron.git

Answer 12 CSS:

Pseudo-classes enable you to target an element when it's in a particular state, as if you had added a class for that state to the DOM. Pseudo-elements act as if you had added a whole new element to the DOM, and enable you to style that.

Differnce

A pseudo-element is a 'fake' element, it isn't really in the document with the 'real' ones. Pseudo-classes are like 'fake' classes that are applied to elements under certain conditions, much like how you would manipulate the classes of elements using JavaScript.

JavaScript

Answer 1 JS:

Hoisting is a concept that enables us to extract values of variables and functions even before initializing/assigning value without getting errors and this happens during the 1st phase (memory creation phase) of the Execution

Context.

Features of Hoisting:

In JavaScript, Hoisting is the default behavior of moving all the declarations at the top of the scope before code execution. Basically, it gives us an advantage that no matter where functions and variables are declared, they are moved to the top of their scope regardless of whether their scope is global or local.

It allows us to call functions before even writing them in our code

Answer 2 JS:

A higher order function is a function that takes one or more functions as arguments, or returns a function as its result.

There are several different types of higher order functions like map and reduce.

When working with arrays, you can use the map(), reduce(), filter(), and sort() functions to manipulate and transform data in an array.

When working with objects, you can use the Object.entries() function to create a new array from an object.

When working with functions, you can use the compose() function to create complex functions from simpler ones.

Difference

map() vs forEach()

Some of the difference between map() and forEach() methods are listed below –

The map() method returns a new array, whereas the forEach() method does not return a new array.

The map() method is used to transform the elements of an array, whereas the forEach() method is used to loop through the elements of an array.

The map() method can be used with other array methods, such as the filter() method, whereas the forEach() method cannot be used with other array methods.

Answer 3 JS:

Use .bind() when you want that function to later be called with a certain context, useful in events. Use .call() or .apply() when you want to invoke the function immediately, and modify the context.

Call/apply call the function immediately, whereas bind returns a function that, when later executed, will have the correct context set for calling the original function. This way you can maintain context in async callbacks and events.

Answer 4 JS:

Event Bubbling

While developing a webpage or a website via JavaScript, the concept of event bubbling is used where the event handlers are invoked when one element is nested on to the other element and are part of the same event. This technique or method is known as Event Bubbling. Thus, while performing event flow for a web page, event bubbling is used. We can understand event bubbling as a sequence of calling the event handlers when one element is nested in another element, and both the elements have registered listeners for the same event. So beginning from the deepest element to its parents covering all its ancestors on the way to top to bottom, calling is performed.

Example of Event Bubbling

Let's look at the below example to understand the working concept of Event Bubbling:

```
<!DOCTYPE html>

<html>

<head>

   <meta charset="utf-8">

   <meta name="viewport" content="width=device-width">

   <title>Event Bubbling</title>

</head>

<body>

   <div id="p1">

      <button id="c1">I am child button</button>

   </div>

      <script>

      var parent = document.querySelector('#p1');

         parent.addEventListener('click', function(){

            console.log("Parent is invoked");

         });

   var child = document.querySelector('#c1');

         child.addEventListener('click', function(){
```

```
            console.log("Child is invoked");
        });
    </script>
</body>
</html>
```

Event Capturing

Netscape Browser was the first to introduce the concept of Event Capturing. Event Capturing is opposite to event bubbling, where in event capturing, an event moves from the outermost element to the target. Otherwise, in case of event bubbling, the event movement begins from the target to the outermost element in the file. Event Capturing is performed before event bubbling but capturing is used very rarely because event bubbling is sufficient to handle the event flow.

Example of Event Capturing

Let's see an example code to understand the working of Event Capturing:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Event Capturing</title>
</head>
<body>
    <div id="p1">
        <button id="c1">I am Child</button>
    </div>

    <script>
```

```
        var parent = document.querySelector('#p1');
        var child = document.querySelector('#c1');


        parent.addEventListener('click', function(){
            console.log("Parent is invoked");
        },true);
        child.addEventListener('click', function(){
            console.log("Child is invoked");
        });
    </script>
</body>
</html>
```

Answer 5 JS:

Currying is an advanced function in JavaScript which is used for the manipulation of functions' arguments and parameters. It was named after "Haskell B. Curry". The concept of currying in Javascript comes from the Lambda calculus.

Currying takes a function that receives more than one parameter and breaks it into a series of unray (one parameter) functions. Hence, the currying function takes only one parameter at a time.

Uses of Currying Function

Currying in JavaScript can be for the following reasons:

Currying is helpful in Event handling.

By using the currying function, we can avoid passing the same variable many times.

Currying in JavaScript can be used to make a higher-order function.

Example One: A Simple, Three-Parameter Function

In this example we will see a simple function that will accept three parameters:

```
const addition =(x, y, z)=>{
        return x+y+z
```

```
}

console.log(addition(2, 3, 5)) // 10
```

Output:

10

Answer 6 JS:

JavaScript Promise are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code. Prior to promises events and callback functions were used but they had limited functionalities and created unmanageable code. Multiple callback functions would create callback hell that leads to unmanageable code. Promises are used to handle asynchronous operations in JavaScript.

The states of promises in JavaScript

A promise has three states:

pending: the promise is still in the works

fulfilled: the promise resolves successfully and returns a value

rejected: the promise fails with an error

CREATE OWN PROMISE

```
const STATE = {
    PENDING: 'PENDING',
    FULFILLED: 'FULFILLED',
    REJECTED: 'REJECTED',
}


function isThenable(val) {
    return val instanceof MyPromise;
}
class MyPromise {
    constructor(callback) {
```

```
    this.state = STATE.PENDING;
    this.value = undefined;
    this.handlers = [];
    try {
        callback(this._resolve, this._reject);
    } catch (err) {
        this._reject(err)
    }
}

_resolve = (value) => {
    this.updateResult(value, STATE.FULFILLED);
}

_reject = (error) => {
    this.updateResult(error, STATE.REJECTED);
}

updateResult(value, state) {
    // This is to make the processing async
    setTimeout(() => {
        // process the promise if it is still in pending state
        if (this.state !== STATE.PENDING) {
            return;
        }
```

```
        // check is value is also a promise
        if (isThenable(value)) {
            return value.then(this._resolve, this._reject);
        }

        this.value = value;
        this.state = state;

        // execute handlers if already attached
        this.executeHandlers();
    }, 0);
}

addHandlers(handlers) {
    this.handlers.push(handlers);
    this.executeHandlers();
}

executeHandlers() {
    // Don't execute handlers if promise is not yet fulfilled or rejected
    if (this.state === STATE.PENDING) {
        return null;
    }
```

```
    // We have multiple handlers because add them for .finally block too
    this.handlers.forEach((handler) => {
        if (this.state === STATE.FULFILLED) {
            return handler.onSuccess(this.value);
        }
        return handler.onFail(this.value);
    });
    // After processing all handlers, we reset it to empty.
    this.handlers = [];
}


    then(onSuccess, onFail) {
        return new MyPromise((res, rej) => {
                this.addHandlers({
                    onSuccess: function(value) {
                        // if no onSuccess provided, resolve the value for the next promise chain
                        if (!onSuccess) {
                            return res(value);
                        }
                        try {
                            return res(onSuccess(value))
                        } catch(err) {
                            return rej(err);
                        }
                    },
```

```
                onFail: function(value) {
                    // if no onFail provided, reject the value for the next promise chain
                    if (!onFail) {
                        return rej(value);
                    }
                    try {
                        return res(onFail(value))
                    } catch(err) {
                        return rej(err);
                    }
                }
            });
        });
    }

    // Since then method take the second function as onFail, we can leverage it while implementing catch
    catch(onFail) {
        return this.then(null, onFail);
    }

    // Finally block returns a promise which fails or succeedes with the previous promise resove value
    finally(callback) {
        return new MyPromise((res, rej) => {
            let val;
            let wasRejected;
```

```
                this.then((value) => {
                    wasRejected = false;
                    val = value;
                    return callback();
                }, (err) => {
                    wasRejected = true;
                    val = err;
                    return callback();
                }).then(() => {
                    // If the callback didn't have any error we resolve/reject the promise based on promise state
                    if(!wasRejected) {
                        return res(val);
                    }
                    return rej(val);
                })
            })
        }
}
const testPromiseWithLateResolve = new MyPromise((res, rej) => {
    setTimeout(() => {
        res('Promise 1 is resolved')
    }, 1000);
});
testPromiseWithLateResolve.then((val) => {
    console.log(val);
```

```
});
const testPromiseWithLateReject = new MyPromise((res, rej) => {
    setTimeout(() => {
        rej('Promise 2 is rejected')
    }, 1000);
});
testPromiseWithLateReject.then((val) => {
    console.log(val);
}).catch((err) => {
    console.log(err);
});
const testPromiseWithRejectFinally = new MyPromise((res, rej) => {
    setTimeout(() => {
        rej('Promise 2 is rejected')
    }, 1000);
});
testPromiseWithRejectFinally.finally(() => {
    console.log('finally called');
}).catch((err) => {
    console.log('value rejected after finally', err);
});
const testPromiseWithEarlyResolve = new MyPromise((res, rej) => {
        res('Promise 3 is resolved early')
});
setTimeout(() => {
```

```
    testPromiseWithEarlyResolve.then((val) => {
        console.log(val);
    });
}, 3000);
```

Answer 8 JS:

“This” keyword refers to an object that is executing the current piece of code. It references the object that is executing the current function. If the function being referenced is a regular function, “this” references the global object. If the function that is being referenced is a method in an object, “this” references the object itself.

example:

```
const person = {
    firstName: "John",
    lastName : "Doe",
    id       : 5566,
    fullName : function() {
        return this.firstName + " " + this.lastName;
    }
};
```

Answer 10 JS:

The term debounce comes from electronics. When you’re pressing a button, let’s say on your TV remote, the signal travels to the microchip of the remote so quickly that before you manage to release the button, it bounces, and the microchip registers your “click” multiple times.

To mitigate this, once a signal from the button is received, the microchip stops processing signals from the button for a few microseconds while it’s physically impossible for you to press it again.

In JavaScript, the use case is similar. We want to trigger a function, but only once per use case.

Let's say that we want to show suggestions for a search query, but only after a visitor has finished typing

it.

Or we want to save changes on a form, but only when the user is not actively working on those changes, as every "save" costs us a database trip.

And my favorite—some people got really used to Windows 95 and now double click everything

```
function debounce(func, timeout = 300){
    let timer;
    return (...args) => {
        clearTimeout(timer);
        timer = setTimeout(() => { func.apply(this, args); }, timeout);
    };
}
function saveInput(){
    console.log('Saving data');
}
const processChange = debounce(() => saveInput());
```

Answer 11 JS:

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

To use a closure, define a function inside another function and expose it. To expose a function, return it or pass it to another function.

The inner function will have access to the variables in the outer function scope, even after the outer function has returned.

1. Tracking DOM State / Styles

There are quite a few use cases for closures when it comes to working with the DOM. One thing you

might want to do, for example, is keep track of the initial styles of some DOM element, even over the course of multiple style changes.

2. Singletons

The term ‘singleton’ refers to a design pattern in which a given class is only instantiated once and only that one, single instance is ever made available publicly. This is a somewhat controversial design pattern for reasons that probably make more sense in a strict OOP architecture, but I think singletons can still be useful for some isolated services that are not a core part of your business logic. Folks with more experience in OOP will probably have a lot more to say on this topic.

3. Higher-Order Functions

A higher-order function is a function that either takes one or more functions as arguments or returns a function. This is a really useful concept that we can use to help us avoid having to write boilerplate code over and over. We can create higher-order functions that, in turn, create utility functions we can use any time we find ourselves repeatedly writing similar code.

One simple example is a function that creates functions to help us round integers to a certain number of decimal places. Imagine in some cases we have a float that we want to round to the nearest whole number, while in some cases we want to maintain two decimal places.

Answer 12 JS:

Gibhub link :- https://github.com/rahul2271/Blog.git

React

Answer 1 React :

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application. It was created by Jordan Walke, who was a software engineer at Facebook. It was initially developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed ReactJS in 2011 in its newsfeed section, but it was released to the public in the month of May 2013.

1. Easy to Learn and USe

2. Creating Dynamic Web Applications Becomes Easier

3. Reusable Components

4. Performance Enhancement

5. The Support of Handy Tools

6. Known to be SEO Friendly

7. The Benefit of Having JavaScript Library

8. Scope for Testing the Codes

Answer 2 React:

Simply put, the virtual DOM is a representation of a DOM object. In React JS, every DOM element has a corresponding Virtual DOM Object. No doubt that the virtual DOM has the same properties as we have in the normal DOM object but unlike the DOM object where we can directly change what is on the screen, we cannot do that for the virtual DOM.

The virtual DOM provides a mechanism that abstracts manual DOM manipulations away from the developer, helping us to write more predictable code. It does so by comparing two render trees to determine exactly what has changed, only updating what is necessary on the actual DOM.

ADVANTAGES:

It is clear that the performance provided by the Virtual DOM is amazing. Not only that, below are some advantages of the Virtual DOM:

Speed and performance boost

Lightweight

It is simple and clear

Amazing diffing algorithm

It can be used on other frameworks not just react

Answer 3 React:

We have seen so far that React web apps are actually a collection of independent components that run according to the interactions made with them. Every React Component has a lifecycle of its own, lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component's existence. The definition is pretty straightforward but what do we mean by different stages? A React Component can go through ***four stages*** of its life as follows.

Initialization: This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.

Mounting: Mounting is the stage of rendering the JSX returned by the render method itself.

Updating: Updating is the stage when the state of a component is updated and the application is repainted.

Unmounting: As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

React provides the developers with a set of predefined functions that if present are invoked around specific events in the lifetime of the component. Developers are supposed to override the functions with desired logic to execute accordingly. We have illustrated the gist in the following diagram.

Answer 4 React:

Functional Components:

JavaScript function that returns a JSX element

Uses React Hooks to manage state

Uses React Hooks to handle lifecycle events like useState, useEffect, useMemo.

Props are passed as a parameter to the function

Class Components:

JavaScript class that extends React.Component and has a render() method that returns a JSX element

Has built-in state management using this.state

Uses class lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount

Props are accessed through this.props

Answer 5 React:

Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.

Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace your knowledge of React concepts.

When to use a Hooks?

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

Can we use Hooks in Class Components?

You can't use a hook directly in a class component, but you can use a hook in a wrapped function component with

a render prop to achieve this.

Before going ahead with this, if you're able to convert your class component to a function component, prefer that.

Answer 6 React:

React lifecycle methods play a significant role in building projects in React. If you have ever had trouble understanding and implementing them, don't worry, you are not alone. Almost everyone who first started coding in React has run into situations where they were confused because of lifecycle methods. React lifecycles have three phases

1. Mounting – Component is born

2. Update – Component grows

3. Unmount – Component dies

Common React Lifecycle Methods

render()

componentDidMount()

componentDidUpdate()

componentWIllUnmount()

shouldComponentUpdate()

getSnapShotBeforeUpdate()

ADVANTAGES:

It revolutionizes the way you write components.

You can write concise and clearer code.

Hooks are simpler to work with and test. ...

A related logic could be tightly coupled inside a custom hook.

It simplifies how to make code more composable and reusable.

Answer 7 React:

The React useState Hook allows you to have state variables in functional components. You pass the initial state to this function, and it returns a variable with the current state value (not necessarily the initial state) and another function to update this value.

useState is React Hook that allows you to add state to a functional component. It returns an array with two values: the current state and a function to update it. The Hook takes an initial state value as an argument and returns an updated state value whenever the setter function is called. It can be used like this:

const [state, setState] = useState(initialValue);

Here, the initialValue is the value you want to start with, and state is the current state value that can be used in your component. The setState function can be used to update the state, triggering a re-render of your component.

ADVANTAGES:

Ease of Use

useState is very easy to use. It takes a single argument, which is the initial state and returns an array with two elements: the current state and a function to update the state.

Simple State Updates

useState is suitable for simple state updates. If you only need to update one piece of state, useState is a good choice.

Less Boilerplate

useState requires less boilerplate code compared to useReducer. There's no need to define a separate action object or switch statement.

Answer 8 React:

useEffect hook is part of React's Hooks API. The core principle of this hook is to let you perform side effects in your functional components. The useEffect hook is a smooth combination of React's lifecycle methods like componentDidMount, componentDidUpdate and componentWillUnmount. According to React documentation, the useEffect hook was developed to overcome some challenges posed by the life cycle methods of ES6 class components.

Sometimes, we want to run some code after the DOM has been updated. It can be anything, showing pop-ups, sending API requests, logging users' information etc. and such functions don't require cleanup to be performed. They are just hit-once functions and then we forget about them. Such places are the best examples to use the useEffect hook.

ADAVNTAGES:

The useEffect hook will make a network request on component render.

You can write concise and clearer code.

Answer 9 React:

The React Context API is a way for a React app to effectively produce global variables that can be passed around. This is the alternative to "prop drilling" or moving props from grandparent to child to parent, and so on. Context is also touted as an easier, lighter approach to state management using Redux.

Context API is a (kind of) new feature added in version 16.3 of React that allows one to share state across the entire app (or part of it) lightly and with ease.

Github Link:- https://github.com/rahul2271/Theme-Minor-Project.git

Answer 10 React:

The useReducer hook is a built-in hook in React that provides a way to manage complex state logic by using a reducer function. It is an alternative to the useState hook and is particularly useful when dealing with state that involves multiple sub-values or complex state transitions.The useReducer hook takes two arguments: a reducer function and an initial state value. It returns an array with two elements: the current state and a dispatch function. The reducer function receives the current state and an action as arguments and returns the new state based on the action.

Advantages of the useReducer hook:

Centralized state management

Predictable state updates

Complex state structures

Middleware and additional logic

Answer 11 React:

Link: https://github.com/rahul2271/todo-app.git

Answer 12 React:

Link: https://github.com/rahul2271/counter-app.git

Answer 13 React:

Link: https://github.com/rahul2271/calculator.git

Answer 15 React:

Prop drilling refers to the process of passing props through multiple intermediate components that do not need the props themselves but only serve as a conduit for passing the props down to the child components that actually need them. Prop drilling can lead to code clutter and reduced maintainability, as well as make it harder to understand and track the flow of data in the application.

To avoid prop drilling, there are a few alternative approaches you can consider:

Use React Context

Use Redux

Use React Query

Use Component Composition

ExpressJS

Answer 1 ExpressJs:

Middleware in Express are functions that come into play after the server receives the request and before the response is sent to the client. They are arranged in a chain and are called in sequence.

We can use middleware functions for different types of processing tasks required for fulfilling the request like database querying, making API calls, preparing the response, etc, and finally calling the next middleware function in the chain.

Answer 4 ExpressJs:

Authentication

Determines whether users are who they claim to be

Challenges the user to validate credentials (for example, through passwords, answers to security questions, or facial recognition)

Usually done before authorization

Generally, transmits info through an ID Token

Generally governed by the OpenID Connect (OIDC) protocol

Example: Employees in a company are required to authenticate through the network before accessing their company email

Authorization

Determines what users can and cannot access

Verifies whether access is allowed through policies and rules

Usually done after successful authentication

Generally, transmits info through an Access Token

Generally governed by the OAuth 2.0 framework

Example: After an employee successfully authenticates, the system determines what information the employees are allowed to access

Answer 5 ExpressJs:

CommonJS

Works with the Node.js platform

Compiled into AMD modules

All dependencies are listed in the same file

No type-checking capabilities

Packaging up functionality into small pieces

ES Module

Works with the web browser environment

Does not require a module loader like AMD

Reference any other module in the same package available on the global namespace

Robust typing support via imports

Declare dependencies between modules

Answer 6 ExpressJs:

Github Link:- https://github.com/rahul2271/JWT.git

Answer 7 ExpressJs:

Before storing password into database, we have to encrypt it then make it undefined so no other request can fetch password.

Answer 8 ExpressJs:

Node handle request in Event loop, it is an event-listener which listen and process the event.

Answer 9 ExpressJs:

Github Link: https://github.com/rahul2271/E-commerce-fashionista.git