← HTML QUESTIONS →

Answer 1.

No, <!DOCTYPE html> is not a tag in HTML. It is called a Document Type Declaration (DTD).

The <!DOCTYPE html> declaration is used to specify the version of HTML being used in the document. It informs the browser about the markup language and its version, allowing it to render the page correctly.

It is placed at the very beginning of an HTML document, before the <html> tag. The <!DOCTYPE html> declaration is used in modern HTML5 documents as a standard way to define the document type. It helps ensure that the web page is interpreted and rendered correctly by the browser.

Answer 2.

Semantic tags in HTML are elements that provide meaning and context to the structure of a web page. They convey the purpose or role of the content within the document. Some examples of semantic tags in HTML5 include <header>, <nav>, <section>, <article>, <footer>, and <main>.

Reasons why we need semantic tags in HTML:

Improved Accessibility: Semantic tags enhance the accessibility of web content.

Better Search Engine Optimization (SEO): Search engines rely on the structure and semantic meaning of web pages to understand and index their content accurately. By using semantic tags, you provide clearer signals to search engines about the purpose and hierarchy of the content

Answer 3.

HTML Tags and Elements:

HTML Tags:

HTML tags are the markup elements used to define the structure and presentation of content within an HTML document.
They are enclosed in angle brackets (< and >) and appear as opening and closing tags.
Tags are used to mark up specific parts of the content and indicate how they should be rendered by the web browser.
HTML Elements:

HTML elements consist of both the opening and closing tags along with the content placed between them.
Elements are formed by pairing the opening tag, content, and closing tag together.
Elements represent the individual building blocks of an HTML document and define the structure and meaning of the content.

Answer 4.

Answer 5.

Answer 6.

Some advantages of HTML5 over its previous versions are:

Improved Multimedia Support:

HTML5 introduced native support for multimedia elements such as <video> and <audio>.
It eliminated the need for third-party plugins like Flash to play videos and audio on web pages.
This native support provides better compatibility, performance, and a more seamless multimedia experience.
Enhanced Semantics and Structure:

HTML5 introduced a set of new semantic elements such as <header>, <nav>, <section>, and <footer>.
These semantic elements provide a clearer and more meaningful structure to web pages, improving accessibility and search engine optimization.

Answer 7.
https://github.com/cybermonk01/music_html

Answer 8.
The <figure> tag and the <img> tag have different purposes:

<figure> tag: The <figure> tag is used to represent self-contained content that is referenced from the main content. It is typically used to wrap media content, such as images, illustrations, diagrams, or videos, along with their associated captions or descriptions. It provides a semantic way to group related content together.

<img> tag: The <img> tag is specifically used to insert an image into an HTML document. It is a self-closing tag that requires the src attribute to specify the image file's source URL. The <img> tag does not include any specific structure or additional content and is primarily focused on embedding an image within the document.

Answer 9.

HTML Tag:
An HTML tag is an element that defines the structure and content of an HTML document.
Tags are enclosed in angle brackets (< and >) and consist of the tag name and any attributes.
Examples of HTML tags include , <p>, <h1>, and <a>.
HTML Attribute:

An HTML attribute provides additional information or modifies the behavior of an HTML element.
Attributes are added within the opening tag of an HTML element and consist of a name and a value, separated by an equals sign (=).
Examples of HTML attributes include href in <a href="https://example.com">, class in , and src in <img src="image.jpg">.
Global Attributes:

Global attributes are attributes that can be used with any HTML element.
They provide common functionalities and behaviors that can be applied universally.
Examples of global attributes include:
class: Specifies one or more class names for an element.
id: Specifies a unique identifier for an element.
style: Specifies inline CSS styles for an element.

Answer 10.https://github.com/cybermonk01/table_html

← CSS QUESTIONS →

Answer 1.

The Box Model in CSS is a concept that defines how elements are rendered and spaced on a web page. It consists of four main components: content, padding, border, and margin. These components surround an element and affect its total size and layout.

The CSS properties that are part of the Box Model are:

Content: The width and height properties define the dimensions of the content area.
Padding: The padding property defines the space between the content area and the border.
Border: The border property defines the border around the element.
Margin: The margin property defines the space between the border and adjacent elements.
These properties allow developers to control the size, spacing, and positioning of elements on a web page by manipulating the different components of the Box Model.

Answer 2.

The different types of selectors in CSS are:

Element Selectors: Selects elements based on their HTML tag name. For example, p selects all <p> elements.
Class Selectors: Selects elements based on their class attribute. For example, .highlight selects all elements with class="highlight".
ID Selectors: Selects an element based on its unique ID attribute. For example, #myElement selects the element with id="myElement".
Attribute Selectors: Selects elements based on their attribute values. For example, [type="submit"] selects all elements with type="submit".
Pseudo-classes: Selects elements based on their state or position within the document. For example, :hover selects an element when the user hovers over it.
Pseudo-elements: Selects a specific part of an element. For example, ::before selects the content before an element.
Advantages of CSS Selectors:

Specificity: Different types of selectors allow you to target specific elements or groups of elements with precision. This helps in applying styles to specific parts of a web page.
Reusability: Class and ID selectors allow you to apply styles to multiple elements that share the same class or ID. This promotes code reusability and reduces duplication.

Answer 3 .

VW (Viewport Width) and VH (Viewport Height) are units of measurement in CSS that are relative to the size of the viewport, which is the visible portion of the browser window.

The key differences are:

Relative vs. Absolute: VW and VH are relative units, while PX is an absolute unit. VW and VH adjust their values based on the size of the viewport, whereas PX has a fixed value that does not change.

Responsive Design: VW and VH are commonly used in responsive web design to create layouts that adapt to different screen sizes.

Answer 4 .

Inline elements:

Inline elements do not start on a new line; they flow within the text content.

Examples of inline elements are , <a>, , and .

Inline-block elements:

Inline-block elements behave similarly to inline elements by flowing within the text content, but they can have a width and height property.

Examples of inline-block elements are <img>, <button>, and <input>.

Block elements:

Block elements start on a new line and occupy the full available width of their parent container by default.
Block elements have a width and height property, and their width expands to fill the available space.

Examples of block elements are , <p>, <h1> to <h6>, and <section>.

Answer 5.

The main difference between border-box and content-box is how they calculate the total size of an element in CSS.

border-box:
In the border-box model, the specified width and height of an element include the content, padding, and border.
When you add padding or border to an element with border-box box-sizing, it does not increase the total width and height of the element. Instead, the padding and border are included within the specified width and height.

content-box:
In the content-box model (the default box-sizing value), the specified width and height of an element only include the content.
When you add padding or border to an element with content-box box-sizing, it increases the total width and height of the element. The padding and border are added to the specified width and height.

Answer 6.

z-index is a CSS property that controls the stacking order of positioned elements on a web page. It determines how elements are layered and displayed in relation to each other along the z-axis.

The z-index property accepts integer values, where a higher value represents a higher stacking order.

To use z-index, you can assign a value to it in CSS, such as z-index: 1; or z-index: -1;. Elements with a higher z-index value will be visually positioned on top of elements with a lower z-index value.

z-index is particularly useful when dealing with overlapping elements or creating layered effects in web design, allowing you to control the visual hierarchy and placement of elements in the z-axis dimension.

Answer 6.

Flexbox:

Flexbox is a one-dimensional layout system that operates along a single axis, either horizontally or vertically.
It provides a flexible and efficient way to distribute space among elements within a container.

With Flexbox, you can easily control the alignment, order, and size of items within a flex container.
It allows for easy responsiveness and adaptation to different screen sizes and devices.
Flexbox is well-supported by modern browsers and has extensive browser compatibility.
Grid:

Grid is a two-dimensional layout system that allows you to create complex layouts with rows and columns.
It provides a powerful grid-like structure for placing and aligning elements on a web page.
With Grid, you have precise control over the placement, size, and alignment of grid items.
It enables you to create responsive layouts that adapt to different screen sizes and device orientations.
Grid offers advanced features such as grid templates, grid areas, and grid auto-placement, allowing for intricate and flexible layouts.
In summary, Flexbox is a versatile one-dimensional layout system suitable for simpler, flexible layouts, while Grid is a robust two-dimensional layout system that offers more control and complexity for creating advanced, grid-based layouts. Both Flexbox and Grid provide powerful tools for building responsive and visually appealing web page layouts.

Answer 7.

Absolute Positioning:

When an element is positioned using position: absolute;, it is removed from the normal flow of the document and positioned relative to its closest positioned ancestor.

Example: Suppose you have a navigation bar at the top of your webpage, and you want to position a dropdown menu below a specific link. You can use absolute positioning to place the dropdown menu precisely below the link, regardless of the surrounding elements.
Relative Positioning:

When an element is positioned using position: relative;, it remains in the normal flow of the document, but its position can be adjusted relative to its normal position.
Relative positioning allows you to shift an element from its original position using offset properties such as top, right, bottom, or left.

Elements positioned relatively can still interact with other elements and may affect the layout of surrounding content.
Example: Suppose you want to move a paragraph slightly to the right without affecting the layout of other elements. You can use relative positioning and set a positive value for the left property to achieve the desired offset.

Fixed Positioning:

When an element is positioned using position: fixed;, it is positioned relative to the viewport (the browser window) and remains fixed in place, even when the page is scrolled.

Example: Imagine you have a floating "Back to Top" button that you want to remain visible at all times. You can use fixed positioning to keep the button fixed in the corner of the viewport, even as the user scrolls down the page.

Sticky Positioning:

Sticky positioning is a hybrid of relative and fixed positioning.

Sticky positioning is commonly used for headers or navigation menus that "stick" to the top of the screen as the user scrolls down the page.
Example: Suppose you have a navigation menu that you want to remain fixed at the top of the viewport until the user scrolls past it, and then it becomes part of the normal document flow. You can use sticky positioning to achieve this behavior.

Answer 8. https://github.com/cybermonk01/periodic-_table_css
Answer 9.
Answer 10.

Answer 11.
https://github.com/cybermonk01/i-neuron

Answer 12.

Pseudo-classes in CSS:

Pseudo-classes are keywords that can be added to selectors to style specific states or elements based on user interactions or element characteristics. They represent a particular state or condition of an element, such as :hover (when the mouse is over the element) or :focus (when the element is in focus).

Difference between Pseudo-classes and Pseudo-elements:

Pseudo-classes target elements based on their state or user interaction, while pseudo-elements target specific parts or sections of an element's content.
Pseudo-classes are prefixed with a single colon (:), while pseudo-elements are prefixed with double colons (::).
Pseudo-classes are added directly to selectors, while pseudo-elements are added after the selector.

← JavaScript Questions →

Answer 1.

In JavaScript, hoisting is a mechanism where variable and function declarations are moved to the top of their containing scope during the compilation phase, before the code is executed. This means that regardless of where variables and functions are declared in your code, they are treated as if they are declared at the top of their scope

Variable Hoisting:

```
console.log(x); // undefined
var x = 10;
```

In the above example, even though the variable x is accessed before it is assigned a value, it does not result in an error. This is because during hoisting, the declaration var x; is moved to the top of the scope, which means x is defined but has the value undefined until it is explicitly assigned.

Answer 2.
In JavaScript, higher-order functions are functions that can accept other functions as arguments or return functions as results. They provide a way to manipulate and process data in a functional programming style.

map(): The map() function is used to create a new array by applying a provided function to each element of the original array. It returns a new array of the same length as the original, where each element is the result of the function applied to the corresponding element of the original array.

Example:

```
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map((num) => num * num);

console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

forEach(): The forEach() function is used to iterate over the elements of an array and perform a specified operation on each element. It executes a provided function once for each array element, but it does not return a new array. It is mainly used for its side effects, such as logging or modifying elements in place.

Example:

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach((num) => console.log(num));

// Output:
// 1
// 2
// 3
// 4
// 5
```

Answer 3.

```
let person = {
  name: "Krish",
  lName: "Naik",
};

function printFullName(hometown, country) {
  console.log(this.name + "_" + this.lName + " from " + hometown + country);
}

printFullName.call(person, "Lucknow", "India");

printFullName.apply(person, ["Lucknow", "India"]);

const newFun = printFullName.bind(person, "Lucknow", "India");
console.log(newFun);
newFun();
```

Output:

Answer 4.

In JavaScript, event propagation refers to the order in which events are handled when an element is nested within another element and both elements have registered event handlers. There are two main models of event propagation: event bubbling and event capturing.

Event Bubbling:

Event bubbling is the default event propagation mechanism in most modern web browsers. With event bubbling, when an event is triggered on a nested element, the event is first handled by that element's event handlers, then propagated to its parent element, and continues to propagate up the DOM tree until it reaches the document root.

Example:

```
<div id="outer">
  <div id="inner">
    Click me!
  </div>
</div>

document.getElementById("outer").addEventListener("click", function() {
  console.log("Outer clicked!");
});

document.getElementById("inner").addEventListener("click", function() {
  console.log("Inner clicked!");
});
```

In this example, if you click on the "inner" div element, both event handlers will be executed, and the following output will be:

Inner clicked!
Outer clicked!

The event starts at the inner element, triggers its event handler, and then propagates up to the outer element, triggering its event handler.

Event Capturing:

Event capturing is an alternative event propagation mechanism, less commonly used than event bubbling. With event capturing, the event is first captured by the outermost element, and then it propagates down the DOM tree, reaching the target element.

Example:

```
<div id="outer">
  <div id="inner">
    Click me!
  </div>
</div>

document.getElementById("outer").addEventListener("click", function() {
  console.log("Outer clicked!");
}, true);

document.getElementById("inner").addEventListener("click", function() {
  console.log("Inner clicked!");
}, true);
```

By adding the optional third parameter true to the addEventListener() method, we enable event capturing. In this example, if you click on the "inner" div element, the event will be captured at the outer element first, and then at the inner element. The output will be:
Outer clicked!
Inner clicked!

The event starts at the outer element, triggers its event handler first, and then propagates down to the inner element, triggering its event handler.

Answer 5.

Function currying is a technique in JavaScript where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument. The curried function allows partial application of arguments, meaning you can provide some arguments now and pass the remaining arguments later. This enables creating more specialized or reusable functions.
Example:

```
const calc = (a, b, c) => {
  return a + b + c;
};

console.log(calc(1, 2, 3));

const Addition = (a) => {
  return function (b) {
    return function (c) {
      return a + b + c;
    };
  };
};

console.log(Addition(1)(2)(3));
```

Addition will pass three arguments as the function will utilize it then.

Answer 6.

Answer 7.

Promises are objects in JavaScript that represent the eventual completion (or failure) of an asynchronous operation and allow you to handle the result asynchronously. Promises provide a more readable and structured way to handle asynchronous code compared to traditional callback functions.

The different states of a promise are:

Pending: The initial state of a promise. It means that the asynchronous operation associated with the promise is still ongoing, and the final outcome is not yet determined.

Fulfilled: The state of a promise when the asynchronous operation is successfully completed. It means that the promised value or result is available.

Rejected: The state of a promise when the asynchronous operation encounters an error or failure. It means that the promised value or result cannot be obtained due to an error.

Example:

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const data = { name: 'John', age: 25 };

      resolve(data);

      // reject(new Error('Failed to fetch data.'));
    }, 2000);
  });
}

fetchData()
  .then((data) => {
    console.log('Data:', data);
  })
  .catch((error) => {
    console.error('Error:', error);
  });
```

In this example, the fetchData function returns a new promise. Inside the promise's executor function, we use setTimeout to simulate an asynchronous operation that takes 2 seconds to complete.

In the resolved state (fulfilled), we call the resolve function and pass the data object as the promised value. In the rejected state, we call the reject function and pass an Error object as the reason for the rejection.

Answer 8.

In JavaScript, the this keyword refers to the context in which a function is executed. It allows access to the object or value that is currently being referred to within the function.

The value of this is determined dynamically at runtime based on how a function is invoked.

The behavior of this can vary depending on the way a function is called:

Global Scope: In the global scope (outside of any function), this refers to the global object. In a web browser, the global object is usually the window object.

Example:

```
console.log(this === window); // Output: true

function myFunction() {
  console.log(this === window); // Output: true
}

myFunction();
```

In this example, this is the global object, which is window in a web browser. Both the console.log() statements will output true because this refers to the global object in the global scope as well as inside the function.

Answer 9.

The event loop is a mechanism in JavaScript that manages the execution of asynchronous code. It ensures that the program remains responsive by continuously checking for tasks in the callback queue and microtask queue, and executing them when appropriate.

The call stack is a data structure used to keep track of function calls during program execution. It maintains the order of function calls and manages their execution by pushing new frames onto the stack when functions are invoked and popping them off when they complete.

The callback queue is a queue that holds asynchronous tasks or callbacks that are ready to be executed. These tasks are typically the result of events like mouse clicks or network responses. The event loop processes the callback queue, executing the corresponding callback functions.

The microtask queue is a queue that holds microtasks, which are tasks with higher priority than regular tasks in the callback queue. Microtasks are created by Promise resolutions,

process.nextTick, or queueMicrotask functions. The event loop processes the microtask queue before regular tasks, ensuring that microtasks are executed promptly.

In summary, the event loop manages the execution of asynchronous code by checking the call stack, processing the callback queue, and giving priority to microtasks in the microtask queue.

Answer 10.

Debouncing is a technique used in JavaScript to control the frequency of execution of a particular function. It helps to limit the number of times a function is invoked by postponing its execution until a certain period of inactivity has passed.

Example: https://github.com/cybermonk01/debounced_search

Answer 11.
A closure is a function that remembers and has access to variables from its parent scope, even when it is executed outside that scope.

Closures are created when an inner function is returned from an outer function, and the inner function maintains a reference to variables in the outer function's scope.

Use cases of closures:

Data Privacy: Closures are commonly used to create private variables and encapsulate data within functions. By using closures, you can define variables within an outer function that are inaccessible from outside the function, but can still be accessed and modified by the inner function. This provides a way to achieve data privacy and prevents unwanted external access to variables.

Example:

```
function createCounter() {
  let count = 0;
```

```
  return function() {
    count++;
    console.log(count);
  };
}

const counter = createCounter();
counter(); // Output: 1

counter(); // Output: 2
```
In this example, the createCounter function returns an inner function that has access to the count variable defined in its parent scope. The count variable is private and can only be accessed and modified through the counter function, which maintains a closure over the count variable.

Function Factories: Closures can be used to create function factories, which are functions that generate specialized functions based on different parameters or configurations. The generated functions can retain access to the factory's variables and configurations, making them powerful and customizable

Example

```
function createMultiplier(factor) {
  return function(number) {
    return number * factor;
  };
}

const double = createMultiplier(2);
console.log(double(5)); // Output: 10

const triple = createMultiplier(3);
console.log(triple(5)); // Output: 15
```

In this example, the createMultiplier function is a function factory that generates multiplier functions based on a given factor. The generated functions can multiply a given number by the specified factor. Each generated function retains access

to its corresponding factor value through closures, allowing for dynamic and reusable code.

Answer 12.

Answer 1.

React is a popular JavaScript library for building user interfaces. It was developed by Facebook and is widely used in web development for creating dynamic and interactive UI components. React follows a component-based architecture, allowing developers to create reusable UI elements and compose them to build complex UIs.

Advantages of React-
React makes use of a virtual DOM, which is an in-memory version of the real DOM. It enables React to efficiently update and render only the components that are required when the application state changes. This leads to greater performance and a more pleasant user experience.

Declarative syntax: React employs declarative terminology, which allows developers to express how the user interface should look based on the current state. This makes the UI code easier to comprehend and reason about because it concentrates on "what" should be rendered rather than "how" to accomplish it.

React provides quick updates.

Answer 2.

In React, the Virtual DOM (Document Object Model) is a lightweight copy of the actual DOM. It is a representation of the UI components and their structure in memory. When there are updates to the application state, React compares the Virtual DOM with the real DOM and applies only the necessary changes to update the UI efficiently.

Advantages of Virtual DOM -
Optimisation of performance: The Virtual DOM enables React to reduce the number of direct modifications to the actual DOM, which can be costly in terms of performance. Instead of refreshing the entire DOM tree, React determines which components need to be changed by comparing the Virtual DOM to the prior state. The reconciliation method minimises the overall number of DOM modifications and increases performance.

Efficient updates: To discover the differences between the current Virtual DOM and the prior Virtual DOM, React employs a diffing mechanism. React may update only the relevant sections of the actual DOM by recognising the specific changes, decreasing the amount of effort required to update the UI. This method dramatically enhances the performance of UI updates, particularly for mobile devices.

Answer 3.

The lifecycle of React components refers to the different phases or stages that a component goes through from its creation to its removal from the DOM.

The React component lifecycle can be divided into three main phases:
1.Mounting
2.Updating
3.Unmounting

Answer 4.

The difference between functional components and class components in React can be summarized as follows:

Functional Components: Functional components are stateless components that are defined as JavaScript functions. They are simpler and easier to read and write compared to class components. Functional components are mainly used for presenting UI elements based on the input props they receive.

Class Components: Class components are JavaScript classes that extend the React.Component class. They have more features and flexibility compared to

functional components. Class components can have their own internal state using the this.state object. They can define lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount, allowing for more control over component behavior.

Answer 5.

React 16.8 introduced a feature called "Hooks in Classes," which allows you to use some hooks within class components. This feature is primarily intended for gradual migration from class components to functional components with hooks. It enables you to use hooks like useState and useEffect in class components by using a higher-order component called "withHooks."

Here's an example of using hooks in a class component using the "withHooks" higher-order component:

```
import { withHooks } from 'react-hooks-helper';

class MyComponent extends React.Component {
  render() {
    const [count, setCount] = this.props.useState(0);

    this.props.useEffect(() => {
      console.log('Component mounted');
      return () => {
        console.log('Component unmounted');
      };
    }, []);

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

```
}

export default withHooks(MyComponent);
```

Here we use the useState and useEffect hooks within the class component by accessing them through the this.props object provided by the withHooks higher-order component.

Answer 6.

The lifecycle of React components refers to the different phases or stages that a component goes through from its creation to its removal from the DOM.

The React component lifecycle can be divided into three main phases:
1.Mounting
2.Updating
3.Unmounting

Advantages of lifecycle methods in React:

Control over component behavior: Lifecycle methods give you control over what happens at specific points in a component's lifecycle. This allows you to perform actions like data fetching, state updates, or DOM manipulations at the appropriate times.

Side effects management: Lifecycle methods like componentDidMount and componentDidUpdate provide a convenient place to manage side effects, such as making API calls or subscribing to event listeners.

Optimize performance: Lifecycle methods allow you to optimize performance by controlling when expensive operations should occur. For example, you can

prevent unnecessary re-renders by using shouldComponentUpdate to determine if a component should update based on changes in props or state.

Answer 7.

The useState hook is a built-in hook in React that allows functional components to have their own internal state. It provides a way to store and update state within a functional component without the need for a class component. The useState hook returns an array with two elements: the current state value and a function to update that value.

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

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

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default Counter;
```

Advantages of the useState hook:

Simplicity: The useState hook simplifies state management in functional components by allowing you to declare and update state directly within the component function. This eliminates the need for creating a separate class component and managing state through the this.state object and setState method.

No class syntax: With the useState hook, you can use functional components instead of class components. This leads to cleaner and more concise code, as you don't need to deal with the complexities of class syntax, lifecycle methods, or this binding.

Answer 8.

The useEffect hook is a built-in hook in React that allows functional components to perform side effects, such as fetching data, subscribing to events, or manipulating the DOM. It replaces the functionality of lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.

The useEffect hook takes two arguments: a callback function and an optional array of dependencies. The callback function is executed after the component has rendered, and it can contain any code that needs to perform side effects. The dependencies array is used to specify values that the effect depends on. If any of the dependencies change, the effect is re-executed.

Here's an example of how to use the useEffect hook:

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

function UserProfile() {
  const [user, setUser] = useState(null);

  useEffect(() => {
    // Fetch user data
```

```
    fetch('https://api.example.com/user')
      .then(response => response.json())
      .then(data => setUser(data));
  }, []);

  return (
    <div>
      {user ? (
        <div>
          <h1>{user.name}</h1>
          <p>{user.email}</p>
        </div>
      ) : (
        <p>Loading user data...</p>
      )}
    </div>
  );
}

export default UserProfile;
```

Advantages of the useEffect hook:

Simplified side effects: The useEffect hook simplifies the management of side effects in functional components. It allows you to consolidate all the code related to side effects within a single effect callback, making it easier to read and maintain.

Lifecycle control: The useEffect hook covers all aspects of the component's lifecycle, including mounting, updating, and unmounting. By specifying dependencies in the dependencies array, you have fine-grained control over when the effect should run. This helps prevent unnecessary re-renders and ensures that the effect is executed only when needed.

Answer 9.
https://github.com/cybermonk01/light_dark

Answer 10.
The useReducer hook is a built-in hook in React that provides an alternative way to manage complex state logic within functional components. It is often used as an alternative to the useState hook when state transitions involve more complex logic or have multiple possible outcomes.

The useReducer hook follows the same principles as the Redux library, where state transitions are managed through actions and a reducer function. It takes in a reducer function and an initial state value, and returns an array with the current state value and a dispatch function to trigger state transitions.

Here's the basic syntax of the useReducer hook:

```
const [state, dispatch] = useReducer(reducer, initialState);
```

state: Represents the current state value.
dispatch: A function that is used to trigger state transitions by sending an action to the reducer function.
reducer: A function that receives the current state and an action, and returns a new state based on the action type.

Advantages of the useReducer hook:

Centralized state logic: The useReducer hook allows you to centralize complex state logic in a single reducer function. This makes it easier to understand and maintain the logic, especially when the state transitions involve multiple variables or have different outcomes based on the action type.

Predictable state updates: By following the principles of the Redux pattern, useReducer enforces a strict flow of state updates. State transitions are determined by dispatching actions to the reducer function, making it easier to reason about how the state will change in response to different actions

Answer 11.
https://github.com/cybermonk01/neuron-projects/tree/master/todo_webApp/todo

Answer 12.
https://github.com/cybermonk01/neuron-projects/tree/master/todo_webApp/todo

Answer 13.
https://github.com/cybermonk01/calculator

Answer 14.
https://github.com/cybermonk01/tic_tac_toe

Answer 15.

Prop drilling is a term used in React to describe the process of passing props through multiple levels of nested components, even if some intermediate components do not need those props. It can occur when a component needs to access a prop that is provided by a parent component, but it has to pass that prop down to its child components, and so on.

The disadvantages of prop drilling are:

Complexity: Prop drilling can make the codebase more complex and harder to understand. It requires tracking the flow of props through multiple components, which can become confusing, especially in larger applications with deep component hierarchies.

To avoid prop drilling, we can make use of two approaches:

Context API: The Context API in React allows you to create a context and share data across multiple levels of components without the need for explicit prop passing. By using the createContext and Provider components, you can define a context at a higher level and consume it in any child component that needs access to the shared data. This way, you can avoid passing props through intermediate components that don't require them.

State Management Libraries: State management libraries like Redux provide a centralized store that can be accessed by any component in the application. Instead of passing props through the component tree, you can dispatch actions or access the shared state directly from any component.

← Express Question →

Answer 1.
https://github.com/cybermonk01/neuron-projects/tree/master/20posts

Answer 2.
A middleware in the context of web development refers to a software component that sits between the client and server components of an application. It intercepts and processes requests and responses, allowing for additional functionality to be added to the application's request/response cycle. Middlewares are commonly used to implement cross-cutting concerns such as authentication, logging, error handling, and more.

Example of authentication middleware-

```
const isAuthenticated = (req, res, next) => {

  if (req.user && req.user.isAuthenticated) {
    next();       You, now • Uncommitted changes
  } else {

    res.status(401).json({ message: "Unauthorized" });
  }
};


app.get("/posts", isAuthenticated, (req, res) => {

  const post = {
    title: "Sample Post",
    content: "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
  };
  res.json(post);
});
```

Answer 3.

https://github.com/cybermonk01/neuron-projects/tree/master/blog_crud

Answer 4.

Authentication:

Authentication is the process of verifying the identity of a user or entity. It ensures that the claimed identity is valid and corresponds to the actual user or entity. Authentication answers the question, "Who are you?"

In the context of web applications, authentication involves validating the credentials provided by a user during the login process. This could be a username and password, a token, or other forms of identification. The purpose of authentication is to ensure that the user is who they claim to be before granting access to protected resources or functionalities.

Authorization:
Authorization, on the other hand, is the process of granting or denying access to specific resources or functionalities based on the authenticated user's privileges. It defines what actions or operations a user is allowed to perform within the system. Authorization answers the question, "What are you allowed to do?"

Once a user has been authenticated, authorization determines the permissions and privileges associated with that user. These permissions may be based on roles, user groups, or specific access control rules. Authorization ensures that authenticated users can only access or modify the resources and functionalities that they are authorized to, while denying access to restricted areas.

In summary, authentication verifies the identity of a user or entity, while authorization determines what actions and resources a user is allowed to access based on their authenticated identity. Both authentication and authorization are essential components of a secure system, working together to ensure appropriate access control and protect sensitive information.

Answer 5.

The difference between EJS (Embedded JavaScript) and CommonJS (CJS) modules lies in their purpose and usage:

EJS (Embedded JavaScript):

EJS is a templating language used primarily for generating HTML markup with JavaScript. It allows you to embed JavaScript code within HTML templates to create dynamic content.
EJS is not a module system but rather a template engine. It is often used in server-side frameworks like Node.js to render dynamic web pages by combining data and templates.
EJS templates can be used to generate HTML on the server-side and send it to the client for rendering.
CommonJS (CJS) Modules:

CommonJS is a module system used primarily in server-side JavaScript environments like Node.js. It provides a way to organize and modularize code into separate files, allowing for better code organization and reusability.

CommonJS modules use the require() function to import modules and the module.exports or exports object to export values from a module.
CommonJS modules are used for sharing code between different files or modules in a program, allowing for the separation of concerns and encapsulation of functionality.
In summary, EJS is a templating language used for generating dynamic HTML content, while CommonJS is a module system used for organizing and sharing code in server-side JavaScript environments. They serve different purposes and are used in different contexts within a web application

Answer 6.

JWT (JSON Web Token) is an open standard for securely transmitting information between two parties as a JSON object. It consists of three parts: a header, a payload, and a signature. The header specifies the token type and the signing algorithm used. The payload contains claims or statements about the entity, such as user information or authorization data. The signature is created using the encoded header, encoded payload, and a secret key, and is used to verify the integrity of the token. JWTs are commonly used for authentication and authorization in web applications.

Jwt auth -
https://github.com/cybermonk01/jwt_auth

Answer 7.

When storing a user's password in a database, it is crucial to follow security best practices to protect the user's sensitive information. Here are the steps to take before storing a password:

Hashing:

Hashing is the process of converting a password into a fixed-length string of characters using a cryptographic hash function. It is a one-way process, meaning the original password cannot be derived from the hash value.

Salting:

Salt is a randomly generated value that is added to the password before hashing. It ensures that even if two users have the same password, their hash values will be different due to the unique salt.

Answer 8.

The event loop in Node.js is a mechanism that enables non-blocking, asynchronous processing. It allows Node.js to handle concurrent operations efficiently. The event loop continuously checks an event queue for pending events and dispatches them to the appropriate event handlers. This non-blocking execution enables Node.js to remain responsive even during long-running or blocking operations, making it ideal for building scalable and high-performance applications.

Answer 9.

https://github.com/cybermonk01/Full_Stack_Ecommerce
Deployed Link-
https://www.onlinecanadianmedpharmacy.com/