**Promises** -

1. Explain what a JavaScript Promise is and describe its primary purpose.

Answer) A promise is a javascript object A JavaScript Promise is an object that helps manage asynchronous operations.

It is used to find out if the asynchronous operation is successfully completed or not.

1. Describe the states of a Promise and what each state signifies.

Answer) A promise may have one of three states.

* Pending: A promise starts in a pending state  That means the process is not complete
* Fulfilled: If the operation is successful, the process ends in a fulfilled state.
* Rejected: if an error occurs, the process ends in a rejected state.

**API** -

1. Explain what an API (Application Programming Interface) is in the context of web development and how JavaScript interacts with it.

Answer) API is often used for enabling front end part to communicate with the backend part. This interaction is typically done over HTTP/HTTPS, using various data formats like JSON or XML.

JavaScript, running in the browser, interacts with APIs to fetch or send data to a server. This is commonly done using several methods:

XMLHttpRequest, Fetch API, AJAX.

1. Describe the difference between synchronous and asynchronous API calls in JavaScript. Why are asynchronous calls generally preferred?

Answer) When a synchronous API call is made, the JavaScript engine waits for the operation to complete before moving on to the next line of code.

Asynchronous calls do not block the execution of code. When an asynchronous API call is made, the JavaScript engine can continue executing the rest of the code while waiting for the operation to complete.

In asynchronous calls we use async and await. For async functions we use await.

Async functions implicitly return promises, while Await pauses the execution until the promise is resolved.

1. What is the Fetch API in JavaScript? Provide an example of how to make a GET request using the Fetch API and explain the code.

The Fetch API allows you to access APIs and perform a network request using standard request methods such as GET, POST, PUT, PATCH, and DELETE. The Fetch API returns a promise, so you need to chain the function call with . then() and . catch() methods, or use the async/await syntax.

function fetchData() {

fetch('https://api.example.com/data')

.then(response => {

if (!response.ok) {

throw new Error('Network response was not ok ' + response.statusText);

}

return response.json();

})

.then(data => {

console.log(data);

})

.catch(error => {

console.error('There was a problem with the fetch operation:', error);

});

}

fetchData();

**useEffect** -

1. Explain the purpose of the useEffect hook in React.

The useEffect hook in React is used to perform side effects in functional components. such as fetching data from an API, modifying the DOM directly, or setting up subscriptions or timers.

Purpose of useEffect:

Data Fetching, Updating the DOM.

1. Describe the syntax of the useEffect hook and explain how it can be used to perform side effects in functional components. Provide an example to illustrate your explanation. The useEffect hook takes two arguments:

**Effect Function:** A function that contains the code for the side effect.

**Dependency Array (optional):** An array of dependencies that determine when the effect should re-run.

**Syntax:**

useEffect(effectFunction, [dependencies]);

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

function DataFetchingComponent() {

const [data, setData] = useState(null);

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

useEffect(() => {

const fetchData = async () => {

try {

const response = await fetch('https://api.example.com/data');

if (!response.ok) {

throw new Error('Network response was not ok ' + response.statusText);

}

const result = await response.json();

setData(result);

setLoading(false);

} catch (error) {

setError(error);

setLoading(false);

}

};

fetchData();

return () => {

console.log('Cleanup if necessary');

};

}, []);

1. How can you control when the useEffect hook runs? Discuss the role of the dependency array and provide examples of different use cases.

The dependency array ([]) is the second argument of the useEffect hook. It allows you to specify dependencies that the effect relies on. React compares the current values of these dependencies with their previous values on each render. If any of the dependencies have changed, React re-invokes the effect function. If the array is empty ([]), the effect runs only once after the initial render.

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

function ExampleComponent() {

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

useEffect(() => {

console.log('Effect ran');

document.title = `You clicked ${count} times`;

}, [count]); // Dependency array with 'count'

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>Click me</button>

</div>

);

}

export default ExampleComponent;

**useState** -

1. Explain what the useState hook is in React and its primary purpose in functional components.

Answer) In React, the useState hook is a function that allows functional components to manage state without needing to use class component.

Purpose:

State Management:

it enables functional components to have state variables. Each state variable can hold a piece of data that may change over time, triggering re-renders of the component when updated

1. Describe the syntax of the useState hook and explain the meaning of its return values.

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

**state:** This is the current state value, which is initially set to initialState.

**setState:** This is a function that allows you to update the state. When setState is called, React schedules a re-render of the component with the updated state value.

**initialState:** This is the initial value of the state.

1. How can you initialize state with the useState hook? Provide an example with a detailed explanation.

Importing useState, Initializing State, Updating State, Rendering.

Example:

import React, { useState } from 'react';

function Counter() {

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

const incrementCount = () => {

setCount(count + 1);

};

return (

<div>

<p>Count: {count}</p>

<button onClick={incrementCount}>Increment</button>

</div>

);

}

export default Counter;

1. What is the significance of the setter function returned by the useState hook? Illustrate with an example how you can update the state.

The setter function received from useState allows you to update the state variable declared with useState. It ensures that React knows when state changes occur, triggering re-renders of components that depend on that state.

Example:

import React, { useState } from 'react';

function Counter() {

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

const incrementCount = () => {

setCount(count + 1);

};

const resetCount = () => {

setCount(0);

};

return (

<div>

<p>Count: {count}</p>

<button onClick={incrementCount}>Increment</button>

<button onClick={resetCount}>Reset</button>

</div>

);

}

export default Counter;

**Files in React app** -

1. Explain the typical structure of a React project. What are the main directories and files you would expect to find, and what is the purpose of each?

public/ Directory: Contains static assets and the HTML file where your React application is rendered.

**Main Files:**

* index.html: The main HTML file that serves as the entry point for your React application.
* favicon.ico: Icon displayed in the browser tab.
* Other static assets like images, fonts, etc.

src/ Directory:

Purpose: Contains all the source code for your React application.

Main Subdirectories:

assets/: Holds images, stylesheets, or other static assets used in the application.

components/: Reusable UI components used throughout the application.

pages/: Components representing different pages or views of the application.

services/: Handles API calls or other data-fetching services.

utils/: Utility functions or helper modules used across the application.

App.js and index.js:

App.js: The root component of your React application, where components are composed to define the overall structure.

index.js: The entry point where React renders your application into the DOM. It typically imports App.js and wraps it with other necessary providers (like Redux or Context API) before rendering.

1. Discuss the role of configuration files such as package.json

Configuration files like package.json play a crucial role in managing a React project, providing essential metadata, scripts, dependencies, and configurations