Promises -

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

   → Promise in javascript is an object which shows the eventual completion or failure of the asynchronous operation and resulting values.It allows you to associate handlers with asynchronous actions with success value or failure reason.

2. Describe the states of a Promise and what each state signifies.
   →There are three states associated with the promises:
   a).Fulfilled-> If the operation is getting fulfilled successfully.
   b).Rejected->If the operation is getting denied to complete.
   c).Pending-> If the operation is in the state where it needs to either get fulfilled or rejected but hasn't happened yet.
   Methods used here to make these process completed are →
   a. .then ->at the time of fulfilled or rejected
   b. .catch ->at the time of rejected
   c. .finally -> it will happen when the promises are settled regardless of the outcome.

APIs-

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

→ API means Application Programming Interface and it is a set of rules which is used to communicate with the different software applications.It defines the method and data formats that application can make request and then exchange the info.

a). Api enable communication between client and server with the help of HTTP/HTTPS Using REST (Representational State Transfer) of with the help of SOAP(simple object Access protocol).
b). APIs expose the endpoints which are specific URLs representing various resources Or actions.
c). At the last the data is exchanged using the JSON format or XML.

Javascript interacts with the API with the HTTP request which enabling the web applications to dynamically fetch and send data without reloading the page.This is done using the "fetch".

2. Describe the difference between synchronous and asynchronous API calls in JavaScript. Why are asynchronous calls generally preferred?
→ Synchronous is done in a sequence means every steps needs to get completed before the next step starts. Execution of the program is paused to get the operation completed before moving onto the next line of the code.

Characteristics of it is blocking of the codes and simpler to read .

→Asynchronous don’t block the execution of the program, the lines of code get executed waiting for the response from the api.
Characteristics are non blocking of the code and concurrency in the asynchronous operation which hence improve the performance and responsiveness.

Asynchronous calls are generally preferred because of :
A→ Improved User Experience.
B→ Performance
C→ Scalability
D→ Avoiding blocking of the code.

3. 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.

It is a javascript interface which makes the network request similar to XHR. However it is a powerful and flexible user set using the promises for the asynchronous operations.
Features of Fetch API
a-> Promise-based
b-> Readability
c->Modern Standards

Example :-->

```
const apiUrl = 'https://api.example.com/data';
fetch(apiUrl)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('There was a problem with the fetch operation:', error);
  });
```

Benefits ->
a->Simpler syntax
b->Better handling of the JSON
c->Modern Features

useEffect -

1. Explain the purpose of the useEffect hook in React.
   → To perform the sideEffects on the functional components. The effects include data fetching , then updating the DOM ,setting up subscriptions then cleaning up resources.
2. 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.
   →
   Syntax is->

```
useEffect(() => {
  // Side effect code here
  return () => {
    // Optional cleanup code here
  };
}, [dependencies]);
```

The parameters here are useEffefct and Dependency Array and it is optional.

The example of the useEffect is→

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

function DataFetchingComponent() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => {
        setData(data);
        setLoading(false);
      })
      .catch(error => {
        console.error('Error fetching data:', error);
        setLoading(false);
      });

    return () => {
      console.log('Cleanup on unmount');
    };
  }, []);
```

```
  return (
    <div>
      {loading ? 'Loading...' : <pre>{JSON.stringify(data, null, 2)}</pre>}
    </div>
  );
}

export default DataFetchingComponent;
```

3. 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 which is passed in the second argument to useffect , and it specifies the values that effect depends on.
   React compares these dependencies between renders and only runs whenever any of them may have changed.
   Role of dependency array:->
   A→ No dependency array→runs after every render.

   syntax:->

```
    useEffect(() => {
  console.log('Effect runs after every render');
});
```

   B→ Empty Dependency Array:--> runs only once after initial render similar to componentdidmount in class components.

   syntax:->

```
useEffect(() => {
  console.log('Effect runs once after initial render');
}, []);
```

   C.-->Specified Dependencies:-> runs after initial render and then whenever any changes done in those parameters of the arrays or can say any dependencies change in an array.

   syntax:->

```
const [count, setCount] = useState(0);
useEffect(() => {
  console.log('Effect runs after "count" changes:', count);
}, [count]);
```

1. Fetching data once when on mount

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

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

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.error('Error:', error));
  }, []); // Empty array ensures this runs once after initial render

  return (
    <div>
      {data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}
    </div>
  );
}
```

2. Running when specific value changes

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

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

  useEffect(() => {
    console.log(`Count has changed: ${count}`);
  }, [count]); // Runs only when "count" changes

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

3. Cleaning up after an effect.

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

function WindowResizeComponent() {
  const [windowWidth, setWindowWidth] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => setWindowWidth(window.innerWidth);
    window.addEventListener('resize', handleResize);

    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []); // Empty array ensures the effect runs once and the cleanup runs on
unmount

  return <div>Window width: {windowWidth}</div>;
}
```

4. Effect with multiple dependencies

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

function MultipleDependenciesComponent() {
  const [count, setCount] = useState(0);
  const [text, setText] = useState('');

  useEffect(() => {
    console.log('Effect runs when either "count" or "text" changes');
  }, [count, text]); // Runs when either "count" or "text" changes

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <input
        type="text"
        value={text}
        onChange={e => setText(e.target.value)}
      />
    </div>
  );
}
```

useState -

1. Explain what the useState hook is in React and its primary purpose in functional components.
   →
   UseState hook is a fundamental of the react's hooks API. It allows the functional components to have state which was previously possible in class components only. It enables the functional components to store and update the local component state.

   Syntax is :-->
   const [state, setState] = useState(initialState);

   example:->

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

function Counter() {
  // Declare a state variable named "count" with an initial value of 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      {/* Update the state when the button is clicked */}
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

Advanced usage→ Using state as objects

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

function UserProfile() {
  const [user, setUser] = useState({ name: ' ', age: 0 });

  const updateName = (newName) => {
    setUser(prevUser => ({
      ...prevUser,
      name: newName,
    }));
  };

  return (
    <div>
      <p>Name: {user.name}</p>
      <p>Age: {user.age}</p>
      <button onClick={() => updateName('Alice')}>Set Name to Alice</button>
    </div>
  );
}

export default UserProfile;
```

2. Describe the syntax of the useState hook and explain the meaning of its return values.
   →
   Syntax
   const [state, setState] = useState(initialState);

   example:-->

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

function Counter() {
  // Declaring a state variable 'count' initialized to 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

3. How can you initialize a state with the useState hook? Provide an example with a detailed explanation.
   →
   In functional component useState is use to initialize the state.
   This hook declare the state variable and update them with functional components.

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

function Counter() {
  // Initializing state with useState
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

   Benefits of useState Initialization
   a)simplicity
   b)Functional components
   c)Encapsulation->state variable declared with useState are scoped to the component , encapsulating state logic with it.

   Advanced Initialization

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

function FormComponent() {
  // Initializing state with an object
  const [formData, setFormData] = useState({
    username: '',
    email: '',
    password: ''
  });

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData(prevState => ({
      ...prevState,
```

```
      [name]: value
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log('Form data:', formData);
    // Further logic for form submission
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="username"
        value={formData.username}
        onChange={handleInputChange}
        placeholder="Username"
      />
      <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleInputChange}
        placeholder="Email"
      />
      <input
        type="password"
        name="password"
        value={formData.password}
        onChange={handleInputChange}
        placeholder="Password"
      />
      <button type="submit">Submit</button>
    </form>
  );
}

export default FormComponent;
```

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

   When you declare a state variable using useState, React initializes it with the provided initial state value. Alongside this state variable, useState returns a setter function. This setter function is responsible for updating the state to a new value when called. React automatically re-renders the component with the updated state value, ensuring that the UI reflects the most current state of the application.
   Example

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

function Counter() {
  // Initializing state with useState
  const [count, setCount] = useState(0);

  const incrementCount = () => {
    // Updating the count state using the setter function
    setCount(count + 1);
  };

  const resetCount = () => {
    // Setting the count state back to 0
    setCount(0);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={incrementCount}>
        Increment
      </button>
      <button onClick={resetCount}>
        Reset
      </button>
    </div>
  );
}
export default Counter;
```

   By using the functional updates also we can update the state using the setter function

```
setCount(prevCount => prevCount + 1);
```

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?

```
my-react-app/
→
├── public/
│   ├── index.html
│   ├── favicon.ico
│   └── ...
│
├── src/
│   ├── index.js
│   ├── App.js
│   ├── components/
│   │   ├── Header/
│   │   │   ├── Header.js
│   │   │   ├── Header.css
│   │   │   └── ...
│   │   ├── Footer/
│   │   │   ├── Footer.js
│   │   │   ├── Footer.css
│   │   │   └── ...
│   │   └── ...
│   │
│   ├── pages/
│   │   ├── Home/
│   │   │   ├── Home.js
│   │   │   ├── Home.css
│   │   │   └── ...
│   │   ├── About/
│   │   │   ├── About.js
│   │   │   ├── About.css
│   │   │   └── ...
│   │   └── ...
│   │
│   ├── styles/
│   │   ├── index.css
│   │
│   ├── assets/
│   │   ├── images/
│   │   ├── icons/
│   │   └── ...
│   │
│   ├── services/
```

```
│   │   ├── api.js (API service)
│   │   ├── authService.js (Authentication service)
│   │   └── ...
│   │
│   ├── utils/
│   │   ├── helpers.js (Utility functions)
│   │   ├── constants.js (Constants)
│   │   └── ...
│   │
│   └── index.css (global styles)
│
├── tests/
│   ├── App.test.js
│   ├── components/
│   │   ├── Header.test.js
│   │   ├── Footer.test.js
│   │   └── ...
│   └── ...
│
├── .gitignore
├── package.json
├── package-lock.json or yarn.lock
├── babel.config.js or .babelrc
└── webpack.config.js (if custom configuration)
```

2. Discuss the role of configuration files such as package.json.
   →
   a)Managing Directories.
   →One of the primary functions of package.json is to manage project dependencies. Dependencies are external libraries or modules required for your application to function correctly. These can include frameworks like React, utility libraries, or even tools for development and testing.
   b)Defining the project metadata
   c) Scripts for automation
   d)Managing developer dependencies
   e)Scripts Execution Context

   Hence,
   package.json is central to managing a JavaScript project's dependencies, defining metadata, scripting automation, and organizing developer tools. It standardizes project setup across different environments, enhances collaboration among team members, and ensures consistent build and deployment processes. Understanding and effectively

managing package.json is crucial for maintaining a well-structured and reliable React application.