**Q1. Explain the key features of React**

1. Easy creation of dynamic applications: React makes it easier to create dynamic web applications because it requires less coding and offers more functionality, as opposed to JavaScript, where coding often gets complex very quickly.
2. Improved performance: React uses Virtual DOM, thereby creating web applications faster. Virtual DOM compares the components’ previous states and updates only the items in the Real DOM that were changed, instead of updating all of the components again, as conventional web applications do.
3. Reusable components: Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their logic and controls, and they can be reused throughout the application, which in turn dramatically reduces the application’s development time.
4. Unidirectional data flow: React follows a unidirectional data flow. This means that when designing a React app, developers often nest child components within parent components. Since the data flows in a single direction, it becomes easier to debug errors and know where a problem occurs in an application at the moment in question.
5. Small learning curve: React is easy to learn, as it mostly combines [basic HTML](https://www.simplilearn.com/tutorials/html-tutorial/what-is-html) and JavaScript concepts with some beneficial additions. Still, as is the case with other tools and frameworks, you have to spend some time to get a proper understanding of React’s library.
6. It can be used for the development of both web and mobile apps: We already know that React is used for the development of [web applications](https://www.simplilearn.com/progressive-web-apps-article), but that’s not all it can do. There is a framework called React Native, derived from React itself, that is hugely popular and is used for creating beautiful mobile applications. So, in reality, React can be used for making both web and mobile applications.
7. Dedicated tools for easy debugging: Facebook has released a Chrome extension that can be used to debug React applications. This makes the process of debugging React web applications faster and easier.

**Q2. What problem does React solve in web development?**

Traditionally, web development involved manipulating the DOM directly with JavaScript. This approach can be cumbersome, error-prone, and difficult to scale.

React solves these problems by providing a more efficient and organized way to build UIs.

**Q3. Is React a JavaScript library or a framework?**

React is a library not a framework.

**Q4. How does React make it easier to build user interfaces?**

React gives some extra features whit the help of this we can easily build user interface.

Some Features are:

JSX

Component Base

Reuseable Component

UniDirectional Data Flow.

**Q5.** **List the steps to set up a basic React development environment.**

1. **Installing Node.js and NPM**

First, you need to have Node.js and NPM (Node Package Manager) installed on your machine. You can download the latest version of Node.js from the [official website](https://nodejs.org/en/). The installation package includes both Node.js and NPM.

After installing Node.js, open your terminal or command prompt and verify the installation by running the following commands:

npm -v and node -v

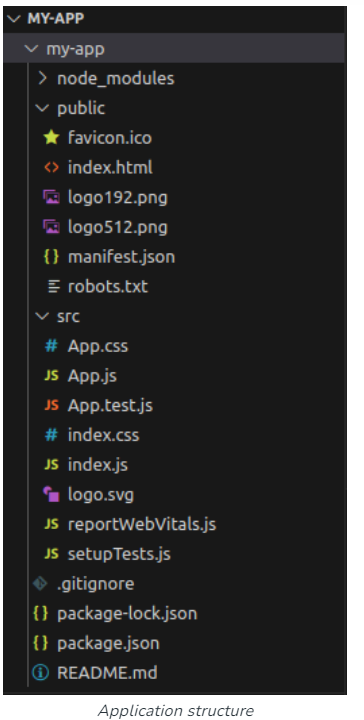
1. **Setting up a new React project with Create React App**

npm install -g create-react-app

Once the installation is complete, create a new React project by running:

create-react-app my-react-app

1. **Exploring the project structure**



1. **Running the development server**

To start the development server, run the following command:

npm start

1. **Customizing your development environment**

**Q6. What is Babel, and why is it used in React development?**

Babel is a very famous **transpiler** that basically allows us to use future JavaScript in today’s browsers.

In simple words, it can convert the latest version of JavaScript code into the one that the browser understands.

Transpiler is a tool that is used to convert source code into another source code that is of the same level.

The latest standard version that JavaScript follows is ES2020 which is not fully supported by all browsers hence we make use of a tool such as ‘babel’ so that we can convert it into the code that today’s browser understands.

**Use Of Babel**

We use Babel with React to transpile the JSX code into simple React functions that can be understood by browsers.

Using this way we can assure that our JSX code can work in almost any browser. This combination is widely used in modern-day web development.

**Q7. Write a JSX code to render a simple div element.**

import React from 'react';

function Example() {

  return (

    <>

      <div>Simple JSX which render the div element</div>

    </>

  );

}

export default Example;

**Q8.** **What is the key difference between JSX and HTML when it comes to element attributes**

The basic differences between JSX and HTML are as follows:

|  |  |
| --- | --- |
| HTML | JSX |
| In HTML, multiple elements can be returned. For example: *<ul>* *<li>unordered list* *<ol>* *<li>ordered list</li>* *<li>ordered list</li>* *<li>ordered list</li>* *</ol>* *</li>* *<li>unordered list</li>* *<li>unordered list</li>* *</ul>* | Nested JSX must return one element, which we’ll call a parent element that wraps all other levels of nested elements: *<div>* *<p>pink</p>* *<p>yellow</p>* *<p>green</p>* *</div>* Without the wrapper element, JSX won’t transpile. In React, we can render JSX directly into HTML DOM using React rendering API, aka *ReactDOM*. The formula for rendering React elements seems like this: *ReactDOM.render(componentToRender, targetNode)* *ReactDOM.render()* must be called after the JSX elements declarations. |
| HTML elements have attributes.  e.g., *maxlength* in *<input maxlength=”16″ />* | JSX elements have props.  e.g., *maxLength* in *<input maxLength=”16″ />* |
| It is not necessary to use camelCase for attributes, ids and event references. Its totally your call to use camelCase, lowercase or hyphens for naming them. | All HTML attributes and event references in JSX become camelCase, this way, *onclick* event becomes*onClick* and *onchange* — *onChange.* |
| The class attribute can be used on any HTML element. The class name can be used by CSS and JavaScript to perform certain tasks for elements with the specified class name. | You can’t use the word *class* to define HTML classes, since *class* is a reserved word in JavaScript, instead, use — *className*. |
| In HTML almost all tags have an opening and a closing tag except probably a few like  *<br/>* | In JSX, however, any element can be written as a self-closing tag, for example:*<div/>* Example: *const string = <img src={user.avatarUrl}  />;* |

**Q9. Write JSX code to render a element with the text "Hello, React!"**

import React from 'react';

function App() {

  return (

    <>

      <h1>"Hello React!"</h1>

    </>

  );

}

export default App;

**Q10. Explain how you can apply CSS classes to JSX elements?**

To apply css classes to JSX element first we have to give class name.

Instead of using class we use className in React.

import React from 'react'

import ‘./App.css’;

const Home = () => {

  return (

    <>

      <div className="main">

        Hello how are you

      </div>

    </>

  )

}

export default Home;

**App.css File**

.main{

  color:red;

  font-family: 'Courier New', Courier, monospace;

  font-size: large;

}

Now we have some other ways to use css in react.

* Inline Css

To style an element with the inline style attribute, the value must be a JavaScript object:

class MyHeader extends React.Component {

  render() {

    return (

      <div>

      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>

      <p>Add a little style!</p>

      </div>

    );

  }

}

* Using JavaScript Object

You can also create an object with styling information, and refer to it in the style attribute:

class MyHeader extends React.Component {

  render() {

    const mystyle = { // Javascript object

      color: "white",

      backgroundColor: "DodgerBlue",

      padding: "10px",

      fontFamily: "Arial"

    };

    return (

      <div>

      <h1 style={mystyle}>Hello Style!</h1> // use that object as a style

      <p>Add a little style!</p>

      </div>

    );

  }

}

**Q11.** **Add conditional rendering to a JSX component, so it displays one message if a variable is true and another message if it's false.**

import React from 'react'

const Home = () => {

  const val = true;

  return (

    <>

    {

      val ? "Print Trure" : "Print False"

    }

    </>

  )

}

export default Home;

**Q12. Develop a class-based React component called Counter that displays a number and has two buttons, one for incrementing the number and one for decrementing it.**

class Counter extends React.Component{

    constructor(props){

        super(props);

        this.state = {

            count : 0

        };

    };

    render(){

        return (

            <>

            <h1>Counter</h1>

            <button onClick={()=> this.setState({count:this.state.count - 1})}>-</button>

            <p>Count Value {this.state.count}</p>

            <button onClick={()=> this.setState({count: this.state.count + 1})}>+</button>

            </>

        )

    }

}

ReactDOM.render(<Counter />, document.getElementById('app'))

Q13**. Implement a functional component called "Person" that receives a "name" prop and displays "Hello, [name]."**

For that we have to create 2 components

**App.jsx**

import React from 'react';

import Name from './Name'; // Import another componet

const App = () => {

    const name = "Abdul Rahman";

  return (

    <div>

        <Name name={name} /> // pass the value to Name component

    </div>

  )

}

export default App

Name.jsx

import React from 'react'

const Name = (props) => { // props as an argument

  return (

    <div>Hello {props.name}</div> // get the value which comes from App comp

  )

}

export default Name

Q14. **Describe a scenario in which you would use state to manage data within a React component. Provide an example use case**

import React, { useState } from 'react'

const Home = () => {

  const [val, setVal] = useState(0); // define hooke to set the initial val

// define onClick functon in which we manage the state and increment the value by 1

  const handleCountValue = ()=>{

    setVal(val + 1); // set the value it will update val variable of usestate

  }

  return (

    <>

      <h1>Counter {val}</h1>

      <button onClick={handleCountValue}>+</button>

    </>

  )

}

export default Home;

**Q15. Construct a component called "ToDoList" that takes an array of tasks as a prop and renders an unordered list of tasks. Ensure each list item has a unique key**

App.jsx

function App() {

// Array of task

 const initialValue = [

   "Star Wars", "Return of the Jedi", "Empire Strikes Back",

 ]

 const [ movies, setMovies ] = useState(initialValue);

const deleteItems = (id)=>{

setNewItem((oldItem)=>{

return oldItem.filter((arrElm, index)=>{

return index !== id;

})

})

}

 return (

   <div>

     {movies.map((movie,ind) => {

       return(

<Item

key={ind}

id={ind}

itemVal = {movie}

select = {deleteMovie} />;

)

     })}

   </div>

 );

}

export default App;

**Item.jsx**

import React from 'react';

const Item = (props) => {

  return (

    <>

    <div className="todo\_style">

    <i className="ri-delete-bin-6-line" onClick={()=> {props.select(props.id)}}></i>

    <li>{props.itemVal}</li>

    </div>

    </>

  )

}

export default Item

Q16. **Explain the importance of keys when rendering lists in React.**

A “key” is a special string attribute you need to include when creating lists of elements in React.

Keys are used in React to identify which items in the list are changed, updated, or deleted.

Keys are used to give an identity to the elements in the lists.

It is recommended to use a string as a key that uniquely identifies the items in the list.

Q17. **Create a React component called ShoppingList that receives an array of shopping items as a prop and renders an ordered list (<ol>) with each item as a list item (<li>).**

MyPage.jsx

import React, { useState } from 'react'

import ShopList from './ShopList'

const MyPage = () => {

    // create the useState and store the list of item in state in array format

    const [item, setitem] = useState([

        {

            id: 1,

            name:"Suger",

            price: 40

        },

        {

            id: 2,

            name:"Tea",

            price: 20

        }

    ])

  return (

    <>

        {/\*pass the item to ShopList component as a props  \*/}

        <ShopList list={item} />

    </>

  )

}

export default MyPage

ShopList.jsx

import React, { useState } from 'react'

const ShopList = (props) => {        // get the item value using props

    const [listItem, setListItem] = useState(props.list) // store the value in a state

  return (

    <>

    {/\* Use the order list to display the item \*/}

        <ol>

            {

                // use of the map function to map all the item value which comes from other component

                listItem.map((val)=>{

                    return (

                        <>

                            <li key={val.id}>{val.name} {val.price}</li>   {/\* display all the item \*/}

                        </>

                    )

                })

            }

        </ol>

    </>

  )

}

export default ShopList

Q18. **What are React events, and how are they similar or different from standard HTML events**

## Difference between HTML and React Event Handling :

| **In HTML** | **In ReactJS** |
| --- | --- |
| we specify event using “onclick”,”onsubmit”which is the normal convention. | We specify the event using  “onClick”,”onSubmit” likewise which is camel case convention. |
| We bind or provide the listener in form of the string. | We bind or provide the listener in form of function name or method name as a variable. |
| The string we pass to the listener should have “( )” at the end in order to work. | We are only suppose to pass the method  name and nothing else. React can determine that it has to run this method. |
| We can add event listener any time using external javascript. | We have to specify all the Events at the time of creating the component. |

Q19. **Create a React component that renders a button. Implement an event handler that logs a message to the console when the button is clicked.**

import React from 'react';

class Message extends React.Component{

    displayMessage = ()=>{

      console.log("Hello");

    }

    render(){

      return (

        <button onClick={this.displayMessage}>Message</button>

      )

    }

  }

  export default Message;

Q20. **Develop a React component that renders a list of items. Implement an event handler that allows users to remove items from the list when a "Delete" button is clicked.**

import React, { useState } from "react";

const list = ["khan", "Sahab"];

const Message = () => {

  const [listVal, setListVal] = useState(list);

  const handleRemove = (id)=> {

    let newList = list.filter((elm,ind)=>{

        console.log(elm);

        return id !== ind;

    });

    setListVal(newList);

  }

  return (

    <>

      <div>Display List</div>

      <ul>

        {listVal.map((item, id) => {

          return (

            <>

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

              <button type="button" onClick={() => handleRemove(id)}>

                Remove

              </button>

            </>

          );

        })}

      </ul>

    </>

  );

};

export default Message;

**Q21. Create a form with a checkbox input for "subscribe to newsletter." Implement a controlled component for the checkbox and display a confirmation message when the checkbox is checked.**

import React, { useState } from "react";

const Message = () => {

  const [isChecked, setIsChecked] = useState(false);

  const handleChange = ()=>{

    setIsChecked((pre)=> !pre);

  }

  return (

    <>

      <h1>Display Form</h1>

        <form >

          <input type="checkbox" checked={isChecked} onChange={handleChange}/>

            <span>subscribe to newsletter</span>

            <p>{isChecked ? "Subscribe" : "Unsubscribe"}</p>

        </form>

    </>

  );

};

export default Message;

**Q22. Implement form validation for a phone number input field. Ensure that the phone number entered is in a valid format (e.g., xxx-xxx-xxxx) and display an error message if it's not.**

import React, { useState } from "react";

const Message = () => {

  const [phone, setPhone] = useState();

  const [error, setError] = useState(false)

  const handlephone = (e)=>{

    e.preventDefault();

    let validNumber = e.target.value;

    const length = validNumber.length;

    if(length == 10){

        setPhone(e.target.value);

    }else{

        setError(true);

    }

  }

  return (

    <>

      <h1>Display Form</h1>

        <form >

            <label>Enter mobile Number</label> <br />

            <input type="text" value={phone} onChange={handlephone} />

            {

                error ? "<p>Invalid Number. Number must be of 10 digit</p>" :""

            }

        </form>

    </>

  );

};

export default Message;

**Q23. Implement real-time validation for an email input field. Display a green checkmark icon next to the input when the entered email is valid and a red X icon when it's invalid.**

import React, { useState } from "react";

import validator from "validator";

const Message = () => {

  const [emailError, setEmailError] = useState(false)

  const validateEmail = (e) => {

    var email = e.target.value

    if (validator.isEmail(email)) {

      setEmailError(true)

    }

  }

  return (

    <div style={{

      margin: 'auto',

      marginLeft: '300px',

    }}>

      <pre>

        <h2>Validating Email in ReactJS</h2>

        <span>Enter Email: </span>

        <input type="text" id="userEmail"

        onChange={(e) => validateEmail(e)}></input> <br />

        <span style={{

          fontWeight: 'bold',

          color: emailError ? 'green' : 'red',

        }}>{emailError ? 'Valid Email :)' : "Enter valid Email!"}</span>

      </pre>

    </div>

  );

};

export default Message;

**Q24. Create a simple React application that uses the Context API to manage the theme of the application (e.g., light and dark mode). Allow users to toggle between the two themes using a button. Use the useContext hook to access and apply the theme context.**

**ThemeChange.js**

import { createContext, useState } from "react";     // import useState and createContext

export const ThemeChange = createContext("");   // use the createContext

export const ThemeChangeProvider = ({children})=>{      // Create the Provider

    const [Theme, setTheme] = useState(false);  // set the initial state value using useState

    console.log(Theme);

    return (

        // Provide the state value

        <ThemeChange.Provider value={{Theme,setTheme}}>

                     {children}

        </ThemeChange.Provider>

    )

}

**App.jsx**

import Home from "./component/Home";

import { ThemeChangeProvider } from "./context/ThemeChange";

function App() {

  return (

    <>

    {/\* Wrape the ThemeProvider to component with this we can access the Context value everyWhere \*/}

    <ThemeChangeProvider>

        <Home />

    </ThemeChangeProvider>

    </>

  );

}

export default App;

**Home.jsx**

import React, {useContext} from 'react';  // import useContext

import { ThemeChange } from '../context/ThemeChange'; // import ThemeChange from context

import '../App.css'

const Home = () => {

    const {Theme, setTheme} = useContext(ThemeChange);  // Access the Context state value using useContext

    // Define the method on click of button theme change

    const handleTheme = ()=>{

        setTheme((pre)=> !pre);    // change the state value using update function of useState

    }

  return (

    // set the css using ternory operatior

    <div className={Theme ? "dark" : 'light'}>

        <button onClick={handleTheme}>Change Theme</button>

    </div>

  )

}

export default Home

**App.css**

\*{

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

body{

  width: 100%;

  height: 100vh;

}

.dark{

  width: 100%;

  height: 100vh;

  background-color: rgba(0, 0, 0, 0.178);

}

.light{

  width: 100%;

  height: 100vh;

  background-color: rgba(255, 255, 255, 0.178);

}

**Q25. Explain the role of the useContext hook in React. How does it enable components to access values from the context**

The useContext hook is used to consume values from a React context.

Context provides a way to pass data through the component tree without having to pass props manually at every level.

To determine the context value, React searches the component tree and finds the closest context provider above for that particular context.

**Q26.** **What is a custom hook in React, and why would you use one in your application**

We know that hooks like useState, and useEffect are reusable components. Sometimes we make components that we have to reuse again and again in the application. In this case, we can convert the component to hooks by extracting logic from it.

**Need for Custom Hooks**

The main reason why you should be using Custom hooks is to maintain the concept of [DRY](https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/)(Don’t Repeat Yourself) in your React apps.

**Building a custom hook**

Creating a custom hook is the same as creating a JavaScript function whose name starts with “use”. It can use other hooks inside it, return anything you want it to return, take anything as parameters.

**Note:** It is important to name your custom hooks starting with “use”, because without it React can’t realize that it is a custom hook and therefore can’t apply the rules of hooks to it. So, you should name it starting with “use”.

Example:

**FirstComponent.jsx**

// Filename - First Component

import React from "react";

// importing the custom hook

import useCustomHook from "./useCustomHook";

function FirstComponent(props){

    // ClickedButton = resetCounter;

    const clickedButton = useCustomHook(0 , "FirstComponent");

    return (

        <div>

            <h1> This is the First Component</h1>

            <button onClick={clickedButton}>

                Click here!

            </button>

        </div>

    );

}

export default FirstComponent;

useCustomHook.jsx

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

const useCustomHook = (initialVal, componentName) => {

    const [counter, setCounter] = useState(initialVal);

    const incCounterValue = ()=>{

        setCounter(counter+1);

    }

    // const decCounterValue = ()=>{

    //     setCounter(counter - 1);

    // }

    useEffect(() => {

        // Some logic that will be used in multiple components

        console.log("The button of the "

        + componentName + " is clicked "

        + counter + " times.");

    } , [counter , componentName]);

  return incCounterValue;

}

export default useCustomHook

**App.jsx**

import FirstComponent from "./component/FirstComponent";

function App() {

  return (

    <>

      <FirstComponent />

    </>

  );

}

export default App;

**Q27.** **Build a custom hook called useKeyPress that listens for a specific key press (e.g., the "Enter" key) and returns a boolean indicating whether that key is currently pressed. Use this hook in a component to trigger an action when the user presses the specified key.**

**SecondCompoent.jsx**

import React, { useState } from 'react';

import useKeyPress from './useKeyPress';

const SecondComponent = () => {

    // const [val, setVal] = useState();

    const handleKeyPress = useKeyPress();

  return (

    <>

        <div>Key Press</div>

        <button onKeyPress={(e)=> handleKeyPress(e)}>Key Press</button>

    </>

  )

}

export default SecondComponent

useKeyPress – Custom Hook

import React from 'react'

const useKeyPress = (e) => {

    const handleKey = ()=>{

        console.log(true);

    }

  return handleKey;

}

export default useKeyPress

App.jsx

import SecondComponent from "./component/SecondComponent";

function App() {

  return (

    <SecondComponent />

  );

}

export default App;

Q28.

Q29. **Build a React component that uses Styled Components to style a custom button. Create variations of the button (e.g., primary, secondary) using props and conditional styling**

