Page No.
Date

* Introduction

React is a popular javascript library for building user interfaces. It is also referred to as a front-end javascript library. It was developed by facebook and is widely used for creating dynamic and interactive web applications.

* What is React ?

React is a javaScript library for building user interfaces on the web. React is a declarative component based library that allows developers to build reusable UI components and it follows the virtual DOM approach, which optimizes rendering performance by minimizing DOM updates. React is fast and works well with other tools and libraries.

* History of react

React was invented by facebook developers who found the traditional DOM slow.

By implementing a virtual DOM, React addressed this issue and gained popularity rapidly

The current stable version of ReactJS is 18.3.1 released on June APril, 2024.

The libary continues to evolve, introducing new features with each update.

Page No.
Date

* Features of React

React is one of the most demanding Javascript libraries because it is equipped with a ton of features which makes it faster and production-ready. Below are the few features of React.

1 Component-Based Architecture

React provides the feature to break down the UI into smaller, self-contained components. Each component can have its own state and props.

2 JSX (Javascript Syntax Extension)

JSX is a syntax extension for Javascript that allows developers to write HTML like code with in their Javascript file. It makes React components more readable and expressive.

* ex

```
Const name = "G.Rathod";
Const element = <h1> Welcome to {name} </h1>;
```

3 Virtual DOM

React maintains a lightweight representation of the actual DOM in memory. When changes occur, React efficiently updates only the necessary parts of the DoM

4 One-way Data Binding

The data in react flows only in one direction ex. the data is transferred from top to bottom, from Parent components to child components.

5 Performance

React uses virtual DOM and updates only the no modified parts. So this makes the DOM to run faster DOM executes in memory so we can create seperate components which makes the DOM run faster.

6 Components

React divides the webpage into multiple components as it is component-based. Each component is a part of the UI design which has its own logic and design the component logic which is written in Javascript makes it easy and run faster and can be reusable.

7 Single Page Application (SPAs)

React is recommended in creating SPAs, allowing smooth content updates without page reloads. Its focus on reusable components makes it ideal for real-time applications

Page No.
Date

* React setup

Method 1: using create react-app

Step 1: Navigate to the folder where you want to create the project and open it in terminal

Step 2: In the terminal of the application directory type the following command

- npx create-reacte-app folder-name

Step 3: Navigate to the newly created folder using the command

- cd folder-name

Step 4: A default application will be created with the following project structure.

Step 5: To run this application type the following command in terminal

- npm start

Page No.
Date

* Props method

Props are used to pass data to React compone
we call a Child component from parents we can
pass data as Props. This helps the parent compone
communicate with the child.

Note: before use of Props we must import

- npm i prop-types
- import PropTypes from 'prop-types';

* ex APP.js

```
import './APP.CSS';
import Navbar from './component/Navbar';

function APP() {
        return
            <>
        <Navbar title="Method" />
            </>
            );
        }
```

* Navbar.js

```
import React from 'react'
import PropTypes from 'prop-types';
```

```
export default function Navbar(Props){

    return(
        <nav>
            <ul>
            <li>{props.title}</li>
            <li> About </li>
            <li> Contect </li>
            </ul>
        </nav>
    )

                                    }
```

* Props types

  There are four Props types which is given below.

  1 array
  2 string
  3 Number
  4 boolean

- We can set Prop type using given syntax
- We mean use in component file like Navbar.js

* ex

```
Navbar.PropTypes = {

        title : PropTypes.string,
    contect-no: PropTypes.number.

                    }
```

* default Props

We can set default Props using given syntax

- ex

```
Navbar.defaultProps = {
                title: "Rathod",
                contact_No: 901626,
                }
```

Page No.
Date

* usestate Hook

usestate is a special function that lets you add state to functional components. It provides a way to declare add manage state variable directly within a function component. It should be noted that one use of usestate() can only be used to one state variable.

- To import the usestate hook write the falowing code at the top level of your component

syntax: import {usestate} from "react";

- This hook takes some initial state and returns two value. The first value contains the state and the second value is a function that updates the state

syntax : const [text, setText] = useState("Enter text here")

Page No.
Date

* JSX

JSX (JavaScript XML) is a syntax for javascript look similar to XML or HTML. It's used with React to describe what the UI should look like. JSX is not mandatory for using for using React but it is highly recommended.

* Rules to write JSX

1 Always return single root Element
2 Close all the tags
3 Use camelcase convention whether possible eg: className

* Syntax

```
<>
    <title> Basic page</title>
    <h1> welcome </h1>
</>
```

Note: JSX use to write html in JavaScript

Page No.
Date

* React Router

React Router is a popular library used to handle routing in React application. React Routing is the process of navigation between diffrent pages or views within a single page application.

* Concept of React Router

1 This approach enhances the the user exceperience by providing a faster and more seamless interaction compared to traditional multi-page application.

2 Client-side routing is used to manage the navigation between views without refreshing the entire page.

3 React Router supports dynamic routing.

4 Nested routers allow you to define routes within routes.

5 The 'Navigate' component can be used within routes to handle redirection.

* import React Router

```
npm install react-router-dom
```

Note: We can use key if we mount one component with diffrent promps.

* example

APP.js

```
import React from 'react';
import {BrowserRouter as Router, Route, Switch,
Link} from 'react-router-dom';

Const Home = () => <h2> Home page </h2>;
const About = () => <h2> About page </h2>;
const contact = () => <h2> contact page </h2>;

function APP APP() {
        return (
  <Router>
    <nav>
      <ul>
    <li> <Link to="/"> Home </Link> </li>
    <li> <Link to="/About"> About </Link></li>
    <li> <Link to"/contact"> contact </Link></li>
      </ul>
    <nav>

  <Switch>
    <Route exact path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/contact" component={Contact} />
  </Switch>
   </Router>
                );
                }
export default APP;
```

Page No.
Date

* React component Lifecycle techniques

React component are the reusable and independent bits of code that help build the user interface of a webpage. Each of these components has various lifecycle methods which are invoked at different stages of their lifecycle. There are 4 stages in the lifecycle of a react component.

1 Initialization

It is the first state of a React component's lifecycle. The component is defined with the default props and it's initial state in the constructor of the component. This phase occurs only once in the lifecycle before the creation of the component or any props from the parent is passed into it and consists of the following method which are called in the same order.

- constructor()

This method is called before the component is mounted. This is only used for two purposes: Initializing local state by assigning an object to "this.state" and binding the event handler method to an instance.

- getDefaultProps()

This method can be used to define the default value of this.props. Also we need to initialize the CreateReactClass before writing anything else for using these method.

- getInitialState()

This method can be used to define the default value of this.state.

2 Mounting

It is the second of a React Component's lifecycle In this stage the component is mounted on the DOM and rendered for the first time on the webpage after the component has been initialized. It consists of the following methods which are called in the same order.

- getDerivedStateFromProps()

This method is invoked on the initial mount and any subsequent update right before calling the render method. To update the state it returns an object and to update nothing it returns null

- render()

This method is responsible for returning single root HTML node element defined in each component. When called, this method examines this.Props and this.state and returns one of the following types:

1. React element
2. Arrays and fragments
3. Portals
4. String and numbers
5. Booleans or null.

- componentDidMount()

This method is invoked immediately after the render() method. Now we can perform any querying operations with the DOM.

- componentWillMount()

This method is invoked right before the render() method. The component will not re-render if setState() is called inside this method.

3 Updating

It is the third stage of a react component's lifecycle. In this stage, the state and props are updated following various event like clicking on a button, pressing a certain key etc. We can communicate with the hierarchy of the components and handle and handle user interaction on the webpage. It consists of the following methods.

- shouldComponentUpdate()

This method is invoked before the render method when new props or states are being received and the DOM is decided to be updated by the component. The component will update if this method returns true or skip the updating if it returns false.

The code inside the render method will be invoked again if shouldComponentUpdate() return false.

- getSnapshotBeforeUpdate()

This method invoked just before the DOM is updated with the latest rendered output It helps to capture information before it is potentially changed from the DOM.

- componentDidUpdate()

This method is invoked immediately after the component is updated. we can use this method to execute a block of code once after the rendering. It is not called for the initial rendering

- componentWillUpdate()

This method is invoked when new props or states are being received just before rendering of the component. It can be used to prepare before an update occurs. If shouldComponentUpdate() return false it will not be called.

- componentWillRecieveProps()

This method is invoked before new props are received by a mounted component. To perform state transition by using the this.setState() method, we should compare this.props and nextProps if we want to update the state in response to prop changes. React only call this method if some of the component's props are updated. It doesn't call this method with initial props during the mounting stage.

4 an unmounting

This is the final phase of the lifecycle of the component which is the phase of unmounting the component from the DOM. The following function is the solo member of this phase.

1 componentWillUnmount()

This function is invoked before the component is finally unmounted from the DOM i.e. this function gets invoked once before the component is removed from the page and this denoted the end of the cycle. lifecycle.

Page No.
Date

* React Hooks

React Hooks provide functional components with the ability to use state and manage side effects. They were first introduced in React 16.8 and allow developers to hook into the state and other React features without having to write a class.

Note: Hooks can't be used with class components

* Type of React Hooks

1 State Hooks
2 Context Hooks
3 Ref Hooks
4 Effect Hooks
5 Performance Hooks
6 Resource Hooks
7 Other Hooks

1 State Hooks

State Hooks stores and provide access the information

i See useState Hook: useState Hook provides state variable with direct update access.

ii useReducer Hook: useReducer Hook provides a state variable along with the update logic in reducer function

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

2 Context HOOKS

Context hooks make it possible to access the info. without being passed as a Prop.

i useContext Hook : Share the data as a global data with passing down props in component tree

```
const context = useContext(myContext);
```

3 Ref Hooks

Refs creates the variable containing the info. not used for rendering e.g. no DOM nodes

ii useRef : Declares a reference to the DOM element mostly a DOM node

ii useImperativeHandle : It is an additional hook that declares a customizable reference

```
const textRef = useRef(NULL);
```

4 Effect Hooks

Effect connect the components and make it sync with the system. It includes change in browser DOM, networks and other libraries

i useEffect : useEffect Hook connects the components to external system.

ii useLayoutEffect : used to measure the layout fires when the screen render.

iii useInsertionEffect: used to insert the css dynamically, fires before the changes made to DOM

```
useEffect(() => {
        // block of code
    }, [dependencies])
```

5 Performance Hooks

Performance hooks are a way to skip the unnecessary work and optimise the rendering performance

i useMemo: return a memoized value value reducing unnecessary computations.

ii useCallback: return a memoized callback that changes if the dependencies are changed.

iii useTransition: enables us to specify which state changes are critical or urgent and which are not.

iv useDefferedValue: allows you to postpone the rendering of a component until a specific condition is met

```
const memoizedValue = useMemo(function, arrayDependencies)
```

function = Function that Return value

6 Resource Hooks

It allows component to access the resource without being a part of their state e.g. accessing a styling or reading a Promises.

i use: used to read the value of resources.

```
const data = use(dataPromise);
```

* Rules for using Hooks

1 Only functional components can use hooks.
2 Hooks must be imported from React
3 Calling of hooks should always be done at top level of components.
4 Hooks should not be inside conditional statement.