|  |  |
| --- | --- |
| React: is used to create dynamic and interactive website. |  |
| Main.jsx:  **ReactDOM**.createRoot(document.getElementById('root')).render(  **<React.StrictMode>**  <App />  </React.**StrictMode**>,  )  **ReactDOM:** is a react library to create app for website  To create app for mobile phone we use ReactNative  **<React.StrictMode>,** When you wrap your components with it activates additional checks and warnings for your development environment. is a component itself. | Ternary operators:  **Condition? expressionIfTrue : expressionIfFalse;**  const message = age >= 18 ? 'adult' : 'kid';  Conditional rendering with logical AND operator.  {list.length !== 0 && <p>{list.join(', ')}</p>}  **setInterval(function, intervalDuration);** |
| **Building components** |  |
| **Returning multiple elements in react:**  In react a component cannot return more than 1 element. So to return multiple elements we can use one of the 3   1. Wrap: the elements in a   **<div> place multiple elements here </div>**   1. Use Fragment import { Fragment } from react   **<Fragment> place multiple elements here </Fragment>**   1. Recommended way:   **Place <> after return** : to tell react to return multiple elements | **Using Bootstrap:**   * To use fancy drop downs   Appart from  import 'bootstrap/dist/css/bootstrap.min.css'  we should also import  **import 'bootstrap/dist/js/bootstrap.bundle.min';**   * **npm install react-bootstrap bootstrap**   **to use it:**  **import { Button, Navbar, Container, Row, Col } from 'react-bootstrap';**  **react-bootstrap**: This package contains pre-built Bootstrap components that are rewritten as React components. It allows you to use Bootstrap's UI components in React applications without the need for jQuery or Bootstrap's own JavaScript.   * **npm install react-bootstrap-icons –save** -save: will update the package.json file to add react-bootstrap-icons   Ass dependency so that it can be installed using npm |

|  |  |
| --- | --- |
| **COMPONENT LIBRARY**  **NavBar.js:**  import React from 'react';  import { Navbar, Nav, Container } from 'react-bootstrap';  const NavBar = () => {    return (      <Navbar expand="lg">        <Container>          <Navbar.Brand href="#home">            <img src={""} alt="logo" />          </Navbar.Brand>          <Navbar.Toggle aria-controls="basic-navbar-nav" />          <Navbar.Collapse id="basic-navbar-nav">            <Nav className="me-auto">              <Nav.Link href="#home">Home</Nav.Link>              <Nav.Link href="#link">Work</Nav.Link>              <Nav.Link href="#about">About</Nav.Link>              {/\* Add more Nav.Link components for additional navigation \*/}            </Nav>          </Navbar.Collapse>        </Container>      </Navbar>    );  };export default NavBar;  **The <Col> component**  **in Bootstrap's grid system helps create columns with varying widths across different screen sizes using breakpoints (xs, md, xl). For example:**   * **XS={12}: Takes full width on extra small screens (12 out of 12 columns).** * **md={6}: Occupies half the width on medium screens (6 out of 12 columns).** * **xl={7}: Takes a bit more than half the width on extra large screens (7 out of 12 columns).**   **FORM**  **Const [form, setForm] = useState(formInitial)**  **setForm({**  **...form,**  **[fieldName]: value**  **});**  [fieldName]: value syntax in JavaScript is called computed property names. It allows you to dynamically set a property key in an object based on the value of a variable  fieldName is 'firstName' and value is 'John', this code updates the form state by creating a new object that retains the existing state (...form) and updates the firstName field with the value 'John'.  **Udating form data**  **<input type="text" placeholder="First Name" name="fname" value={form.fname} onChange={ (e) => onFormUpdate('fname', e.target.value)} />**  **NB: e.target.value :**   * **always e.target.value should be used** * retrieves the current value of the input field that triggered the change. In a text input, e.target.value contains the text that the user has entered. | **Important Functions**  **Function name: setInterval**  **Syntax:** setInterval(function, intervalDuration);  **Example:**  Ex-1: setInterval (console.log(“Thomas Kitaba”), 3000 )  Ex-2: setInterval (()=> { console.log(“one second” }, 1000)  **Output:** prints Thomas Kitaba in every 3 second;  **Function name: math.random() \* 100**  **Syntax:** setInterval(function, intervalDuration);  **Example:**  setInterval( console.log(“Thomas Kitaba”), 3000 )  **Output:** prints Thomas Kitaba in every 3 second;  **React Hooks:**  The useState hook is used to add state to functional components in React  import { useState } from 'react';  syntax: **[variableName, updaterFunction] = useState(initialValue)**  **example:**  **1.**const [activelink, setactivelink] = useState('home');  2. const [intrnval, setInterval] = useStte(0)   * a component can have either of these 3 states  1. mount: when the component loads 2. update: when the components props is updated 3. unmount: when the component unloads    **[activelink, setactivelink]**: Destructuring assignment is used here to assign values returned by useState to two variables: activelink and setactivelink.   **useState('home')**: The useState hook is called with an initial state value of 'home'. This hook returns an array with two elements:   * activelink: This variable holds the current state value. In this case, it is initialized with the string 'home'. * setactivelink: This function is used to update the state. When setactivelink is called, it triggers a re-render with the updated state value.   // Use the state variable 'activelink' in your component  console.log(activelink); // Outputs: 'home'  // Update the state using the 'setactivelink' function  setactivelink('about');  console.log(activelink); // Outputs: 'about'  **useEffect**  useEffect(() => {  // Code to run when the effect is triggered  return () => {  // Cleanup code (optional) to run when the component unmounts or before the next effect  };  }, [dependencies]);  Or in other words  useEffect(() => {  console.log('Effect is triggered');  return () => {  console.log('Clean-up function is triggered');  };  }, []);   1. **Function inside useEffect**: This function contains the code you want to run when the effect is triggered. It might include setting up event listeners, fetching data, manipulating the DOM, etc. 2. **Return Function (Optional)**: This return function, if provided, is used for cleanup purposes. **It's executed when the component unmounts or before the next effect is run**. It's useful for cleaning up subscriptions, removing event listeners, or any other cleanup tasks. 3. **Dependencies (Optional)**: An array of dependencies. If provided, the effect will only re-run if any of these dependencies change between renders. If the dependencies array is empty ([]), the effect will run only once (similar to componentDidMount in class components). |
|  |  |
|  |  |
|  |  |