REACT

- **React -** React is a popular JavaScript **library** for building user interfaces (UIs) for web applications. React allows developers to create reusable UI components that can efficiently update and render changes as the application state changes. One of the key features of React is its use of a virtual DOM (Document Object Model), which is an in-memory representation of the actual HTML DOM. By using the virtual DOM, React can efficiently update only the necessary parts of the UI when the application state changes, instead of re-rendering the entire page. This approach improves performance and allows for a more responsive user experience.

    - **Advantages of React**
        - **It is composable**: Composition is a function of combining parts or elements to form a whole. With modern frameworks like React, we can divide these codes and put them in custom components. Then we can utilize these components and integrate them into one place. Hence the code becomes a lot more maintainable and flexible.
        - **Virtual DOM:** React utilizes a virtual DOM, which is a lightweight representation of the actual DOM. This approach allows React to perform efficient updates by minimizing direct manipulations to the DOM. React calculates the minimal set of changes needed and applies them to the real DOM, resulting in improved performance.

    - **Disadvantages of React**
        - **Initial loading time**
        - **JavaScript Dependent:** React relies heavily on JavaScript. If you are not familiar with JavaScript, it may require additional time to learn or work in conjunction with JavaScript concepts and libraries.

- **Dependencies** are packages or libraries required for your React application to run correctly in a production environment.

- **Dev dependencies** are packages needed during the development process, not for production builds. They include tools, testing frameworks, build scripts, and linters.

- **Lazy loading -** Lazy loading is a technique used in React.js to optimize the loading of components or resources in a web application. It allows you to load components or assets dynamically only when they are actually needed, rather than loading everything upfront. This can significantly improve the initial loading time and reduce the amount of data transferred, especially in larger applications with complex component hierarchies.

- **Suspense** - When a component is wrapped in a Suspense component, it can render a fallback UI (such as a loading spinner or a placeholder) while the data or resources it depends on are being fetched or loaded. Once the data or resources are available, the component can render the actual content.

- **JSX -** JSX (JavaScript XML) is an extension to the JavaScript language syntax used in React applications. It allows developers to write HTML-like code directly within JavaScript, making it easier to create and manipulate React elements and components.

- **Babel -** Babel is a JavaScript compiler commonly used with React.js. It allows developers to write code using the latest ECMAScript standards and React-specific syntax like JSX. Babel transforms this code into a format that can be executed by older browsers, ensuring compatibility. It also provides features such as transpiling modern JavaScript syntax, adding polyfills for missing features, and a plugin ecosystem for customization. By using Babel, developers can write modern code in React without worrying about browser support.

- **DOM -** DOM stands for Document Object Model. It is a programming interface for web documents, representing the structure and content of an HTML, XHTML, or XML document as a tree-like model. The DOM provides a way for programs to dynamically access and manipulate the elements, attributes, and text within web pages.

- **VIRTUAL DOM -** The Virtual DOM is a concept and technique used by React to optimise and efficiently update the Document Object Model (DOM) in web applications. It is an abstraction of the actual DOM and acts as a lightweight copy of it, residing in memory.

    - **Difiing -** In the context of React and the Virtual DOM, "diffing" refers to the process of comparing two versions of the Virtual DOM to identify the minimal set of changes needed to update the actual DOM. React performs this diffing process to determine the additions, removals, or updates of elements between the old and new versions of the Virtual DOM.

    - **Reconciliation Algorithm**: reconciliation refers to the process of efficiently updating the actual DOM based on changes in the virtual DOM representation.

- **Components -** Components are the building blocks of user interfaces in React. They are reusable and self-contained units that encapsulate the logic, structure, and presentation of a part of the user interface. In React, components are written as JavaScript functions or classes.
    - **Class component -** Class components are defined as ES6 classes that extend React. They have additional features, such as state management and lifecycle methods. Class components are useful when you need to maintain a state or handle more complex logic.

    - **Function component -** Function components are defined as JavaScript functions that receive input data (props) and return React elements to describe what should be rendered. Function components are simpler and more lightweight compared to class components.

- **Array destructuring -** Array destructuring is a feature in JavaScript that allows you to extract values from an array and assign them to variables in a concise and readable way. It simplifies the process of accessing individual elements or properties of an array.

- **Map -** In React, the map() function is a powerful array method used to iterate over an array and transform each element into a new array. It is commonly used in combination with JSX to generate dynamic lists or render components dynamically based on the data.

- **Filter** - In React, you can use the filter method to filter elements from an array based on a condition. The filter method creates a new array that includes only the elements that pass the provided condition.

- **Difference between JSX and HTML** - JSX is a syntactic extension to JavaScript that enables you to write HTML-like code within JavaScript. It provides more flexibility and dynamic capabilities compared to static HTML. JSX is commonly used in React applications to define the structure, appearance, and behaviour of components. JSX allows you to embed JavaScript expressions within curly braces `{}`. This allows you to dynamically generate content, compute values, or call functions inside JSX. HTML, on the other hand, doesn't support this kind of dynamic evaluation.

- **Lifecycle methods** - In React, lifecycle methods are special methods that are invoked at different stages of a component's lifecycle. They allow you to perform certain actions or logic at specific points in the component's life, such as when it is being created, updated, or removed from the DOM. Mounting Phase, Updating Phase, Unmounting Phase

- **Fragment** - In React.js, a fragment is a special component that allows you to group multiple elements together without introducing an additional DOM element. It provides a way to return multiple elements from a component without the need for a wrapper element. **Syntax - <> </>**

- **HOOKS -** Hooks are a feature introduced in React 16.8 that allows you to use state and other React features in function components without the need to write a class. They provide a way to add stateful logic and side effects to functional components, enabling them to have their own internal state and handle lifecycle events.

    - **useState -** The useState hook is a built-in hook in React that allows functional components to have their own state. It provides a way to add stateful logic to functional components without the need to use a class. The useState hook is used by calling it within a functional component and passing an initial value for the state. It returns an array with two elements: the current state value and a function to an update of that state.

    - **useEffect -** In React, the useEffect hook is a built-in hook that allows you to perform side effects in functional components. Side effects may include data fetching from an API, Cleanup Operations, subscribing to events, or manually changing the DOM.
        - **Mounting**
        - **Updating**
        - **Unmounting**

    - **useContext -** useContext is a hook that allows the component to consume data from a parent component's context without the need to pass down the data through each intermediate.

    - **useReducer** - In React, useReducer is a built-in hook that allows you to **manage** the state in a more **complex** manner using a reducer function. It is an alternative to the useState hook when the state logic becomes more advanced and involves **multiple** related values or **complex state transitions**. **The useReducer hook takes two arguments: the reducer function and the initial state value**. The reducer function is responsible for specifying how the state should be updated based on different actions. It receives the current state and an action object as arguments and returns the new state.

- **useRef** - useRef is a hook that allows you to create a mutable reference to a DOM element or a value. It's commonly used when you need to access or manipulate a specific DOM element directly. **useRef** doesn't cause re-renders when the value changes, while **useState** triggers a re-render when the state value changes.

- **useHistory vs useNavigate** - In **React Router**, both useHistory and useNavigate are hooks provided by the library for programmatic navigation between different routes or URLs. While both useHistory and useNavigate offer similar functionality for navigation, **useNavigate** is the recommended approach in React Router version 6 and provides a cleaner API.

- **useCallback** - the useCallback hook is used to **memoize a function** and prevent unnecessary re-renders of components that depend on that function. It's particularly useful when passing functions down to child components, as it helps optimise performance by preventing unnecessary re-creation of those functions.

- **useMemo** - the useMemo hook is used to **memoize the result** of a function computation. It allows you to optimise expensive calculations or computations that are repeated frequently within a component.

- **Error boundaries** in React are a feature that allows you to catch and handle errors that occur during the rendering phase or in the lifecycle methods of a component tree. They help to prevent the entire application from crashing due to a single component's error. By default, if an error

occurs during rendering, React will unmount the component tree and display an error message to the user. However, error boundaries provide a way to handle errors gracefully by capturing them within a specific component and displaying a fallback UI instead of the full crash.

- **Higher-Order Function Component** (HOC) is a pattern in React that allows you to enhance or modify the behaviour of an existing component by wrapping it with another component. HOCs are functions that take a component as an argument and return a new component.

- **Event handlers**
    - **onClick** - You can handle the onClick event using the onClick prop on a JSX element. The onClick prop takes a function that will be called when the element is clicked.

    - **onChange** - You can handle the onChange event using the onChange prop on form elements such as <input>, <select>, and <textarea>. The onChange prop takes a function that will be called when the value of the form element changes.

- **React router** - React Router enables developers to define routes in their application, map them to specific components, and handle navigation between those components. It allows for dynamic rendering of components based on the current URL or route.

- **Controlled** component is one where the form data is controlled by React state. In a controlled component, the value of the input element is controlled by React and is typically stored in the component's state. The component also provides an event handler to update the state whenever

the input value changes. This allows React to have full control over the input state.

- **Uncontrolled** component is one where the form data is handled by the DOM itself, rather than controlled by React state. In this approach, the input value is managed by the browser, and React does not track or manage its state explicitly.

- **Optional Chaining** (?.) operator accesses an object's property or calls a function. If the object accessed or function called using this operator is undefined or null, the expression short circuits and evaluates to undefined instead of throwing an error.

- **Batching** in React refers to the mechanism by which multiple **setState** calls or other updates to the component's state are grouped together and processed in a single batch or update cycle. Batching helps optimise the rendering performance by minimising the number of re-renders and reducing unnecessary work.

- **Profilers -** It measures how many times the react application renders. It helps to identify the part of the application which is slow so that the developer can optimise it for better performance. It is lightweight and no third-party dependency is required.

- **State Management Concepts**
    a. **Props** - In React, "props" is an abbreviation for "properties." Props are a mechanism for passing data from a parent component to its child components. When you create a custom component in React, you can define and specify custom attributes for that component.

These attributes are known as props. Props allow you to pass data, configuration, or functions from a parent component to its child components, enabling communication and customization between components.

b. **Context API** - The Context API is a feature in React that provides a way to share data between components without having to pass props manually through every level of the component tree. It allows you to create a global state that can be accessed and updated by components at different levels of the tree.

c. **Redux** - Redux is a **state management library** for JavaScript applications, including those built with React. It provides a predictable state container that helps manage the state of an application in a centralised and organised way. Redux follows the principles of the Flux architecture and emphasises a unidirectional data flow.
    i. **Middlewares** - Middlewares are functions that intercept and modify the behaviour of certain operations or processes. Redux middleware intercepts actions dispatched to the Redux store before they reach the reducers, allowing for additional functionality to be applied.

        1. **Thunk** refers to a technique used in Redux for handling **asynchronous** actions. Thunks are functions that can be dispatched as Redux actions. They provide a way to delay the execution of an action and perform asynchronous operations before dispatching the actual action. Thunks are typically used when you need to

make API calls, perform side effects, or handle asynchronous logic before updating the Redux state.

2. **Redux Saga -** Redux Saga is a middleware that uses generator functions to handle side effects in Redux applications. It provides a more powerful and flexible approach to handling asynchronous actions compared to Redux Thunk. Sagas run in the background, listening for specific actions and executing logic accordingly. These sagas can listen to dispatched actions, perform asynchronous operations, and dispatch new actions to update the Redux store.

3. **Logger middleware** - Redux Logger is a middleware that logs the Redux actions and states changes to the console. It is helpful for debugging and understanding how the state evolves over time.

ii. **Store:** It is an object which provides the state of the application. This object is accessible with the help of the provider in the files of the project. The only way to change the state inside it is to **dispatch** an action on it. WORKING OF STORE, ACTION, REDUCERS

iii. **Action -** an action is a plain JavaScript object carrying information about the **type** of change to the state and any **payload** data required to make such a change.WORKING OF STORE, ACTION, REDUCERS

iv. **Dispatch** - When you call dispatch with an action object as an argument, Redux sends that action to the store, which then passes it to the reducers. The reducers will determine how the state should be updated based on the action type and the data provided in the action's payload.

v. **Dispatch chaining** - Dispatch chaining refers to the practice of dispatching multiple actions in a sequence or chain. It is a technique used in Redux to handle a series of actions one after another, typically in response to an event or asynchronous operation.

vi. **Reducer** - a reducer is a pure function that specifies how the application's state should change in response to actions dispatched to the Redux store. Reducers take in the current state and an action as arguments, and they return to a new state based on the action. WORKING OF STORE, ACTION, REDUCERS

vii. **useSelector** - useSelector hook is provided by the react-redux library and allows you to extract data from the Redux store in a React component.

viii. **Redux toolkit** - Redux Toolkit aims to make Redux development more efficient and developer-friendly by providing a set of recommended patterns and utilities. Redux

Toolkit includes a **configureStore()** function that combines multiple Redux setup steps into a single function call.

ix. **useLayoutEffect** - The useLayoutEffect hook in React is similar to the **useEffect** hook, but it is executed synchronously immediately after all DOM mutations are completed but before the browser has a chance to paint the screen. It allows you to perform actions that require access to the DOM layout or measurements.

x. **useImperativehandle** - useImperativeHandle is a React Hook that lets you customise the handle exposed as a ref.

xi. **Persistor in redux** - **Redux persist** takes your Redux state object and saves it to Persistence storage, so that the state can be restored when the application is reloaded or reopened. This is particularly useful for preserving user preferences, session data, or any other application-specific data that needs to persist beyond a single session.