**Practical No. 5**

**Study and implementation of ReactJs**

**PRN: 22510033**

**Batch: T5**

# Perform following problem statements using ReactJs Problem Statement 0: Basics of ReactJs

* What is React and what problem does it solve?

**React** is an open-source JavaScript library used for building user interfaces, particularly single-page applications (SPAs). It was developed by Facebook and is widely used to create complex, interactive, and dynamic web applications.

**Efficient Updates and Rendering**:

* Traditional DOM manipulation is slow and cumbersome, especially for complex applications with frequent updates. React’s virtual DOM ensures that updates are handled efficiently, minimizing the need for direct manipulation of the actual DOM.

**Component Reusability**:

* React encourages building applications using reusable components. This modularity allows developers to reuse code across different parts of the application or even across different projects, leading to more maintainable and scalable codebases.

**State Management**:

* Managing state in large applications can become complex and error-prone. React’s approach to state management within components, along with tools like React Context or third-party libraries like Redux, simplifies state management and makes it easier to understand and debug the application.

D**eclarative Programming**:

* React’s declarative syntax makes it easier to understand how the UI should look at any given time. This reduces bugs and simplifies the process of designing user interfaces, as developers focus on describing the UI rather than directly manipulating the DOM.

**Ecosystem and Tooling**:

* React has a vast ecosystem of libraries, tools, and extensions that can be integrated into projects. This ecosystem helps solve various problems such as routing, state management, and API integration, making React a flexible choice for web development.
* What are React components and how are they used?

**React components** are the building blocks of a React application. They are self-contained, reusable pieces of code that represent parts of the user interface (UI). Each component encapsulates its structure (HTML), style (CSS), and behavior (JavaScript) into a single entity, which can be composed with other components to build complex UIs.

* 1. **Functional Components**:

These are JavaScript functions that return a React element (usually JSX) representing the UI. Functional components are simple and primarily used for presenting static content or UI.

* 1. **Class Components**:

These are ES6 classes that extend React.component and must have a render method that returns JSX. Class components can have state and lifecycle methods, making them suitable for more complex scenarios.

Usage:

**Props**: Components receive inputs called props, which customize their output.

**State**: Components manage internal data using state.

**Composition**: Components can be nested within other components to build the UI.

* What is JSX in React?

**JSX (JavaScript XML)** is a syntax extension for JavaScript used in React to describe what the UI should look like. It allows you to write HTML-like code within JavaScript, which React then transforms into React elements that the browser can render.

* What are props in React and how do they differ from state?

**Definition**: Props are inputs to a React component. They are passed down from a parent component to a child component and are used to configure or customize the child component.

**Immutable**: Props are read-only. A component cannot modify its own props; they are controlled by the parent component.

**Usage**: Props are typically used to pass data or event handlers to child components.

**Props**:

Passed from parent to child.

Immutable within the receiving component.

Used to configure or pass data to child components.

**State**:

Managed within the component itself.

Mutable and can be updated within the component.

Used to handle dynamic data that affects the component’s rendering.

* What is state in React and how does it work?

**Definition**: State is a data structure that holds information about a component’s current situation or status. It is managed within the component itself.

**Mutable**: Unlike props, the state is mutable. A component can update its own state over time, typically in response to user actions or events.

**Usage**: State is used to manage dynamic data that can change over the lifecycle of the component.

* What are React lifecycle methods, and why are they important?

React lifecycle methods are special methods in class components that allow developers to hook into specific points in the component's lifecycle— from its initial creation and rendering, through updates, to its eventual unmounting. They are important because they provide control over the behaviour of components at different stages, enabling developers to manage side effects, optimize performance, and handle complex logic during these stages.

**Main Lifecycle Phases:**

**Mounting**: When the component is being created and inserted into the DOM.

* + constructor()
  + componentDidMount()

**Updating**: When the component is being re-rendered due to changes in props or state.

* + shouldComponentUpdate()
  + componentDidUpdate()

**Unmounting**: When the component is being removed from the DOM.

* + componentWillUnmount()
* Elaborate following with respect to ReactJs
  + Event Handling

function Button() { function handleClick() { alert('Button clicked!'); } return <button onClick={handleClick}>Click Me</button>; }

* + Conditional Rendering

function Greeting({ isLoggedIn }) {

return isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign in.</h1>;

}

* + Lists and Keys

Lists in React are used to render a collection of items. Each item in a list should have a unique key prop to help React identify and efficiently update individual items.

* + Forms

Forms in React handle user input. Unlike in plain HTML, form inputs in React are typically controlled components, meaning their values are controlled by React state.

* + Hooks

React Hooks allow you to use state and other React features in functional components. Common hooks include useState, useEffect, and useContext.

* + React Router

React Router is a library for handling routing in React applications. It allows you to define routes and navigate between different components based on the URL.

* + State Management

State management in React involves handling the state of your application in a way that ensures consistency and predictability. For complex applications, external state management libraries like Redux or MobX are often used.

* + React Context API

The Context API provides a way to share values like state, themes, or functions between components without passing props manually at every level.

* How can you optimize the performance of a React application?

**Use React.memo**: Memoize components to prevent unnecessary re-renders when props haven’t changed.

**Use useCallback and useMemo**: Memoize functions and values to avoid recreating them on every render.

**Lazy Loading Components**: Load components only when they are needed using React.lazy and Suspense.

**Code Splitting**: Break your code into smaller bundles to reduce the initial load time.

**Avoid Inline Functions**: Define functions outside of the render method to avoid re-creating them on each render.

**Efficient State Management**: Minimize the use of global state and ensure that components only re-render when necessary.

**Optimize Images and Assets**: Compress and optimize images, and use modern image formats like WebP.

**Use the Production Build**: Ensure your app is running in production mode, which includes optimizations like minification and dead code elimination.

**Use Virtualization for Large Lists**: Use libraries like react-window or react-virtualized to efficiently render large lists.

# Problem Statement 1: Star Wars character app

**(**In this problem statement, example of Star Wars is given, you may choose any characters from the series of the movie like Harry Potter, etc. Every group in a batch will have different characters.)

* Using a public API, display a list of all Star Wars characters using the endpoint “/people”. The API has paging, so the developer must also implement pagination. Also, a simple loader for fetching/refetching data as well as handling the error state (i.e., if the API server is down).
* For every user, we’d like to display a card with the name of each character along with a random picture for each character (see Picsum photos for random picture inspiration). Each character card should be colored based on their species and have some kind of animation when the user hovers over the card. When we click on a character’s card, more information should appear in a modal about the character.
* In the character details modal, we’d like to display information about the person: name as the header of the modal, height displayed in meters, mass in kg, date

person was added to the API (in dd-MM-yyyy format), number of films the person appears in and their birth year. We should also fetch information about the person’s homeworld and display its name, terrain, climate, and amount of residents





