BATCH: T5

PL-3 ASSIGNMENT NO. 5

**Study and implementation of ReactJs**

NAME: Priti Sunil Desai

PRN NO. : 22510062

**Perform following problem statements using ReactJs**

**Problem Statement 0: Basics of ReactJs**

1 What is React and what problem does it solve?

**React** is a popular open-source JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications (SPAs). It allows developers to create large web applications that can update and render efficiently in response to changing data. React is focused primarily on the **view layer** of an application, meaning it is concerned with what users see and interact with in the browser.

**Efficiently Updating the UI**: React uses a concept called the **Virtual DOM**. Instead of directly manipulating the actual DOM, React maintains a lightweight representation of the DOM (the Virtual DOM). When the state of a component changes, React efficiently updates only the parts of the DOM that need to change, resulting in faster and smoother UI updates.

**Component Reusability**: React promotes a **component-based architecture**, where UI elements are broken down into small, reusable components. These components can be composed and reused across different parts of an application, reducing redundancy and making the code more modular and maintainable.

**Declarative Programming Model**: React embraces a **declarative programming model**, where developers describe what the UI should look like for a given state, and React takes care of updating the UI to match that state. This approach simplifies reasoning about the code and reduces the likelihood of bugs.

2 What are React components and how are they used?

**React components** are the fundamental building blocks of a React application. They are isolated, reusable pieces of UI that manage their own state and behavior. In a React app, everything is essentially a component, from buttons and forms to entire pages and layouts. Components allow developers to break down complex user interfaces into smaller, more manageable parts that can be developed, tested, and reused independently.

* **Functional Components**: Thisare just like JavaScript functions that accept properties and return a React element.
* **Class based Components**:This are a little more complex than the functional components. A class component can show **inheritance**and access data of other components.

React components are used to build the UI of a React application. Each component can represent a small part of the UI or a more significant, complex part. Here's how they are used:

**Creating Components**:

* Components are created either as functions (functional components) or classes (class components).

**Composing Components**:

* Components can be composed together to form more complex UIs. For example, a Header component, a Footer component, and a Content component can be combined to create a Page component.

**Props**:

* **Props** (short for "properties") are used to pass data from a parent component to a child component.

**State**:

* **State** is an object that holds information that may change over the component's lifetime

**Lifecycle Methods**:

* Class components can use lifecycle methods to run code at specific times in a component's life (e.g., when it mounts, updates, or unmounts).

3 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 developers to write HTML-like code directly within JavaScript, making it easier to visualize and construct user interfaces. JSX is not a necessity for using React, but it is widely adopted because it simplifies the process of creating React components.

JSX is a powerful and convenient syntax extension in React that enables developers to write HTML-like code directly in JavaScript. It improves code readability, integrates seamlessly with JavaScript, and simplifies the process of creating complex UIs. While optional, JSX is a key feature that enhances the development experience in React.

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

**Props** in React are a way to pass data from one component to another. They are used to customize components and make them more dynamic by allowing them to receive input data and render content accordingly. Props are read-only and cannot be modified by the component that receives them; they are immutable.

**State** is another way to manage data in a React component, but unlike props, state is used to manage data that can change over time. State is local to the component in which it is defined and can be modified by that component, triggering a re-render to reflect the updated data in the UI.

1 **Props**: Immutable. They are passed from parent to child and cannot be modified by the receiving component. used to pass data from parent to child components, allowing for component customization and reuse.

2 **State**: Mutable. It is managed within the component and can be updated over time, which will trigger a re-render of the component. used to handle data that changes over time, directly impacting the component's rendering.

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

**State** in React is an object that holds information or data about a component that can change over time. Unlike props, which are immutable and passed down from parent to child components, state is managed and maintained within the component itself. When the state of a component changes, React automatically re-renders the component to reflect the new state in the UI.

1. In functional components, state is initialized using the useState hook.

2. In class components, state is initialized in the constructor

3 State is updated using the setState method in class components or the setState function provided by the useState hook in functional components.

4 When state is updated, React re-renders the component to display the new state.

5 When state changes, React triggers a re-render of the component to ensure the UI is updated to reflect the new state.

6 React uses an optimized rendering process with the Virtual DOM, so only the parts of the UI that need to change are re-rendered, making the update process efficient.

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

React lifecycle methods are special methods in React class components that are invoked at different stages of a component's life. They allow developers to hook into these stages to execute code at specific times, such as when a component is mounted, updated, or unmounted. These methods are critical for managing the side effects, optimizing performance, and handling resource cleanup in a React application.

A React component goes through three main phases in its lifecycle:

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

Updating: When the component is being re-rendered as a result of changes to props or state.

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

7 Elaborate following with respect to ReactJs

o Event Handling->

**Event handling** in React involves responding to user interactions like clicks, form submissions, and keystrokes. React provides a synthetic event system that wraps the native browser events for consistent behavior across different browsers.

o Conditional Rendering-> **Conditional Rendering**

**Conditional rendering** in React is used to render different components or elements based on certain conditions. This is achieved using JavaScript operators like if, ternary, and &&.

o Lists and Keys->

**Lists and keys** are essential for rendering multiple elements dynamically. React uses keys to identify which items have changed, are added, or are removed.

o Forms->

**Forms** in React are used to handle user input and form submissions. React provides controlled and uncontrolled components for managing form data.

o Hooks->

**Hooks** are functions that let you use state and other React features in functional components. They were introduced in React 16.8.

o React Router->

**React Router** is a library for handling routing in React applications. It allows you to manage navigation and render different components based on the URL.

o State Management->

**State management** involves handling and organizing the state of an application. React’s built-in state management can be extended with libraries for more complex needs.

o React Context API->

**React Context API** provides a way to share state across the entire app or part of it without passing props manually through every level of the component tree.

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

Optimizing the performance of a React application involves various strategies and techniques to ensure that the app runs efficiently, responds quickly to user interactions, and handles large amounts of data or complex UI updates smoothly. Here are some key methods for optimizing React performance:

**1.Avoid Unnecessary Re-renders**

**PureComponent and React.memo**:

* + Use React.PureComponent for class components or React.memo for functional components to prevent unnecessary re-renders. These components perform a shallow comparison of props and state to decide if re-rendering is necessary.

**ShouldComponent Update**:

* Implement the shouldComponentUpdate lifecycle method in class components to manually control when a component should re-render.

**2.Optimize Rendering**

**Key Prop in Lists**:

* + Ensure that each item in a list has a unique key prop to help React efficiently identify which items have changed.

**Avoid Inline Functions and Object Creation**:

* Avoid creating inline functions or objects within render methods, as this can create new instances on each render, causing unnecessary re-renders of child components.

**3.Code Splitting and Lazy Loading**

**React.lazy and Suspense**:

* + Use React.lazy and Suspense to load components only when they are needed, reducing the initial load time of your application.

**4.Optimize Performance with Refs**

**Avoiding Unnecessary DOM Access:**

Use refs for accessing and interacting with the DOM directly, especially when dealing with performance-critical operations like animations or focus management.

**5.Efficient State Management**

**Local vs. Global State**:

Use local state when possible to avoid unnecessary renders across the entire application. For global state, consider using state management libraries like Redux or the Context API with memoized selectors.

**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.

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.)

1.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).

2. 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.

3. 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, dateperson 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.







