**Batch: T5**

**Practical No. 5**

**Title of Assignment: Study and implementation of ReactJs**

**Student Name:Trishul Balaso Patil**

**Student PRN:22510051**

Perform following problem statements using ReactJs:

**Problem Statement 0: Basics of ReactJs**

**Q.1)What is React and what problem does it solve?**

React is a JavaScript library for building user interfaces, especially for single-page applications. It solves the following problems:

1. Component Reusability: Allows you to create reusable UI components.
2. Efficient DOM Updates: Uses a virtual DOM to update the real DOM efficiently.
3. State Management: Manages and updates UI state in a predictable way.
4. Declarative Syntax: Describes what the UI should look like based on the state, simplifying UI development.

**Q.2)What are React components and how are they used?**

React components are reusable pieces of code that define parts of a user interface. They can be:

1. Functional Components: Simple functions that return JSX.
2. Class Components: ES6 classes with state and lifecycle methods.

Usage:

* Create components to encapsulate UI and logic.
* Render components in other components using JSX.
* Manage state and handle events within components.

**Q.3) What is JSX in React?**

JSX (JavaScript XML) is a syntax extension for JavaScript used in React. It allows you to write HTML-like code within JavaScript, making it easier to define and visualize the user interface of components.

### Key Points About JSX:

* HTML-Like Syntax: JSX lets you write UI components in a syntax similar to HTML, which can be more intuitive than plain JavaScript.
* Integration with JavaScript: You can embed JavaScript expressions and logic directly within JSX, allowing dynamic content and interactivity.
* Attributes and Children: JSX supports attributes and nested elements, reflecting the hierarchical nature of HTML.
* Transformation: Since browsers do not understand JSX directly, it needs to be converted into regular JavaScript code using tools like Babel.

### Benefits:

* Improves Readability: Combining the structure of the UI with its logic in one place makes code easier to understand and maintain.
* Declarative Approach: JSX provides a more readable and declarative way to describe what the UI should look like based on the application's state.

**Q.4)What are props in React and how do they differ from state?**

### Props:

* Purpose: Props are used to pass data from a parent component to a child component. They allow components to be dynamic and reusable by providing them with different inputs.
* Immutability: Props are read-only and cannot be modified by the child component. They are set by the parent component and provide information or configuration.
* Usage: Props are used to configure or customize child components and to pass data between components.

### State:

* Purpose: State is used to manage and track data that changes over time within a component. It allows a component to maintain and update its own internal data.
* Mutability: State is mutable and can be changed by the component itself using functions like setState in class components or the useState hook in functional components.
* Usage: State is used to handle dynamic and interactive elements within a component, such as user input, form data, or toggling features.

### Key Differences:

* Source of Data:
  + Props: Provided by the parent component and used to pass data down to child components.
  + State: Managed within the component itself and used to handle internal data and interactions.
* Modification:
  + Props: Cannot be modified by the child component; they are immutable.
  + State: Can be updated by the component that owns it.
* Lifecycle:
  + Props: Exist for the lifetime of the parent component and are passed down from parent to child.
  + State: Can change over time in response to user actions or other events within the component.

**Q.5)What is state in React and how does it work?**

State **in React is a mechanism for managing and tracking data within a component that can change over time. It allows components to handle dynamic content and interactions.**

### Key Features of State:

* Purpose: State is used to keep track of data that can change, such as user inputs or toggles. It helps manage and update the component's UI based on user interactions or other events.
* Mutability: State can be updated by the component itself. When the state changes, React automatically re-renders the component to reflect the updated data.
* Initialization: State is initialized when a component is created. It starts with default values and can be updated as needed.
* Encapsulation: State is specific to the component that owns it. Other components cannot directly modify it, although the state can be passed down as props if needed.

### How It Works:

1. Initialization: State is set up when the component is first rendered, with default values provided by the component.
2. Updating State: Components can update their state through methods or hooks provided by React. This triggers a re-render of the component.
3. Re-rendering: When state changes, React updates the component's UI to reflect the new state. This ensures that the user interface remains in sync with the component's data.
4. Local to Component: State is local to the component and is not shared directly with other components, though it can be passed down as props.

**Q.6)What are React lifecycle methods, and why are they important?**

React lifecycle methods are special methods in class components that allow you to hook into different phases of a component's lifecycle. They provide ways to perform actions at specific points, such as when the component is created, updated, or removed.

### Key Phases of the Component Lifecycle:

1. Mounting: When a component is being created and inserted into the DOM.
   * Important Methods:
     + constructor(): Initializes state and binds methods.
     + componentDidMount(): Invoked immediately after the component is added to the DOM. Useful for performing setup tasks like data fetching.
2. Updating: When a component is being re-rendered due to changes in state or props.
   * Important Methods:
     + componentDidUpdate(prevProps, prevState): Invoked immediately after updating occurs. Useful for performing side effects based on state or prop changes.
     + shouldComponentUpdate(nextProps, nextState): Determines if a component should re-render. Helps optimize performance by preventing unnecessary updates.
3. Unmounting: When a component is being removed from the DOM.
   * Important Method:
     + componentWillUnmount(): Invoked immediately before a component is removed. Useful for cleanup tasks like invalidating timers or canceling network requests.
4. Error Handling: When there’s an error during rendering, in a lifecycle method, or in the constructor of any child component.
   * Important Method:
     + componentDidCatch(error, info): Allows you to handle errors gracefully and provide fallback UI.

### Importance of Lifecycle Methods:

1. Resource Management: They help manage resources, such as network requests, subscriptions, and timers, ensuring they are created and cleaned up properly.
2. Performance Optimization: Methods like shouldComponentUpdate help optimize performance by controlling unnecessary re-renders.
3. Side Effects: They allow you to perform side effects, such as fetching data or manually interacting with the DOM, at appropriate times during the component’s lifecycle.
4. Error Handling: They provide a way to catch and handle errors in the component tree, improving the robustness of the application.

**Q.7)Elaborate following with respect to ReactJs**

**o Event Handling**

**o Conditional Rendering**

**o Lists and Keys**

**o Forms**

**o Hooks**

**o React Router**

**o State Management**

**o React Context API**

### 1. Event Handling

In React, event handling involves attaching event listeners (like onClick, onChange) to elements. React uses a synthetic event system to ensure consistent behavior across browsers. Event handlers are functions that respond to user interactions, such as clicks or form submissions.

### 2. Conditional Rendering

Conditional rendering in React allows you to display different UI elements based on conditions. This is achieved using JavaScript logic in JSX, such as if-else statements or ternary operators, to render components or elements conditionally.

### 3. Lists and Keys

React uses lists to render multiple elements efficiently. Each element in a list should have a unique key prop to help React identify and manage items during updates. Keys are essential for optimizing performance and ensuring correct rendering when elements change.

### 4. Forms

Forms in React handle user input through controlled components. The form elements (input fields, checkboxes) are linked to the component’s state. Event handlers manage changes, and form submissions are handled with functions that prevent default behavior and process the data.

### 5. Hooks

Hooks are functions that allow you to use state and other React features in functional components. Key hooks include useState for managing state and useEffect for side effects (e.g., data fetching). Hooks simplify component logic and state management without using class components.

### 6. React Router

React Router is a library for handling navigation and routing in React applications. It enables the creation of single-page applications with multiple views. Routes are defined to render different components based on the URL, allowing for client-side navigation.

### 7. State Management

State management in React involves handling and updating component state. For local state, use useState in functional components or this.state in class components. For global state across components, libraries like Redux or Context API are often used.

### 8. React Context API

The React Context API provides a way to pass data through the component tree without having to pass props manually at every level. It’s used for global state management, allowing components to access shared state and functions efficiently.

**Q.8)How can you optimize the performance of a React application?**

To optimize React application performance:

1. Memoization: Use React.memo and PureComponent to prevent unnecessary re-renders.
2. State Management: Update state only when necessary and keep state local.
3. Code Splitting: Use React.lazy and Suspense for loading components only when needed.
4. Lazy Loading: Load images and resources as they come into view.
5. Avoid Inline Functions: Define functions outside render or use useCallback.
6. Unique Keys: Ensure list items have unique, stable keys and use virtualization for large lists.
7. Minimize Reconciliation: Batch state updates and avoid expensive operations in render methods.
8. Optimize CSS: Use efficient styling methods and avoid complex CSS that causes reflows.
9. Performance Analysis: Use React DevTools and browser performance tools to identify and fix bottlenecks.

**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**

**Output:**



