**Module-7 React - Applying Redux**

1. **What is Redux?**

* Redux is an open-source library for managing application state in JavaScript applications. It is often used in conjunction with React, but it can be used with other libraries or frameworks as well.
* Redux works by providing a central store that holds the entire state of an application. The state is updated using pure functions called reducers, which take the current state and an action as arguments and return a new state.

1. **What is Redux Thunk used for?**

* Redux Thunk is a middleware library that allows you to write asynchronous logic that interacts with the Redux store. It is used to handle asynchronous actions in Redux applications.
* When you dispatch an action in Redux, it is typically a plain JavaScript object that describes the action that has occurred. However, sometimes you may need to perform some asynchronous logic before dispatching an action. For example, you may need to make an API call to retrieve some data, or you may need to perform some calculations that take some time.

1. **What is Pure Component? When to use Pure Component over Component?**

* In React, a Pure Component is a special type of Component that implements the shouldComponentUpdate() lifecycle method to perform a shallow comparison of the component's props and state. If there is no change in props or state, then the component does not re-render. This optimization technique can help improve performance in applications with complex component hierarchies or large data sets.
* In general, it is recommended to use Pure Components over regular Components whenever possible, as they can help reduce the number of unnecessary re-renders, which can improve the performance of your React application.

1. **What is the second argument that can optionally be passed tosetState and what is its purpose?**

* The second argument that can optionally be passed to the setState() method in React is a callback function. This function will be executed after the state has been updated and the component has been re-rendered.
* The purpose of this callback function is to allow you to perform additional operations or actions after the state has been updated. For example, you could use this callback function to update the component's props based on the new state, fetch new data from an API, or perform any other action that depends on the updated state.
* Here is an example of using the second argument of setState() to update the component's props based on the new state:
* **class MyComponent extends React.Component {**

**constructor(props) {**

**super(props);**

**this.state = {**

**count: 0**

**};**

**}**

**handleButtonClick = () => {**

**this.setState({ count: this.state.count + 1 }, () => {**

**this.props.onCountChange(this.state.count);**

**});**

**}**

**render() {**

**return (**

**<div>**

**<p>Count: {this.state.count}</p>**

**<button onClick={this.handleButtonClick}>Increment Count</button>**

**</div>**

**);**

**}**

**}**

* In this example, the handleButtonClick() method calls setState() to update the count state. It also passes a callback function as the second argument to setState(), which updates the component's props by calling the onCountChange() function with the new count value. This allows the parent component to receive updates about the count value whenever it changes.

1. **Create a Table and Search data from table using React Js ?**

* **import React, { useState } from "react";**

**const Table = ({ data }) => {**

**const [searchTerm, setSearchTerm] = useState("");**

**const filteredData = data.filter((item) =>**

**item.name.toLowerCase().includes(searchTerm.toLowerCase())**

**);**

**return (**

**<div>**

**<input**

**type="text"**

**placeholder="Search..."**

**onChange={(e) => setSearchTerm(e.target.value)}**

**/>**

**<table>**

**<thead>**

**<tr>**

**<th>Name</th>**

**<th>Age</th>**

**</tr>**

**</thead>**

**<tbody>**

**{filteredData.map((item) => (**

**<TableRow key={item.id} name={item.name} age={item.age} />**

**))}**

**</tbody>**

**</table>**

**</div>**

**);**

**};**

**const TableRow = ({ name, age }) => {**

**return (**

**<tr>**

**<td>{name}</td>**

**<td>{age}</td>**

**</tr>**

**);**

**};**

**export default Table;**

1. **Create Login registration with CRUD Application using API (Redux)**

* **const express = require('express');**

**const bodyParser = require('body-parser');**

**const app = express();**

**const port = 3000;**

**app.use(bodyParser.urlencoded({ extended: true }));**

**app.use(bodyParser.json());**

**let users = [**

**{**

**id: 1,**

**name: 'John Doe',**

**email: 'john.doe@gmail.com',**

**password: 'password123'**

**}**

**];**

**// Register a new user**

**app.post('/api/users/register', (req, res) => {**

**const { name, email, password } = req.body;**

**const id = users.length + 1;**

**users.push({ id, name, email, password });**

**res.send(`User ${name} registered successfully!`);**

**});**

**// Authenticate user credentials**

**app.post('/api/users/login', (req, res) => {**

**const { email, password } = req.body;**

**const user = users.find(u => u.email === email && u.password === password);**

**if (user) {**

**res.send(`Welcome, ${user.name}!`);**

**} else {**

**res.status(401).send('Invalid email or password.');**

**}**

**});**

**// Get all users**

**app.get('/api/users', (req, res) => {**

**res.json(users);**

**});**

**// Get a specific user**

**app.get('/api/users/:id', (req, res) => {**

**const user = users.find(u => u.id === parseInt(req.params.id));**

**if (user) {**

**res.json(user);**

**} else {**

**res.status(404).send('User not found.');**

**}**

**});**

**// Update a user**

**app.put('/api/users/:id', (req, res) => {**

**const { name, email, password } = req.body;**

**const user = users.find(u => u.id === parseInt(req.params.id));**

**if (user) {**

**user.name = name || user.name;**

**user.email = email || user.email;**

**user.password = password || user.password;**

**res.send(`User ${user.name} updated successfully!`);**

**} else {**

**res.status(404).send('User not found.');**

**}**

**});**

**// Delete a user**

**app.delete('/api/users/:id', (req, res) => {**

**const userIndex = users.findIndex(u => u.id === parseInt(req.params.id));**

**if (userIndex !== -1) {**

**users.splice(userIndex,**