Ordered React Learning Topics (Deduplicated)

1. Foundational Concepts

- Library vs Framework
- Features of React.js
- Advantages and Disadvantages of React.js
- Virtual DOM
- Real DOM vs Virtual DOM
- Diffing
- Reconciliation
- React Fiber
- Single Page Application (SPA)
- Cons of SPA
- Client-Side Rendering (CSR) vs Server-Side Rendering (SSR)
- Initial Load Time Difference between CSR and SSR
- SEO and CSR
- JSX
- Key Differences between JSX and HTML
- Rules of JSX
- React without JSX
- React.createElement
- htmlFor Attribute
- Transpilation
- Babel
- Webpack
- Package.json vs Package-lock.json
- npm
- Vite
- Pros of Vite
- Hot Module Replacement
- ES Modules
- Named Exports without Curly Braces
- Destructuring
- Arrow Functions
- Spread Operator
- Span Tag
- Viewport
- DOM Direct Manipulation

2. React Components

- Component Definition
- Functional vs Class Components
- React Classes
- Component Lifecycle
- Lifecycle Methods in Class Components

- Stateful vs Stateless Components
- Props
- State vs Props
- Props Immutability
- State vs Local Variables
- Why prevState is Used for Mutation
- Conditional Rendering
- Conditional Rendering using && Syntax
- Dynamic Rendering
- Custom Components
- Built-in Components (Fragment, Suspense, StrictMode)
- React.StrictMode
- Enabling/Disabling StrictMode
- Console Message Appearing Twice
- Fragments
- Advantages of React Fragment
- Fragment vs
- Purpose of Root DOM Element
- DefaultProps
- PropTypes
- React.createElement vs React.cloneElement

3. Hooks

- Hooks in React
- Rules of Hooks
- useState
- Difference between Normal let, const Variables and useState
- useEffect
- useEffect with No Dependency Array
- useEffect with Empty Dependency Array
- Dependencies of useEffect
- Cleanup Function in useEffect
- Handling componentWillUnmount in Functional Components
- Handling componentDidUpdate in Functional Components
- useLayoutEffect
- useEffect vs useLayoutEffect
- useRef
- Advantages of useRef
- Updating useRef
- ref.current.value
- useMemo
- Cons of Memoization
- useCallback
- useMemo vs useCallback
- useReducer
- useContext
- Custom Hooks

- Return in useEffect Hook Syntax

4. Component Communication

- Parent to Child Communication
- Child to Parent Communication
- Lifting State Up
- Prop Drilling
- How to Overcome Prop Drilling
- ForwardRef
- update child component's state from parent component

5. Event Handling

- Event Handling
- Event Handlers (e.g., handleClick, handleChange)
- Synthetic Events
- Advantages of Synthetic Events
- Examples of Synthetic Events
- Event Pooling
- e.preventDefault

6. Forms and Components

- Controlled Components
- Uncontrolled Components
- Benefits of Uncontrolled Elements
- How to Pull Data from Uncontrolled Components
- Manipulating Uncontrolled Components

7. React Router

- React Router
- Types of Routers
- useNavigate
- useHistory vs useNavigate
- Link vs NavLink
- Outlet

8. State Management

- State Management
- Context API
- useReducer

9. Performance Optimization

- React.memo
- Code Splitting
- Lazy Loading

- Suspense
- Fallback
- React Lazy
- Debouncing
- Throttling
- Tree Shaking
- React Profiler
- Profiling
- How to Improve Performance in React
- Limitations of React (SEO, Initial Load)

10. Advanced Topics

- Error Boundaries
- Pure Components
- Higher-Order Components (HOCs)
- Advantages of HOCs
- Render Props
- React Portals
- dangerouslySetInnerHTML
- Shadow DOM
- Shadow DOM vs Virtual DOM
- React Mixins
- Polyfills
- Axios Interceptor
- Shallow Rendering
- React 19 Features
- Directives (use client, use server)
- Integrating React into Existing Application
- Incremental Rendering
- ReactDOM.createRoot

11. Storage

- Local Storage vs Session Storage
- IndexedDB

12. Styling

- Sass Styling
- Adding Styles to Mapped Elements

13. Debugging and Tools

- Debug React
- React Profiler
- Webpack vs Vite

14. Practical Workouts

- Basic Component Workout
- Return Statement
- One Single Element Returned
- Map Syntax
- Filter Method vs Reduce Method
- Add Two Numbers
- Check Matching Input Fields
- Timer Using useEffect
- setInterval in useEffect
- clearInterval
- Toggle Checkbox via Button (Parent-Child Communication)
- Fetch Data Using useEffect
- Display Text in h1 from Input
- Counter Component
- List ul li Implementation
- Show Alerts

15. To-Do App Features

- Prevent Duplicate Tasks
- Mark Task as Completed
- Edit Tasks
- Delete Confirmation
- Local Storage
- Re-order Tasks
- Display Time Since Added
- Filter Tasks
- Count Todos
- Display Count of Completed Todos
- Optimistic Updates

16. Miscellaneous

- One-Way Data Binding
- How SPA Loads Dynamic Data
- Flow of React
- Keys in Diffing Algorithm
- Key Prop in List
- Can Index of an Array Be Used as Key Prop?
- PNG vs JPG

WEEK 2 REACT

1. Foundational Concepts

- Fiber Improvements
- Hydration
- Automatic Batching
- React Concurrent Mode
- Referential Equality
- Memory Leaking
- Parts of URL
- Transpiler vs Compiler

2. React Components

- Component Composition
- Pure Functions
- React.memo vs Pure Components
- React.memo vs useMemo
- Consumers in Class Component Context
- componentDidCatch

3. Hooks

- useId
- useImperativeHandle
- createRef
- useMemo Dependency
- useReducer Action Dispatching
- useReducer 3rd Argument
- init in useReducer
- Stale Closure
- When to Use useCallback
- What Happens if Dependency Array is Wrong in useMemo
- What Happens if Dependency Array is Wrong in useCallback
- What Happens if We Pass Empty Array for useMemo
- What Happens if We Pass Empty Array for useCallback
- Can a Component Have Multiple useContext
- Use Cases of useContext
- Methods to Store and Update Data Using Context API
- Modifying State Passed Down Through Context
- Updating State Passed Deep Down Through Context
- useLocation

4. Component Communication

- Props Proxy for HOC
- Applications of HOCs

5. React Router

- <Link/> vs <a>
- MemoryRouter
- Hash Router
- Reading Current URL
- Reading Query Params
- Reading Path Params
- Redirecting User
- Switch
- Explore History Object
- Route Protection
- React.lazy
- Suspense fallback
- outlet

6. State Management

- Global State Management System Using useContext and useReducer

7. Performance Optimization

- Disadvantage of Lazy Loading
- How to Load Millions of Data Smoothly from Server Side to Client Side

8. Event Handling

- Timer Functions

9. Forms and Components

- Don’t Trim Passwords
- File Type and Size Validation
- Prevent Showing Login/Signup if Logged In
- Restrict Auth Pages if Authenticated
- Validation Messages
- Auth State Handling

10. Storage

- Use .env File
- Store Secrets in Environment Variables
- Avoid Secrets in Code

11. Debugging and Tools

- Avoid Loose API Calls in Components
- Create Axios Instance
- Axios Interceptors
- Axios CancelToken

12. Practical Workouts

- Implement Lazy Loading (Load Component After 5 Sec)
- Find Average of 2 Numbers Using useRef
- Make a Component to Find the Square of a Number Using useMemo
- Calculator Using useContext
- Create Counter Using useReducer
- Simple Calculator Using useReducer
- Create Counter App Using useCallback
- Increment and Decrement Counter Using useContext
- Save User Data in Context & Use It in Components
- Fetch Data from Context & Render It in UI
- Implement Context for Parsing the User Logged-In State
- Render a Component Once Page is Resized
- Change Background Color Using useRef
- Change Div Color When Clicking Button
- useRef Set Timer with Start and Stop
- Auto Increment/Decrement Counter Using useState or useReducer
- Netflix Signup/Login

13. OLX Clone Features

- OLX Clone (~60% Complete)
- Clone 4-5 Pages
- App Should Be Mobile Optimized
- Include Signup & Login Forms
- Avoid Browser Alerts
- Add Product
- After Adding Product, Redirect to Product Detail Page
- Product Details Page
- Favorite Items
- Edit Items
- Sell Page Should Be Protected
- Fix Bugs with OLX Project
- Changes When User Logs Out

14. Miscellaneous

- Return Value of Reducer Function
- Dynamic Imports
- Why Not Pass Index as Key

WEEK 3 REACT

1. JavaScript and ES6 Fundamentals

Essential JavaScript concepts for React development.

- **DOM Concepts in JavaScript**: Understand DOM manipulation for direct element access in React.
- **ES6 Versions Syntaxes (Map, Filter, Reduce)**: Use map for rendering lists, filter for search, reduce for aggregating data (e.g., task counts).
- **Import/Export**: Structure code with ES modules for component imports.
- **Arrow Functions**: Write concise event handlers and callbacks for Todo app actions.

2. React Core Concepts

Foundational React concepts for building the Todo app.

- **React as a Library vs. Framework**: Recognize React as a UI library, not a full framework.
- **Single Page vs. Multi-Page Application**: Understand SPAs for dynamic Todo app rendering.
- **Virtual DOM**: Learn how React optimizes updates via virtual DOM.
- **Reconciliation**: Study how React compares virtual DOMs for efficient rendering.
- **React Fiber**: Explore React's reconciliation engine for smooth task list updates.
- **JSX**: Write UI with JSX syntax for Todo app components.
- **Transpiler vs. Compiler**: Understand Babel's role in converting JSX to JavaScript.
- **Babel**: Learn how Babel transpiles JSX for the Todo app.
- **Webpack**: Understand bundling for React apps.
- **Vite**: Explore Vite as a faster alternative to Webpack.
- **Introduction to Vite**: Learn Vite for setting up the Todo app.
- **Advantages of Vite over CRA**: Compare Vite's speed with Create React App.
- **Understanding vite.config.js and Basic Project Structure**: Configure Vite for the Todo app.
- **ReactDOMClient**: Use createRoot for rendering the app.
- **ReactDOMServer**: Understand server-side rendering (SSR) for advanced use.
- **What is the Purpose of the Root DOM Element in React?**: Learn where React mounts the app.
- **Concept of Reusability**: Build reusable components like TodoItem.
- **Why We Keep Component Names in Pascal Case in React?**: Follow React naming conventions.
- **Dynamic Rendering**: Render UI based on state (e.g., task list).
- **One-Way Data Binding**: Understand React's data flow for props and state.
- **Latest Features in React**: Explore React 18/19 features like automatic batching.
- **Automatic Batching**: Learn how React groups state updates for performance.
- **React Concurrent**: Understand concurrent rendering for smoother UI.
- **Polyfills**: Add browser compatibility for older browsers.

3. Components and Props

Structure components and manage data flow for the Todo app.

- **What is a Component?**: Learn React's building blocks (e.g., TodoForm).
- **Functional Components**: Use modern functional components with hooks.
- **Class Components (Basic - Lifecycle Methods)**: Understand legacy class components.
- **Props vs. State**: Differentiate immutable props and mutable state.
- **Props Immutability**: Ensure props are not modified in child components.
- **Key Props in Arrays**: Use unique IDs for efficient task list rendering.
- **PropTypes in React: How to Validate Props?**: Validate props for TodoItem.
- **Passing Data from Parent to Child**: Pass task data to TodoItem.
- **Passing Data from Child to Parent Using Callback Functions**: Send delete/edit actions to parent.
- **Passing Data from Child to Parent Using useRef**: Explore refs for child-to-parent communication.
- **Passing Data from Child to Parent Using Context**: Use context for global data sharing.
- **Lifting State Up in React**: Centralize state in App for task management.
- **Prop Drilling**: Pass props through components; consider alternatives.
- **How to Overcome Prop Drilling**: Use useContext for global state.
- **Siblings Communication in React**: Communicate between TodoItem components via parent.
- **Siblings Communication Using Middleware**: Use Redux for sibling interactions.
- **React Mixins**: Understand legacy mixins (rarely used in modern React).
- **ForwardRef**: Pass refs to child components for input focus.
- **Render Props**: Share code between components (alternative to hooks).
- **Higher-Order Components (HOC)**: Wrap components for added functionality.
- **Examples for HOF**: Study HOC patterns for logging or analytics.
- **Create Props Proxy for HOC**: Implement HOC for prop manipulation.
- **Pure Components**: Optimize components with shallow prop comparison.
- **Creating a Pure Component**: Use React.memo for TodoItem.
- **Controlled vs. Uncontrolled Components**: Use controlled inputs for task form.
- **Accessing Data from Uncontrolled Element**: Use refs for uncontrolled inputs.
- **Shadow DOM**: Understand isolated DOM for styling (optional for Todo app).
- **Shadow DOM vs. Virtual DOM**: Compare React's virtual DOM with shadow DOM.

4. State Management with Hooks

Manage state and side effects for the Todo app.

- **What is State?**: Manage task list and UI state with useState.
- **useState**: Store tasks and form inputs.
- **Return Datatype of useState**: Understand [state, setState] tuple.
- **Updating State That's an Array**: Use spread operator for task list updates.
- **Why Arrays and Objects Are Copied When Updating State**: Maintain immutability.

- **Purpose of Using Spread Operator When Updating**: Ensure immutable state updates.
- **Difference Between Normal let, const Variables and useState**: Compare reactivity.
- **State vs. Local Variables**: Use state for reactive UI, variables for non-reactive.
- **useRef**: Persist values like input focus without re-renders.
- **Advantages of useRef**: Access DOM or store mutable data.
- **useRef vs. Regular Variable to Hold Value**: Compare non-reactive storage.
- **Changing Value in useRef**: Update ref.current for input focus.
- **useEffect**: Handle side effects like local storage sync.
- **Which Lifecycle Hooks Are Replaced by useEffect in Functional Components?**: Map to componentDidMount, DidUpdate, WillUnmount.
- **useEffect's Behavior When Dependency Array is Empty vs. Not Provided**: Understand mount vs. every render.
- **[] in useEffect**: Run effect only on mount.
- **Dependencies of useEffect**: Control effect triggers.
- **Cleanup Function in useEffect**: Clean up timers or subscriptions.
- **How to Achieve componentWillUnmount in Functional Component**: Use useEffect cleanup.
- **Unmounting in Functional Component**: Handle component removal.
- **useLayoutEffect**: Run effects before browser paint.
- **Difference Between useEffect and useLayoutEffect**: Compare timing of effects.
- **useCallback**: Memoize callbacks to prevent re-creation.
- **Need Practical Exposure of useMemo and useCallback**: Practice memoizing search functions.
- **useMemo**: Memoize computed values like filtered tasks.
- **useMemo Dependency List**: Control memoization triggers.
- **Drawbacks of useMemo & React.memo**: Understand memory overhead.
- **Limitations of useMemo & useCallback**: Evaluate performance trade-offs.
- **useReducer**: Manage complex state logic (alternative to useState).
- **useReducer vs. Redux**: Compare local vs. global state management.
- **Replace useState by useReducer**: Use for complex task state.
- **useContext**: Share global state like user data.
- **Example of Context API Usage**: Implement theme or user context.
- **Context API vs. Redux**: Compare simplicity vs. scalability.
- **Implementation Using useContext**: Use for user authentication state.
- **Send Data from Child to Parent Using Context**: Access context in child components.
- **useTransition**: Prioritize UI updates for smoother interactions.
- **useFormState**: Manage form state (React 19 feature, if applicable).
- **Rules of Using Hooks**: Call hooks at top level, not in conditions.
- **Why Can't Hooks Be Used in Conditions?**: Ensure consistent hook order.
- **Function vs. Custom Hook**: Differentiate regular functions from hooks.
- **Custom Hook Scenario**: Create useLocalStorage for tasks.
- **Create a Custom Hook**: Build reusable logic for task persistence.
- **Create Custom Hook for Increment/Decrement Counter**: Practice with counter logic.
- **Hook vs. Function**: Understand hook restrictions and reactivity.

- **Limitations of Functional Components**: Compare with class components.

5. Form Handling and Validation

Implement forms with validation for tasks and user management.

- **Event Handlers (onClick, onChange)**: Handle form submissions and clicks.
- **Synthetic Events**: Normalize browser events for consistency.
- **Advantages of Using Synthetic Events**: Ensure cross-browser compatibility.
- **Event Pooling**: Understand event reuse for performance.
- **Examples of Synthetic Events**: Study onClick, onChange in forms.
- **Controlled vs. Uncontrolled Components**: Use controlled inputs for task form.
- **How to Pull Data from an Uncontrolled Component**: Access via refs.
- **Validation for Todo App**: Prevent empty or duplicate tasks.
- **Signup Input Validation**: Validate user email and password.
- **Email Should Be Unique**: Ensure unique emails in signup.
- **User Add Error Messages Not Communicated Properly**: Display clear error messages.
- **File Type and File Size**: Validate profile image uploads (e.g., PNG/JPG, size limit).
- **PNG vs. JPG**: Choose PNG for transparency in profile images.

6. Routing and Navigation

Add navigation for the Todo app.

- **React Router**: Implement routing for task views.
- **useNavigate**: Navigate programmatically (replaces useHistory).
- **useHistory vs. useNavigate**: Understand React Router v6 changes.
- **useLocation**: Access current URL for dynamic routing.
- **Dynamic Routing**: Create routes for task categories.
- **Link vs. NavLink**: Use NavLink for active link styling.
- **Outlet**: Render nested routes in Router.
- **Hash Router & Memory Router**: Explore alternative routers.
- **Add Protected Routes**: Restrict routes to authenticated users.

7. Persistence and Side Effects

Persist data and handle async operations.

- **Local Storage vs. Session Storage**: Save tasks in local storage.
- **Explore the Use Cases of Browser Storages**: Compare local storage, session storage, IndexedDB.
- **LocalStorage vs. Redux**: Use local storage for simple persistence, Redux for state.
- **Save Todos in Local Storage**: Persist tasks across sessions.
- **Manage Sessions and Cookies**: Handle user sessions with cookies.
- **httpOnly Cookies**: Secure cookies for authentication tokens.
- **Axios Interceptors**: Add token headers to API requests.
- **Axios CancelToken**: Cancel ongoing API requests.

8. Redux and Global State Management

Manage global state for tasks and authentication.

- **Redux**: Centralize state for tasks and user data.
- **Flux Architecture**: Understand Redux's predecessor.
- **Redux vs. Flux**: Compare state management approaches.
- **Core Principles of Redux**: Single source of truth, immutable state, pure reducers.
- **Single Source of Truth**: Store all app state in Redux.
- **Why Immutability Important in Redux, and How is it Achieved**: Use Immer for immutability.
- **Redux Store**: Central state container.
- **Role of a Store**: Manage app state.
- **Redux Store Methods**: Use getState, dispatch, subscribe.
- **Actions**: Define state changes (e.g., addTodo).
- **Create an Action**: Write action creators for tasks.
- **Action Creators and Action Types in Redux**: Structure actions for clarity.
- **Why Actions Are Important**: Trigger state updates predictably.
- **Reducers**: Pure functions to update state.
- **Why is Reducer a Pure Function**: Ensure predictable state changes.
- **Combined Reducers**: Combine task and auth reducers.
- **Practice Implementation of Combined Reducers**: Structure reducers for scalability.
- **Dispatcher**: Dispatch actions to update state.
- **Dispatch vs. Subscribe**: Dispatch actions vs. listen to state changes.
- **How to Dispatch an Action**: Use useDispatch for task actions.
- **Constants in Redux**: Define action types as constants.
- **Components of Redux**: Store, actions, reducers.
- **Advantages and Disadvantages of Redux**: Evaluate complexity vs. scalability.
- **Limitations of Redux**: Understand boilerplate and overhead.
- **Pros and Cons of Redux**: Weigh benefits for Todo app.
- **Student Need to Understand the Flow of Redux, Dispatching the Action**: Practice action flow.
- **Redux Toolkit**: Simplify Redux with createSlice, configureStore.
- **What is Redux Toolkit**: Learn utilities for easier Redux setup.
- **Configure Store**: Set up Redux store with Toolkit.
- **Redux Slice**: Create slices for tasks and auth.
- **extraReducer**: Handle async actions in slices.
- **Immer and How is it Used in Redux**: Simplify immutable updates.
- **Selector in Redux**: Access specific state (e.g., completed tasks).
- **Purpose of Selectors in Redux**: Optimize state access.
- **useSelector**: Select Redux state in components.
- **mapStateToProps**: Legacy method for state access.
- **mapDispatchToProps**: Legacy method for dispatching actions.
- **useDispatch Hook**: Dispatch actions in functional components.
- **Redux Middleware**: Handle async operations and logging.
- **Purpose of Middleware in Redux**: Extend Redux for async tasks.
- **Redux Thunk**: Handle async actions simply.

- **Redux Thunk & Redux Saga**: Compare async middleware.
- **When to Use Thunk and Saga**: Use Thunk for simple async, Saga for complex flows.
- **createAsyncThunk**: Create async actions with Toolkit.
- **Call() and Put() in Redux Saga**: Manage Saga effects for async tasks.
- **Redux-Saga**: Handle complex async workflows.
- **How to Handle Asynchronous Operations in Redux**: Use Thunk/Saga for API calls.
- **Synchronous and Asynchronous Actions in Redux**: Differentiate action types.
- **Where We Should Do API Call in Redux**: Use Thunk/Saga for API calls.
- **Redux Persistence**: Persist Redux state with redux-persist.
- **How Does Redux Persist Work?**: Save state to local storage.
- **LocalStorage vs. Redux**: Compare persistence methods.
- **How Redux Ensures Components Updated Correctly**: Use selectors and immutability.
- **How to Store Data to Redux Store**: Dispatch actions to update state.
- **Clear Redux State**: Reset state on logout.
- **How to Debug a Redux Application**: Use Redux DevTools.
- **use Redux DevTools**: Monitor state and actions.
- **How Should You Structure a Redux Application?**: Organize slices, actions, reducers.
- **Learn About Redux Folder Structure (Large Scale Application)**: Structure for scalability.
- **Common Performance Optimizations for Redux Applications**: Optimize selectors, memoization.
- **Optimization Techniques for Redux App**: Reduce re-renders, use Toolkit.
- **How to Handle Impure Actions in Redux**: Use middleware for side effects.
- **Side Effects**: Manage async operations and external effects.
- **Implement a Reducer to Handle Multiple Action Types**: Write task reducer.
- **Concept of Immutable State and Why It is Important in Redux**: Ensure predictable updates.
- **Workout: Dispatching and Accessing State from Store**: Practice task actions.
- **Workout: Save the User Email in Redux Store & Render the Data in Component**: Store user data.
- **Workout: Update the Data Inside Redux Store by Dispatching an Action from a Component**: Update tasks via actions.

9. Authentication and Authorization

Secure the Todo app with JWT and user management.

- **Authentication and Authorization**: Differentiate user verification and access control.
- **JWT (What is JWT Token, Structure, Claims, Security)**: Use JSON Web Tokens for auth.
- **Parts of Token**: Understand header, payload, signature.
- **JWT Signature**: Verify token integrity.
- **JWT Working**: Learn token creation and verification.

- **Jwt.verify vs. Decode**: Verify checks signature; decode reads payload.
- **Jwt Decode vs. Verify**: Clarify decoding vs. validation.
- **Access Token and Refresh Token**: Use access for API calls, refresh for renewal.
- **Purpose of Access Token and Refresh Token**: Secure short-term access, long-term sessions.
- **Use of Refresh Token**: Renew expired access tokens.
- **What Determines the Lifetime of Access Token**: Set expiry (e.g., 15m).
- **How Can You Detect Token Expiry in a Frontend Application?**: Check expiry in payload.
- **Where Should Access and Refresh Tokens Be Stored on Client Side?**: Use httpOnly cookies or secure storage.
- **Which is the Ideal Place to Store JWT Token**: Prefer httpOnly cookies for security.
- **Do Not Print/Log Tokens in Code**: Avoid logging sensitive tokens.
- **Write a Middleware to Console the JWT Token**: Debug tokens safely (avoid in production).
- **Need of Specifying Bearer in Token**: Standardize auth headers.
- **Updating Expired JWT Token**: Use refresh token to fetch new access token.
- **Protect Admin URLs Using JWT**: Restrict routes with token validation.
- **Sessions vs. Tokens**: Compare session-based vs. token-based auth.
- **Manage Sessions and Cookies**: Handle user sessions securely.
- **Login Request Contains a Token Before Login Happens**: Fix logic to avoid premature tokens.
- **Token Expiry Not Handled**: Implement refresh token logic.
- **Logout Doesn't Remove Cookies**: Clear cookies on logout.
- **If User is Logged In, Redirect to Login**: Fix redirect logic.
- **User Can Access Dashboard After Admin Deleted Account**: Validate tokens on access.
- **Don't Have Permission Page**: Show 403 page for unauthorized access.

10. API and Backend Integration

Integrate with backend APIs (Django) for the Todo app.

- **What is CORS?**: Handle cross-origin requests for API calls.
- **Use of express.json**: Parse JSON payloads in backend APIs.
- **Usage of Path Params in APIViews**: Access URL parameters in Django.
- **Reading Query Params**: Handle search queries in backend..
- **Move User List Search to Backend API**: Implement server-side search for users.
- **HTTP Status Codes (204)**: Understand 204 No Content for API responses.

11. Performance and Optimization

Optimize the Todo app for better performance.

- **useMemo**: Memoize computed values like filtered tasks.
- **Purpose of useCallback, useMemo, React.memo**: Prevent unnecessary re-renders.
- **Drawbacks of CSR**: Address SEO and initial load issues.
- **Code Splitting**: Reduce bundle size with dynamic imports.

- **Lazy Loading Code**: Load components on demand.
- **Dynamic Import**: Import modules dynamically for performance.
- **Suspense**: Handle loading states for lazy components.
- **React.memo**: Optimize TodoItem rendering.
- **How to Improve Performance in React Application**: Use memoization, lazy loading.
- **Optimization Techniques for Redux App**: Optimize selectors, reduce state updates.
- **React Profiler**: Debug performance bottlenecks.
- **Starvation**: Understand concurrent rendering issues.
- **Throttling**: Limit event frequency (e.g., search input).
- **Implement Debouncing**: Delay search input handling for performance.
- **JIT**: Understand Just-In-Time compilation in Vite.

12. Error Handling and Debugging

Handle errors and debug effectively.

- **Error Boundary**: Catch component errors to prevent app crashes.
- **Error Should Be Meaningful**: Display clear validation errors.
- **Use Logging Libraries**: Log errors safely (e.g., console.log alternatives).
- **How to Debug a Redux Application**: Use Redux DevTools for state/action tracking.

13. Todo App Specific Features

Implement required features for the Todo app.

- **Ask for Confirmation Before Deleting**: Use window.confirm for deletes.
- **Implement Pagination**: Paginate task list for scalability.

- **User Listing Shows Duplicate Entries**: Fix duplicate user display.
- **Edit/Delete User is Not Working**: Debug user management actions.
- **Edit/Create User from Admin Not Completed**: Complete admin user actions.

14. Advanced React Features

Enhance the Todo app with advanced techniques.

- **React Fragments**: Avoid extra DOM nodes in task list.
- **Why We Use Fragments**: Reduce DOM clutter.
- **Purpose of Fragments in React**: Cleaner markup.
- **React Portals**: Render modals outside DOM hierarchy.
- **dangerouslySetInnerHTML**: Handle raw HTML (use cautiously).
- **Conditional Rendering**: Show/hide UI based on state (e.g., errors).
- **Conditional Rendering Using && Syntax**: Use for simple conditions.
- **Conditional Rendering - Ways to Do It**: Explore &&, ternaries, if.
- **How Does Conditional Rendering Work?**: Toggle UI elements dynamically.
- **Create Element**: Use React.createElement (JSX alternative).

- **react.createElement vs. react.cloneElement**: Compare element creation.
- **Profile Placeholder**: Use default avatar for user profiles.
- **Setting Up ReactJS Environment with Vite**: Configure project setup.
- **List ul li Implementation**: Render task list with <ul>.
- **Implementation of Toggle Button**: Toggle task completion.
- **Write Code for Auto Increment/Decrement Counter with Range 0-10**: Practice counter logic.
- **Check Whether Text Entered in 2 Input Fields Are Matching or Not**: Validate form inputs.
- **Create a Timer Using useEffect**: Display task timestamps.
- **Create a Component to Add Two Numbers by Clicking a Button**: Practice event handling.
- **Entered Text Show It in an h1 Tag**: Display input dynamically.

15. Testing and Validation

Ensure robust functionality and validation.

- **How to Test Redux Reducers**: Write unit tests for reducers.
- **How to Handle ComponentDidUpdate in a Functional Component**: Use useEffect for updates.
- **Phases of State Updation**: Understand state update lifecycle.

- Task: Implement Search
- Subscribe
- Outlet
- Access Token, Refresh Tokens
- Limitations of Redux
- Limitations of Context Library
- Accessing and Modifying Specific States in Redux
- Redux Toolkit
- Generic Views in Django (Not React)
- What Determines the Lifetime of Access Token
- JWT Claims
- JWT Structure
- Security Offered by JWT
- Usage of Path Params in APIViews
- Reading Query Params
- Signup Input Validation
- Profile Info Edit (Optional)
- Edit User in Admin Side
- User Add Error Messages Not Communicated Properly
- Jwt.verify vs Decode
- useReducer vs Redux
- useContext
- Current React Version
- useCallback
- Disadvantages of Redux
- useTransition
- Use Hook
- useFormState
- JIT
- React Fibre
- Memoization (Not Clear)
- Optimistic Updates
- Selector in Redux
- Redux Store
- Single Source of Truth
- Why Actions Are Important
- Components of Redux
- Advantages and Disadvantages of Redux

- Purpose of Access Token and Refresh Token
- Use of Serializers
- Action Creator
- Combined Reducers
- Purpose of the Provider Component in React Redux
- What is Immer and How is it Used in Redux
- Create a To-Do App Using Redux Toolkit
- Implement Pagination
- Move User List Search to Backend API
- Shadow DOM
- Implement Debouncing
- Email Should Be Unique
- User Listing Shows Duplicate Entries
- Ask for Confirmation Before Delete
- Edit/Delete User is Not Working
- Do Not Print/Log Tokens in Code
- Authentication and Authorization
- JWT
- Redux Middleware
- Redux Thunk & Redux Saga
- Use Logging Libraries
- Error Should Be Meaningful
- Profile Placeholder
- File Type and File Size
- Don’t Have Permission Page
- Authorization
- Parts of Token
- Use of Refresh Token
- Principles of Redux
- Student Need to Understand the Flow of Redux, Dispatching the Action
- ReactDOMClient
- ReactDOMServer
- PropTypes
- Render Props
- HOC (Need Clarity)
- Suspense
- Redux vs Flux
- Controlled vs Uncontrolled Component
- Throttling
- Polyfills
- Error Boundary
- Redux Persistence
- Drawbacks of CSR
- Drawbacks of useMemo & React.memo
- React Portals
- Transpiler vs Compiler
- Function vs Custom Hook
- Dynamic Import

- useLayoutEffect
- React Concurrent
- Dynamic Routing
- Changing Value in useRef
- useRef vs Regular Variable to Hold Value
- Why Can’t Hooks Be Used in Conditions
- One Way Data Binding (Clarity)
- Unmounting in Functional Component
- Accessing Data from Uncontrolled Element
- Advantages of useRef
- ForwardRef
- useLocation
- Dynamic Rendering
- What Are React Mixins?
- Context API vs Redux
- Send Data from Child to Parent Using useRef
- Send Data from Child to Parent Using Callback Functions
- PropTypes in React: How to Validate Props?
- Which Lifecycle Hooks Are Replaced by useEffect in Functional Components?
- Create a Custom Hook for Increment/Decrement Counter
- Send Data from Child to Parent Using Context
- Purpose of the Callback Function in setState
- Example of Context API Usage
- Hook vs Function
- Custom Hook Scenario
- useMemo Dependency List
- Lazy Loading Code
- Hooks in Redux
- Redux Thunk
- Difference Between useEffect and useLayoutEffect
- Difference Between Context and Redux
- mapStateToProps
- mapDispatchToProps
- Replace useState by useReducer
- Need Practical Exposure of useMemo and useCallback
- Purpose of Middleware in Redux
- Code Splitting
- useMemo
- Shadow DOM
- dangerouslySetInnerHTML
- Rules of Using Hooks
- React Profiler
- Axios Interceptors
- Examples for HOF
- Pure Components
- Core Principles of Redux
- Synthetic Events
- Conditional Rendering

- Latest Features in React
- Automatic Batching
- Axios CancelToken
- Limitations of Functional Components
- Limitations of useMemo & useCallback
- Starvation
- Create Props Proxy for HOC
- Creating a Pure Component
- Purpose of Using Spread Operator When Updating (React)
- Flux
- Implementation Using useContext (Partial)
- useState Replace useReducer
- Create Custom Hook (Partial)
- ApplyMiddleware
- Actions
- useSelector
- Role of a Store
- How to Handle Asynchronous Operations in Redux
- How to Improve Performance of React Application
- JWT Working
- Write a Middleware to Console the JWT Token
- What is CORS?
- Use of express.json
- Jwt Decode vs Verify (Not Clear)
- Why We Keep the Component Name in Pascal Case in React?
- What Are the Phases of State Updation?
- use Redux DevTools
- Immer, Thunk
- HTTP Status Codes
- 204
- Learn About Redux Folder Structure (Large Scale Application)
- Redux-Saga
- Call()
- Put()
- Siblings Communication Using Middleware
- Semantic Kernel
- If User is Logged In, Redirect to Login
- Edit/Create User from Admin Not Completed
- User Can Access Dashboard After Admin Deleted Account
- Manage Sessions and Cookies
- Need of Specifying Bearer in Token
- Synchronous and Asynchronous Actions in Redux
- Redux and Redux Toolkit Difference
- configureStore
- Project is Incomplete
- Dispatcher
- Constants in Redux
- Redux Store Methods

- Sessions vs Tokens
- Explore the Use Cases of Browser Storages
- extraReducer
- LocalStorage vs Redux
- Hash Router & Memory Router
- How to Dispatch an Action
- Create an Action
- Jwt Verify
- When to Use Thunk and Saga
- Configure Store
- Updating Expired JWT Token
- Clear Redux State
- Call and Put in Redux Saga
- How Redux Ensures Components Updated Correctly
- How Should You Structure a Redux Application?
- Purpose of Selectors in Redux
- combineReducers Function in Redux
- How Can You Detect Token Expiry in a Frontend Application?
- Where Should Access and Refresh Tokens Be Stored on Client Side?
- Explore More in JWT
- Which is the Ideal Place to Store JWT Token
- What is JWT Token
- How to Store Data to Redux Store
- How to Debug a Redux Application
- Add Protected Routes
- Redux Slice
- Reconciliation
- Event Pooling
- Passing Data from Child to Parent (Scenarios)
- Principles of Redux
- Why Immutability Important in Redux, and How is it Achieved
- Flux Architecture
- Pros and Cons of Redux
- useDispatch Hook
- How Does Redux Persist Work?
- What is the Difference Between mapStateToProps and useSelector?
- Setting Up ReactJS Environment with Vite
- Introduction to Vite
- Advantages of Vite over CRA
- Understanding vite.config.js and Basic Project Structure
- DOM Concepts in JavaScript
- Difference Between Single Page and Multi-Page Application
- Concept of Reusability
- Learn About ES6 Versions Syntaxes
- Map
- Filter
- Reduce
- Import/Export

- Learn About Components
- Class (Basic - Lifecycle Methods)
- Functional
- Learn the Concept of Event Handlers
- onClick
- onChange
- React-Redux
- Action Creators and Action Types in Redux
- How to Test Redux Reducers
- Purpose of useCallback, useMemo, React.memo
- Actions and Action Creators
- Middleware in Redux
- What is Redux Toolkit
- How Does React-Redux's Provider Work
- Why Can't We Use LocalStorage Instead of Redux
- httpOnly Cookies
- Login Request Contains a Token Before Login Happens
- Token Expiry Not Handled
- Logout Doesn't Remove Cookies
- Implement a Reducer to Handle Multiple Action Types
- Concept of Immutable State and Why It is Important in Redux
- Common Performance Optimizations for Redux Applications
- How to Handle Impure Actions in Redux
- Side Effects
- Why is Reducer a Pure Function
- JWT Signature
- Practice Implementation of Combined Reducers
- Implement createAsyncThunk
- Core Components of Redux
- Store
- Reducers
- Dispatch vs Subscribe
- Where We Should Do API Call in Redux
- Write Code for Auto Increment/Decrement Counter with Range 0-10
- Redux Flow
- Optimization Techniques for Redux App
- Protect Admin URLs Using JWT
- Workout: Dispatching and Accessing State from Store
- Workout: Save the User Email in Redux Store & Render the Data in Component
- Workout: Update the Data Inside Redux Store by Dispatching an Action from a Component
- Siblings Communication in React00