DAY 28 – CONTEXT API

PROJECT 1 — "User Authentication & Role Management System"

Objective:

Build a **centralized authentication system** using Context API that manages user login, logout, and roles (admin/user).

Requirements:

1. **Auth Context**
    - Create AuthContext with values:
      { user, isAuthenticated, login(), logout() }
    - Store user info (name, email, role)
    - login() → updates context
    - logout() → clears context
2. **Protected Routes**
    - Wrap routes with an <AuthProvider>
    - Allow only authenticated users to access certain pages
    - Redirect unauthenticated users to /login
3. **Role-Based Access**
    - Admins can access /dashboard
    - Users can access /profile
4. **LocalStorage Sync**
    - Persist login data using useEffect inside context
5. **Logout Functionality**
    - Logout button updates context & clears local storage

Concepts Reinforced:
Creating and providing context, using useContext, updating global state, role-based access, local storage persistence.

PROJECT 2 — "E-Commerce Cart Management"

Objective:

Implement a **global shopping cart system** using Context API to manage cart items across all pages.

Requirements:

1. **Cart Context**
    - CartContext with state: { items, addToCart(), removeFromCart(), clearCart() }
    - items → array of product objects
2. **Adding Products**
    - Any "Add to Cart" button on a product card triggers addToCart(product)
3. **Global Cart Count**
    - Display total cart items in the Navbar dynamically (from Context)
4. **Remove & Clear**
    - On Cart Page:
        - "Remove" button → removeFromCart(id)
        - "Clear Cart" → clearCart()
5. **Persistent State**
    - Save cart data to localStorage
    - Load it back with useEffect when the app starts
6. **Checkout**
    - When "Checkout" is clicked, show alert and reset cart

Concepts Reinforced:
Global data sharing, context updates, derived values (cart total), persistent state, global UI sync.

PROJECT 3 — "Theme & Language Switcher Dashboard"

Objective:

Build a **global theme and language switcher** using multiple contexts.

Requirements:

1. **ThemeContext**
    - Values: { theme, toggleTheme() }
    - Default: "light"

- Toggle to “dark” and persist using localStorage

2. **LanguageContext**
    - Values: { language, changeLanguage() }
    - Default: “English”
    - Support: English, Tamil, Hindi, Spanish
3. **Integration**
    - Navbar shows “🌙 / ☀” for theme toggle
    - Dropdown for language selection
    - All pages adapt text & background based on context values
4. **Nested Contexts**
    - Wrap <ThemeProvider> and <LanguageProvider> around App
5. **UI Behavior**
    - Live theme switch without reload
    - Text changes instantly on language switch

Concepts Reinforced:
Multiple contexts, nested providers, context composition, persistent preferences, global UI sync.

PROJECT 4 — “Project Management Dashboard with Context + useReducer”

Objective:

Build a **task management system** where Context API + useReducer manage all task operations globally.

Requirements:

1. **TasksContext + Reducer**
    - state: { tasks: [], completedCount: 0 }
    - dispatch actions:
        - ADD_TASK, DELETE_TASK, TOGGLE_TASK, CLEAR_ALL
2. **Add Task Form**
    - Add new task → dispatch ADD_TASK
3. **Task List Component**
    - Each item shows “Mark Complete” button

- On click → dispatch TOGGLE_TASK

4. **Summary Bar**
    - Shows “Total Tasks” and “Completed Tasks” — derived from context
5. **Global Access**
    - Navbar, Sidebar, and Dashboard all read from the same context
6. **Reducer Logic**
    - All updates centralized in one reducer function

Concepts Reinforced:
Context + useReducer integration, state immutability, global actions, derived data, central state updates.

PROJECT 5 — “Music Player App with Global State”

Objective:

Create a **global music player system** using Context API to share playback data across components.

Requirements:

1. **PlayerContext**
    - state: { currentSong, isPlaying, playlist, volume }
    - Functions: playSong(song), pauseSong(), nextSong(), prevSong(), changeVolume()
2. **Global Playback Controls**
    - Footer (PlayerBar) shows current song name, play/pause buttons
    - Accessible across all pages
3. **Playlist Component**
    - Clicking a song triggers playSong() — updates global context
4. **Volume Control**
    - Range slider updates volume in context (two-way binding)
5. **Persist Current Song**
    - Save currentSong in localStorage, auto-restore on reload
6. **UI Update**
    - Play icon switches to pause dynamically (conditional rendering)

Concepts Reinforced:
Complex global state, multiple context functions, reactive UI updates, local storage sync, and real-time state sharing.
