Experiment No - 03

Roll No.	52
Name	Kshitij Nangare
Class	D15B
Subject	Full Stack Development
Lab Outcome	Implement state management and asynchronous data handling using React Hooks, Redux or Context API.
Date of Performance/ Submission	11/08/2025 18/08/2025
Signature & Grades	

Aim : Manage complex state with Redux or Context API

Theory :

In modern web applications, state refers to the data that defines the behavior and appearance of the application at a given time. Examples of state include:

- Current logged-in user information
- Items in a shopping cart
- Theme preference (light/dark mode)
- Data fetched from an API

When applications grow larger, managing state becomes complex because multiple components may need to access and update the same data. Passing props manually through many components (**prop drilling**) becomes inefficient and harder to maintain. To solve this, developers use **state management tools** like **Context API** and **Redux**.

1. Context API

The **Context API** is a built-in React feature that provides a way to share data globally without manually passing props through every component. It works by using:

- **React.createContext()** → Creates a context object.
- **Provider** → Wraps components and supplies the data.
- **useContext hook** → Consumes the data inside any child component.

Example: Sharing a theme (light/dark) using Context API:

```
// Create context
const ThemeContext = React.createContext();

// Provider
function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Navbar />
    </ThemeContext.Provider>
  );
}

// Consumer
function Navbar() {
  const theme = React.useContext(ThemeContext);
  return <h1>Current Theme: {theme}</h1>;
```

}

Here, the theme value is shared with the Navbar component without passing props explicitly.

Advantages of Context API:

- Simple to use (no extra library required).
- Great for small to medium applications.
- Reduces prop drilling.

Limitations:

- Not very efficient for very large apps.
- Can cause unnecessary re-renders when state updates frequently.

2. Redux

Redux is a powerful JavaScript library used for managing complex application state in a predictable way. It is commonly used in large React applications but can also be used with Angular, Vue, or vanilla JavaScript.

Key Concepts in Redux:

1. **Store** → Centralized container that holds the entire application state.
2. **Action** → Plain JavaScript objects that describe an event or change.
    - Example: { type: "ADD_ITEM", payload: { id: 1, name: "Laptop" } }
3. **Reducer** → A pure function that receives the current state and an action, then returns a new state.
4. **Dispatch** → A function used to send actions to the reducer.

Example: Simple Redux store for a counter:

```
// Reducer function
function counterReducer(state = { count: 0 }, action) {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
}
```

```
// Create store
const store = Redux.createStore(counterReducer);

// Dispatch actions
store.dispatch({ type: "INCREMENT" }); // count = 1
store.dispatch({ type: "DECREMENT" }); // count = 0
```

Here, Redux ensures that every state change is **predictable, traceable, and centralized**.

Advantages of Redux:

- Centralized state management.
- Predictable state transitions.
- Debugging with Redux DevTools.
- Scales well for large enterprise applications.

Limitations:

- Requires extra setup and boilerplate code.
- Learning curve for beginners.

Comparison

Feature	**Context API**	**Redux**
Complexity	Simple, built-in in React	More complex, requires setup
Best Use Case	Small-medium apps (auth, theme)	Large apps with complex states
Performance	May cause re-renders	Optimized with middleware
Debugging	Limited	Powerful with Redux DevTools

Source Code :

```
const Dashboard = () => {
  const { user, logout } = useAuthStore();
  const navigate = useNavigate();
  const [showProfileDropdown, setShowProfileDropdown] = useState(false);
  const [showProjectModal, setShowProjectModal] = useState(false);
  const [projects, setProjects] = useState([]);
  const [loading, setLoading] = useState(true);

  // Fetch projects from backend
  useEffect(() => {
    const fetchProjects = async () => {
      try {
        const res = await axiosInstance.get('/api/projects');
        setProjects(res.data.data || []);
      } catch (err) {
        console.error('Failed to fetch projects', err);
        toast.error(err.response?.data?.message || 'Failed to fetch projects');
      } finally {
        setLoading(false);
      }
    };

    fetchProjects();
  }, []);

  // Handle project status change
  const handleStatusChange = async (projectId, newStatus) => {
    try {
      await axiosInstance.put(`/api/projects/${projectId}`, { status: newStatus });
      setProjects(projects.map(p =>
        p._id === projectId ? { ...p, status: newStatus } : p
      ));
      toast.success(`Project marked as ${newStatus}`);
    } catch (err) {
      console.error('Failed to update project status', err);
      toast.error(err.response?.data?.message || 'Failed to update project');
    }
  };
```

```jsx
const Dashboard = () => {
    <main className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
      <div className="flex flex-col sm:flex-row justify-between items-start sm:items-center mb-8 gap-4">
        <h1 className="text-3xl font-extrabold text-gray-900">
          {user.role === 'developer' ? 'My Projects' : 'Projects Overview'}
        </h1>

        {(user.role === 'admin') && (
          <button
            onClick={() => setShowProjectModal(true)}
            className="flex items-center px-5 py-2.5 bg-blue-600 text-white rounded-lg text-base font-
          >
            <Plus size={20} className="mr-2" />
            New Project
          </button>
        )}
      </div>

      {projects.length === 0 && !loading && (
        <div className="bg-white p-8 rounded-lg shadow-md text-center text-gray-600">
          <p className="text-lg mb-4">No projects found !</p>
          {(user.role === 'admin') && (
            <button
              onClick={() => setShowProjectModal(true)}
              className="inline-flex items-center px-4 py-2 bg-blue-500 text-white rounded-md
            >
              <Plus size={18} className="mr-2" />
              Create Your First Project
            </button>
          )}
        </div>
      )}

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-8">
        {projects.map(project => (
          <ProjectCard
            key={project._id}
            project={project}
            role={user.role}
            currentUserId={user._id}
            onStatusChange={handleStatusChange}
            handleDeleteProject={handleDeleteProject}
          />
        ))}
      </div>
```

```jsx
// Project Card Component
const ProjectCard = ({ project, role, currentUserId, onStatusChange, handleDeleteProject }) => {
  const isLead = project.lead?._id === currentUserId;
  const isDeveloper = project.team.some(member => member._id === currentUserId);

  return (
    <div className="bg-white rounded-xl shadow-lg hover:shadow-xl transition-shadow duration-300 overflow-hidden">
      <div className="p-6">
        <div className="flex items-center justify-between mb-4">
          <h3 className="text-xl font-bold text-gray-900 leading-tight">
            <Link to={`/projects/${project._id}`} className="hover:text-blue-700 transition-colors duration-200">
              {project.name}
            </Link>
          </h3>
          <span className={`px-3 py-1 text-xs font-semibold rounded-full ${
            project.status === 'active'
              ? 'bg-emerald-100 text-emerald-800'
              : project.status === 'pending'
                ? 'bg-amber-100 text-amber-800'
                : 'bg-indigo-100 text-indigo-800' // completed
          }`}>
            {project.status}
          </span>
        </div>

        <p className="mt-2 text-sm text-gray-600 line-clamp-3 mb-2">
          {project.description}
        </p>


        <div className="grid grid-cols-1 sm:grid-cols-2 gap-y-3 gap-x-4 text-sm mb-6">
          <div className="flex items-center text-gray-700">
            <Calendar className="flex-shrink-0 h-5 w-5 text-blue-400 mr-2" />
            <div>
              <p className="text-xs text-gray-500">Deadline</p>
              <p className="font-medium">
                {new Date(project.deadline).toLocaleDateString()}
              </p>
            </div>
          </div>

          <div className="flex items-center text-gray-700">
            <Users className="flex-shrink-0 h-5 w-5 text-blue-400 mr-2" />
            <div>
              <p className="text-xs text-gray-500">Lead</p>
              <p className="font-medium">
```

```jsx
const ProjectCard = ({ project, role, currentUserId, onStatusChange, handleDeleteProject }) => {
                <div>
                  <p className="text-xs text-gray-500">Team</p>
                  <p className="font-medium">
                    {project.team.length} member{project.team.length !== 1 ? 's' : ''}
                  </p>
                </div>
              </div>

              <div className="flex items-center text-gray-700">
                <FileText className="flex-shrink-0 h-5 w-5 text-blue-400 mr-2" />
                <div>
                  <p className="text-xs text-gray-500">Documents</p>
                  <p className="font-medium">
                    {project.documentCount ?? 0} file{project.documentCount !== 1 ? 's' : ''}
                  </p>
                </div>
              </div>
            </div>

            <div className="flex flex-col sm:flex-row gap-3">
              <Link
                to={`/projects/${project._id}`}
                className="flex-1 inline-flex justify-center items-center px-4 py-2 border text-blue-300
              >
                <Users className="mr-2 h-4 w-4" />
                View Details
              </Link>


              {role === 'admin' && (
                <button
                  onClick={() => handleDeleteProject(project._id)}
                  className="flex-1 inline-flex justify-center items-center px-4 py-2 border text-red-300
                >
                  <Trash className="mr-2 h-4 w-4" />
                  Delete
                </button>
              )}
            </div>
          </div>
        </div>
      );
};
```

```jsx
import { Navigate, useLocation } from 'react-router-dom';  169.6k (gzipped: 53.5k)
import useAuthStore from '../stores/authStore';

const ProtectedRoute = ({ children, roles }) => {
  const { user, isAuthenticated, isLoading } = useAuthStore();
  const location = useLocation();

  if (isLoading) {
    return (
      <div className="flex justify-center items-center h-screen">
        <div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-indigo-500"></div>
      </div>
    );
  }

  if (!isAuthenticated) {
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  if (roles && !roles.includes(user?.role)) {
    return <Navigate to="/unauthorized" replace />;
  }

  return children;
};

export default ProtectedRoute;
```

```jsx
import { useEffect } from 'react';  4.7k (gzipped: 2k)
import { useNavigate } from 'react-router-dom';  169.5k (gzipped: 53.5k)
import useAuthStore from '../stores/authStore';

const AuthRedirect = () => {
  const { isAuthenticated, isLoading } = useAuthStore();
  const navigate = useNavigate();

  useEffect(() => {
    if (!isLoading) {
      navigate(isAuthenticated ? '/dashboard' : '/login', { replace: true });
    }
  }, [isAuthenticated, isLoading, navigate]);

  return null;
};

export default AuthRedirect;
```

```
const useAuthStore = create((set, get) => ({
  user: null,
  token: localStorage.getItem('token') || null,
  isLoading: false, // Changed initial state to false
  error: null,
  isAuthenticated: false,

  // Clear errors
  clearError: () => set({ error: null }),

  // Set auth token
  setAuthToken: (token) => {
    if (token) {
      axiosInstance.defaults.headers.common['Authorization'] = `Bearer ${token}`;
      localStorage.setItem('token', token);
      set({ token, isAuthenticated: true });
    } else {
      delete axiosInstance.defaults.headers.common['Authorization'];
      localStorage.removeItem('token');
      set({ token: null, isAuthenticated: false, user: null });
    }
  },

  // Login user
  login: async (email, password) => {
    set({ isLoading: true, error: null });
    try {
      const response = await axiosInstance.post('/api/auth/login', { email, password });

      get().setAuthToken(response.data.token);

      // Fetch user data
      const userResponse = await axiosInstance.get('/api/auth/me');
      set({
        user: userResponse.data.data.user,
        isAuthenticated: true,
        error: null
      });

      toast.success('Logged in successfully!');
      return { success: true };
    } catch (error) {
      const errorMsg = error.response?.data?.message || 'Login failed';
      set({ error: errorMsg, isAuthenticated: false });
      toast.error(errorMsg);
```

```
const useAuthStore = create((set, get) => ({
  login: async (email, password) => {
      return { success: false };
    } finally {
      set({ isLoading: false });
    }
  },

  // Logout user
  logout: () => {
    set({ user: null, isAuthenticated: false });
    get().setAuthToken(null);
    toast.success('Logged out successfully!');
  },

  // Load user on app start
  loadUser: async () => {
    const token = localStorage.getItem('token');
    if (!token) {
      set({ isLoading: false });
      return;
    }

    set({ isLoading: true });
    try {
      get().setAuthToken(token);
      const response = await axiosInstance.get('/api/auth/me');
      set({
        user: response.data.data.user,
        isAuthenticated: true
      });
    } catch (error) {
      get().logout();
    } finally {
      set({ isLoading: false });
    }
  },

  // Check if user has role
  hasRole: (role) => {
    return get().user?.role === role;
  }
}));

export default useAuthStore;
```

Output:

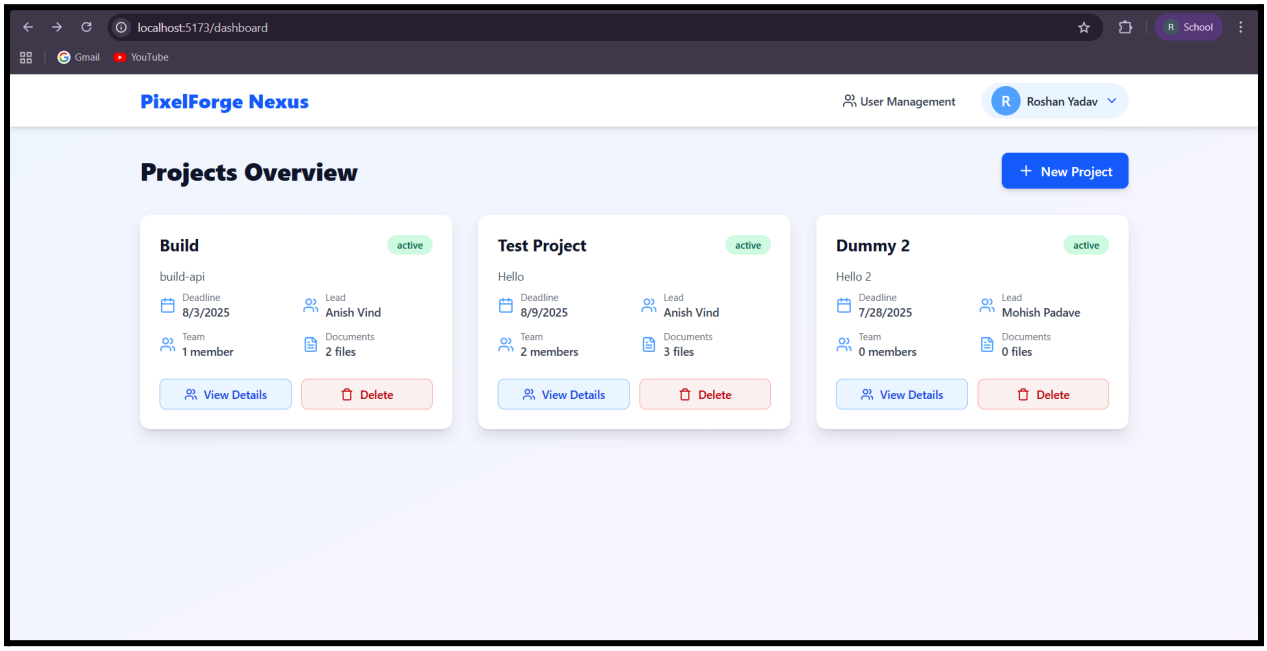

Figure 1

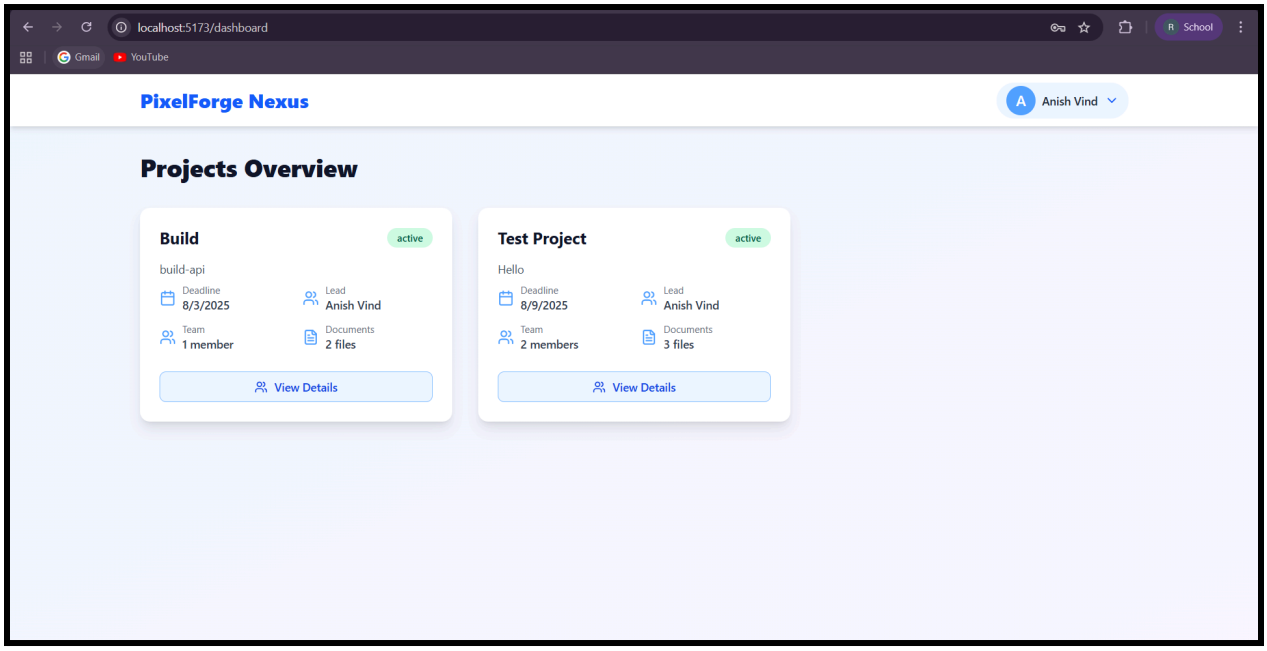

Figure 2

Extra Implementation (30% Additional Work)

As part of the extra work beyond the assigned experiment of managing complex state using **Redux/Context API**, I have also implemented **Zustand**, a modern and lightweight state management library for React.

About Zustand

- **Zustand** (German for *state*) is a small, fast, and scalable state management library.
- It provides a minimalistic API and avoids boilerplate code compared to Redux.
- It uses **hooks-based state management**, making it very easy to integrate into functional React components.
- Unlike Context API, Zustand does not suffer from unnecessary re-renders and scales well for both small and large applications.

Key Features of Zustand

1. **Simplicity** – Very easy to set up with just a few lines of code.
2. **Global State** – Can share state across components without prop drilling.
3. **Performance** – Optimized for fast updates without unnecessary component re-renders.
4. **No Boilerplate** – Unlike Redux, no need for actions, reducers, or dispatchers separately.

Benefits Over Context API/Redux

- **Less Code** → Store and actions are defined in one place.
- **Better Performance** → Selective state subscriptions avoid re-rendering unrelated components.
- **Ease of Use** → Easy to learn and integrate compared to Redux.

Conclusion

By implementing state management with **Redux** or **Context API**, developers can efficiently manage complex application states, avoid prop drilling, and ensure data consistency across components. The Context API is simple and best for small to medium projects, while Redux is more powerful and scalable for large applications. This experiment demonstrates how effective state management improves maintainability, scalability, and performance of full stack applications.