Experiment 3

1. Aim:

Manage Complex State with Redux or Context API

2. Essential Prerequisites:

1. JavaScript Fundamentals (ES6+):

Strong understanding of modern JavaScript features including arrow functions, destructuring, spread operators, array methods (map, filter, reduce), template literals, and async/await.

Ability to work with objects, arrays, and handle data manipulation efficiently.

2. React Core Concepts & Hooks:

Solid grasp of React fundamentals: components, JSX, props, state, and component lifecycle.

Proficiency with essential React hooks: useState for local state, useEffect for side effects, useContext for consuming context, and useReducer for complex state logic

3. State Management Understanding:

Clear understanding of application state, local vs. global state, and unidirectional data flow.

Knowledge of Context API basics including Provider/Consumer pattern and when to use Context vs. local state.

4. Development Environment Setup:

Node.js version 14+ and npm package manager installed and configured

Code editor (preferably VS Code) with React/JavaScript extensions and basic Git knowledge.

5. Component Architecture & Data Flow:

Understanding of component-based architecture, how components communicate, and separation of concerns.

Ability to plan component hierarchy, design data structures, and implement user interactions with proper state updates.

3. Theory:

a) Context API (React's Built-in Solution):

Context API is React's native state management solution that allows you to share data between components without explicitly passing props through every level of the component tree. It consists of three main parts: createContext(), Context.Provider, and useContext() hook. Context API is lightweight, doesn't require additional dependencies, and is perfect for simpler state management scenarios. It uses the Provider pattern where a context provider wraps components and makes data available to all child components through the useContext hook.

b) Redux (Third-party State Management):

Redux is a predictable state container for JavaScript applications that follows three core principles: single source of truth (all state in one store), state is read-only (only actions can change state), and changes are made with pure functions (reducers). Redux provides a centralized store, actions for describing state changes, reducers for handling state updates, and middleware for side effects. It's more powerful for complex applications with extensive state management needs, offering features like time-travel debugging, middleware ecosystem, and strict unidirectional data flow.

c) Redux Vs Context API:

Constraints	Redux	Context API
Approach to State Management	Redux uses a centralized store where actions and reducers modify the state.	Context API uses a decentralized approach where the state is passed through the component tree using provider and consumer.
Components	Store, Actions, and Reducers	Provider and Consumer
Performance	High performance with large applications.	Simple use and limited to small-scale applications.
Setup Requirements	Redux requires a prior setup to be effectively integrated with the React application.	Context API requires minimal setup.
Debugging	With effective Redux Dev Tools debugging becomes easier.	Debugging is difficult in the case of a nested component tree.
Integration	Can be easily integrated with other front-end frameworks.	Can be integrated natively with React components.
Ease of Use	Redux is suitable for experienced developers.	Can be integrated natively with React components
Size of Application	Redux is suitable for large and complex applications.	Context API is suitable for small applications.

d) Project Implementation:

This Movie Watchlist App demonstrates Context API implementation for managing complex state. The application uses useReducer hook combined with Context API to create a robust state management system. The MovieContext.js file serves as the central state container, managing watchlist, watched movies, search functionality, ratings, and reviews. The useReducer hook handles complex state logic with actions like ADD_MOVIE, MARK_AS_WATCHED, SET_RATING, and SET_REVIEW.

Components Breakdown with Examples:

AddMovie.js - This component provides a form interface for users to add new movies to their watchlist. It uses local state (useState) to manage the input field and calls the addMovie function from context when the form is submitted. Example: When a user types "Inception" and clicks "Add to Watchlist", the component dispatches an ADD_MOVIE action with the title, which gets added to the global watchlist state.

Watchlist.js - Displays all movies currently in the user's watchlist with options to mark them as watched or remove them. It maps through the watchlist array from context and renders each movie with action buttons. Example: Shows "Inception [Mark Watched] [Remove]" where clicking "Mark Watched" moves the movie from watchlist to watched section.

WatchedMovies.js - Renders all completed movies with their ratings and reviews. It includes the MovieRating and MovieReview components for each watched movie. Example: Displays "Inception'Mind-blowing sci-fi!' [Edit Rating] [Edit Review] [Remove]" showing the complete movie information.

SearchMovies.js - Provides real-time search functionality with suggestions dropdown. It filters movies as users type and shows clickable suggestions with status indicators. Example: Typing "Inc" shows "Inception 📺 Watchlist" as a clickable suggestion that auto-fills the search.

MovieRating.js - Interactive star rating component allowing users to rate watched movies from 1-5 stars. It manages temporary rating state and saves to global state when confirmed. Example: User clicks on 4th star, sees 4 filled stars, clicks "Save Rating" to permanently store the rating.

MovieReview.js - Text input component for adding personal reviews to watched movies. It provides a textarea interface with save/cancel functionality and displays existing reviews. Example: User writes "Amazing plot twists and great acting!" and saves it, creating a personal note about the movie.

Stats.js - Displays summary statistics about the movie collection including counts and average ratings. It calculates metrics from the global state and presents them in an organized layout. Example: Shows "4 In Watchlist, 6 Watched, 5 Rated, 4.2 Avg Rating" giving users a quick overview.

ProgressTracker.js - Visual progress indicator showing completion percentage with a color-coded progress bar and motivational messages. It calculates progress from total movies vs. watched count. Example: Displays "6 of 10 movies watched (60% completed)" with an orange progress bar and message " Great progress! You're doing well!"

The **App.jsx** wraps all components with MovieProvider, ensuring global state accessibility. The index.css provides responsive styling, while index.js serves as the application entry point. This implementation showcases how Context API can effectively manage complex state without external dependencies, demonstrating React's built-in capabilities for state management.

30% Extra Features (Bonus Implementation)

Local Storage Persistence

The app implements automatic data persistence using the browser's localStorage API, ensuring that all user data survives browser refreshes, restarts, and even computer restarts. This feature works by automatically saving the current application state to localStorage whenever any changes occur, and then retrieving and restoring this data when the app initializes. The implementation uses useEffect hooks that trigger on state changes to save data, and another useEffect on component mount to load saved data, creating a seamless user experience where data is never lost.

Where it's used in the project: This feature is implemented in the MovieContext.js file where two useEffect hooks handle the persistence. The first useEffect runs whenever state.watchlist or state.watched changes, automatically saving the current state to localStorage. The second useEffect runs once when the component mounts, retrieving any previously saved data and dispatching a LOAD_FROM_STORAGE action to restore the user's movie collection. This means users can close their browser, restart their computer, or refresh the page, and all their movies, ratings, and reviews will be preserved exactly as they left them.

Enhanced Interactive Features (Rating System, Reviews, Progress Tracking, and Advanced Search)

The app goes beyond basic functionality by implementing a comprehensive rating system with star ratings, detailed review capabilities, visual progress tracking, and intelligent search with real-time suggestions. These features transform the basic movie list into an engaging, personalized movie management system that provides users with rich interaction capabilities and meaningful feedback about their movie-watching habits. The rating system allows users to express their opinions about movies, the review system creates a personal movie journal, the progress tracker provides motivation and achievement tracking, and the enhanced search creates a modern, responsive user experience.

Where it's used in the project: The rating system is implemented in MovieRating.js where users can click on stars to rate movies, with the ratings stored in the global state and displayed with visual star indicators. The review system is handled by MovieReview.js which provides textarea inputs for writing detailed thoughts about movies. The progress tracking is implemented in ProgressTracker.js which calculates completion percentages and displays them with color-coded progress bars and motivational messages. The enhanced search functionality is in SearchMovies.js which provides real-time filtering, clickable suggestions, and search result counts. All these features work together through the centralized state management in MovieContext.js, where actions like SET_RATING, SET_REVIEW, and search filtering are handled by the reducer and made available to all components through the context provider.

4. CODE: - MovieContext.js

MovieContext.js

```
import React, { createContext, useContext, useReducer, useEffect } from 'react';

// Initial state
const initialState = {
  watchlist: [],
  watched: [],
  searchTerm: ''
};

// Action types
const ACTIONS = {
  ADD_MOVIE: 'ADD_MOVIE',
  REMOVE_FROM_WATCHLIST: 'REMOVE_FROM_WATCHLIST',
  MARK_AS_WATCHED: 'MARK_AS_WATCHED',
  REMOVE_FROM_WATCHED: 'REMOVE_FROM_WATCHED',
  SET_RATING: 'SET_RATING',
  SET_REVIEW: 'SET_REVIEW',
  SET_SEARCH_TERM: 'SET_SEARCH_TERM',
  LOAD_FROM_STORAGE: 'LOAD_FROM_STORAGE',
  CLEAR_WATCHLIST: 'CLEAR_WATCHLIST',
  CLEAR_WATCHED: 'CLEAR_WATCHED'
};

// Reducer function
const movieReducer = (state, action) => {
  switch (action.type) {
    case ACTIONS.ADD_MOVIE:
      const newMovie = {
        id: Date.now(),
        title: action.payload,
        rating: null,
        review: ''
      };
      return {
        ...state,
        watchlist: [...state.watchlist, newMovie]
      };

    case ACTIONS.REMOVE_FROM_WATCHLIST:
      return {
        ...state,
        watchlist: state.watchlist.filter(movie => movie.id !== action.payload)
      };

    case ACTIONS.MARK_AS_WATCHED:
      const movieToMove = state.watchlist.find(movie => movie.id === action.payload);
      return {
        ...state,
        watchlist: state.watchlist.filter(movie => movie.id !== action.payload),
        watched: [...state.watched, { ...movieToMove, rating: null, review: '' }]
      };

    case ACTIONS.REMOVE_FROM_WATCHED:
      return {
        ...state,
        watched: state.watched.filter(movie => movie.id !== action.payload)
      };

    case ACTIONS.SET_RATING:
      return {
        ...state,
        watched: state.watched.map(movie =>
          movie.id === action.payload.movieId
            ? { ...movie, rating: action.payload.rating }
            : movie
        )
      };

    case ACTIONS.SET_REVIEW:
      return {
        ...state,
        watched: state.watched.map(movie =>
          movie.id === action.payload.movieId
            ? { ...movie, review: action.payload.review }
            : movie
        )
      };

```

MovieContext.js

MovieContext.js > ...

```
const movieReducer = (state, action) => {
    case ACTIONS.SET_SEARCH_TERM:
      return {
        ...state,
        searchTerm: action.payload
      };

    case ACTIONS.LOAD_FROM_STORAGE:
      return {
        ...state,
        watchlist: action.payload.watchlist || [],
        watched: action.payload.watched || []
      };

    case ACTIONS.CLEAR_WATCHLIST:
      return {
        ...state,
        watchlist: []
      };

    case ACTIONS.CLEAR_WATCHED:
      return {
        ...state,
        watched: []
      };

    default:
      return state;
  }
};

// Create context
const MovieContext = createContext();

// Custom hook to use the context
export const useMovieContext = () => {
  const context = useContext(MovieContext);
  if (!context) {
    throw new Error('useMovieContext must be used within a MovieProvider');
  }
  return context;
};

// Provider component
export const MovieProvider = ({ children }) => {
  const [state, dispatch] = useReducer(movieReducer, initialState);

  // Load data from localStorage on component mount
  useEffect(() => {
    const savedData = localStorage.getItem('movieWatchlist');
    if (savedData) {
      try {
        const parsedData = JSON.parse(savedData);
        dispatch({ type: ACTIONS.LOAD_FROM_STORAGE, payload: parsedData });
      } catch (error) {
        console.error('Error loading data from localStorage:', error);
      }
    }
  }, []);

  // Save data to localStorage whenever state changes
  useEffect(() => {
    const dataToSave = {
      watchlist: state.watchlist,
      watched: state.watched
    };
    localStorage.setItem('movieWatchlist', JSON.stringify(dataToSave));
  }, [state.watchlist, state.watched]);

  // Action creators
  const addMovie = (title) => {
    if (title.trim()) {
      dispatch({ type: ACTIONS.ADD_MOVIE, payload: title.trim() });
    }
  };

  const removeFromWatchlist = (movieId) => {
    dispatch({ type: ACTIONS.REMOVE_FROM_WATCHLIST, payload: movieId });
  };
```

MovieContext.js

MovieContext.js > ...

```
122  export const MovieProvider = ({ children }) => {
156    };
157
158    const markAsWatched = (movieId) => {
159      dispatch({ type: ACTIONS.MARK_AS_WATCHED, payload: movieId });
160    };
161
162    const removeFromWatched = (movieId) => {
163      dispatch({ type: ACTIONS.REMOVE_FROM_WATCHED, payload: movieId });
164    };
165
166    const setRating = (movieId, rating) => {
167      dispatch({
168        type: ACTIONS.SET_RATING,
169        payload: { movieId, rating }
170      });
171    };
172
173    const setReview = (movieId, review) => {
174      dispatch({
175        type: ACTIONS.SET_REVIEW,
176        payload: { movieId, review }
177      });
178    };
179
180    const setSearchTerm = (term) => {
181      dispatch({ type: ACTIONS.SET_SEARCH_TERM, payload: term });
182    };
183
184    const clearWatchlist = () => {
185      if (window.confirm('Are you sure you want to clear your entire watchlist?')) {
186        dispatch({ type: ACTIONS.CLEAR_WATCHLIST });
187      }
188    };
189
190    const clearWatched = () => {
191      if (window.confirm('Are you sure you want to clear your entire watched list?')) {
192        dispatch({ type: ACTIONS.CLEAR_WATCHED });
193      }
194    };
195
196    // Filter movies based on search term
197    const filteredWatchlist = state.watchlist.filter(movie =>
198      movie.title.toLowerCase().includes(state.searchTerm.toLowerCase())
199    );
200
201    const filteredWatched = state.watched.filter(movie =>
202      movie.title.toLowerCase().includes(state.searchTerm.toLowerCase())
203    );
204
205    // Calculate progress
206    const totalMovies = state.watchlist.length + state.watched.length;
207    const watchedCount = state.watched.length;
208    const progressPercentage = totalMovies > 0 ? Math.round((watchedCount / totalMovies) * 100) : 0;
209
210    const value = {
211      watchlist: filteredWatchlist,
212      watched: filteredWatched,
213      searchTerm: state.searchTerm,
214      totalMovies,
215      watchedCount,
216      progressPercentage,
217      addMovie,
218      removeFromWatchlist,
219      markAsWatched,
220      removeFromWatched,
221      setRating,
222      setReview,
223      setSearchTerm,
224      clearWatchlist,
225      clearWatched
226    };
227
228    return (
229      <MovieContext.Provider value={value}>
230        {children}
231      </MovieContext.Provider>
232    );
233  };
```

App.jsx

MovieContext.js | App.jsx

App.jsx > default

```jsx
import React from 'react';
import { MovieProvider } from './context/MovieContext';
import AddMovie from './components/AddMovie';
import SearchMovies from './components/SearchMovies';
import Stats from './components/Stats';
import ProgressTracker from './components/ProgressTracker';
import Watchlist from './components/Watchlist';
import WatchedMovies from './components/WatchedMovies';

function App() {
  return (
    <MovieP (property) React.HTMLAttributes<HTMLDivElement>.className?: string | undefined
      <div className="container">
        <div className="header">
          <h1>🎬 Movie Watchlist App</h1>
          <p>Manage your movie collection with ease!</p>
        </div>

        <Stats />

        <ProgressTracker />

        <AddMovie />

        <SearchMovies />

        <Watchlist />

        <WatchedMovies />
      </div>
    </MovieProvider>
  );
}

export default App;
```

AddMovie.js

MovieContext.js | App.jsx | AddMovie.js

AddMovie.js > ...

```
import React, { useState } from 'react';
import { useMovieContext } from '../context/MovieContext';

const AddMovie = () => {
  const [movieTitle, setMovieTitle] = useState('');
  const { addMovie } = useMovieContext();

  const handleSubmit = (e) => {
    e.preventDefault();
    if (movieTitle.trim()) {
      addMovie(movieTitle);
      setMovieTitle('');
    }
  };

  return (
    <div className="section">
      <h2>Add New Movie</h2>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label htmlFor="movieTitle">Movie Title:</label>
          <input
            type="text"
            id="movieTitle"
            value={movieTitle}
            onChange={(e) => setMovieTitle(e.target.value)}
            placeholder="Enter movie title..."
            required
          />
        </div>
        <button type="submit" className="btn btn-primary">
          Add to Watchlist
        </button>
      </form>
    </div>
  );
};

export default AddMovie;

```

Watchlist.js

MovieContext.js | App.jsx | Watchlist.js

Watchlist.js > default

```jsx
import React from 'react';
import { useMovieContext } from '../context/MovieContext';

const Watchlist = () => {
  const { watchlist, markAsWatched, removeFromWatchlist, clearWatchlist } = useMovieContext();

  if (watchlist.length === 0) {
    return (
      <div className="section">
        <h2>My Watchlist</h2>
        <div className="empty-state">
          No movies in your watchlist. Add some movies to get started!
        </div>
      </div>
    );
  }

  return (
    <div className="section">
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', marginBottom: '15px' }}>
        <h2>My Watchlist ({watchlist.length})</h2>
        <button
          className="btn btn-danger"
          onClick={clearWatchlist}
          style={{ fontSize: '12px', padding: '8px 15px' }}
        >
          Clear All Watchlist
        </button>
      </div>

      {watchlist.map((movie) => (
        <div key={movie.id} className="movie-item">
          <div className="movie-info">
            <div className="movie-title">{movie.title}</div>
          </div>
          <div className="movie-actions">
            <button
              className="btn btn-success"
              onClick={() => markAsWatched(movie.id)}
            >
              Mark Watched
            </button>
            <button
              className="btn btn-danger"
              onClick={() => removeFromWatchlist(movie.id)}
            >
              Remove
            </button>
          </div>
        </div>
      ))}
    </div>
  );
};

export default Watchlist;
```

SearchMovies.js

MovieContext.js | App.jsx | Watchlist.js | SearchMovies.js

SearchMovies.js > ...

```
1  import React, { useState, useEffect } from 'react';
2  import { useMovieContext } from '../context/MovieContext';
3
4  const SearchMovies = () => {
5    const { searchTerm, setSearchTerm, watchlist, watched } = useMovieContext();
6    const [showSuggestions, setShowSuggestions] = useState(false);
7    const [suggestions, setSuggestions] = useState([]);
8
9    // Generate suggestions based on search term
10   useEffect(() => {
11     if (searchTerm.trim().length > 0) {
12       const allMovies = [...watchlist, ...watched];
13       const filtered = allMovies.filter(movie =>
14         movie.title.toLowerCase().includes(searchTerm.toLowerCase())
15       );
16       setSuggestions(filtered);
17       setShowSuggestions(true);
18     } else {
19       setSuggestions([]);
20       setShowSuggestions(false);
21     }
22   }, [searchTerm, watchlist, watched]);
23
24   const handleSuggestionClick = (movieTitle) => {
25     setSearchTerm(movieTitle);
26     setShowSuggestions(false);
27   };
28
29   const handleInputFocus = () => {
30     if (searchTerm.trim().length > 0 && suggestions.length > 0) {
31       setShowSuggestions(true);
32     }
33   };
34
35   const handleInputBlur = () => {
36     // Delay hiding suggestions to allow clicking on them
37     setTimeout(() => setShowSuggestions(false), 200);
38   };
39
40   const clearSearch = () => {
41     setSearchTerm('');
42     setSuggestions([]);
43     setShowSuggestions(false);
44   };
45
46   return (
47     <div className="section search-container">
48       <h2>Search Movies</h2>
49       <div className="form-group">
50         <label htmlFor="searchInput">Search by title:</label>
51         <div style={{ position: 'relative' }}>
52           <input
53             type="text"
54             id="searchInput"
55             value={searchTerm}
56             onChange={(e) => setSearchTerm(e.target.value)}
57             onFocus={handleInputFocus}
58             onBlur={handleInputBlur}
59             placeholder="Type to search movies..."
60             style={{ paddingRight: '40px' }}
61           />
62           {searchTerm && (
63             <button
64               type="button"
65               onClick={clearSearch}
66               style={{
67                 position: 'absolute',
68                 right: '10px',
69                 top: '50%',
70                 transform: 'translateY(-50%)',
71                 background: 'none',
72                 border: 'none',
73                 fontSize: '18px',
74                 cursor: 'pointer',
75                 color: '#999'
76               }}
77             >
78               ×
79             </button>
80           )}
```

MovieContext.js | App.jsx | Watchlist.js | SearchMovies.js

SearchMovies.js > ...

```
4   const SearchMovies = () => {
80          )}
81        </div>
82
83        {/* Search Results Summary */}
84        {searchTerm && (
85          <div style={{
86            marginTop: '10px',
87            fontSize: '14px',
88            color: '#666',
89            fontStyle: 'italic'
90          }}>
91            Found {suggestions.length} movie{suggestions.length !== 1 ? 's' : ''} matching "{searchTerm}"
92          </div>
93        )}
94
95        {/* Suggestions Dropdown */}
96        {showSuggestions && suggestions.length > 0 && (
97          <div className="suggestions-dropdown">
98            {suggestions.map((movie) => (
99              <div
100               key={movie.id}
101               className="suggestion-item"
102               onClick={() => handleSuggestionClick(movie.title)}
103             >
104               <span className="suggestion-title">{movie.title}</span>
105               <span className="suggestion-status">
106                 {watchlist.find(m => m.id === movie.id) ? '📺 Watchlist' : '✅ Watched'}
107               </span>
108             </div>
109           ))}
110         </div>
111       )}
112
113       {/* No Results Message */}
114       {searchTerm && suggestions.length === 0 && (
115         <div style={{
116           marginTop: '10px',
117           color: '#e74c3c',
118           fontStyle: 'italic'
119         }}>
120           No movies found matching "{searchTerm}"
121         </div>
122       )}
123     </div>
124   </div>
125 );
126 };
127
128 export default SearchMovies;
129
```

WatchedMovies.js

MovieContext.js | App.jsx | Watchlist.js | WatchedMovies.js

WatchedMovies.js > default

```jsx
import React from 'react';
import { useMovieContext } from '../context/MovieContext';
import MovieRating from './MovieRating';
import MovieReview from './MovieReview';

const WatchedMovies = () => {
  const { watched, removeFromWatched, clearWatched } = useMovieContext();

  if (watched.length === 0) {
    return (
      <div className="section">
        <h2>Watched Movies</h2>
        <div className="empty-state">
          No watched movies yet. Mark some movies as watched to see them here!
        </div>
      </div>
    );
  }

  return (
    <div className="section">
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', marginBottom: '15px' }}>
        <h2>Watched Movies ({watched.length})</h2>
        <button
          className="btn btn-danger"
          onClick={clearWatched}
          style={{ fontSize: '12px', padding: '8px 15px' }}
        >
          Clear All Watched
        </button>
      </div>

      {watched.map((movie) => (
        <div key={movie.id} className="movie-item">
          <div className="movie-info">
            <div className="movie-title">{movie.title}</div>
            <MovieRating
              movieId={movie.id}
              currentRating={movie.rating}
            />
            <MovieReview
              movieId={movie.id}
              currentReview={movie.review}
            />
          </div>
          <div className="movie-actions">
            <button
              className="btn btn-danger"
              onClick={() => removeFromWatched(movie.id)}
            >
              Remove
            </button>
          </div>
        </div>
      ))}
    </div>
  );
};

export default WatchedMovies;
```

Stats.js

MovieContext.js | App.jsx | Watchlist.js | WatchedMovies.js | *Stats.js*

Stats.js > ...

```
1   import React from 'react';
2   import { useMovieContext } from '../context/MovieContext';
3
4   const Stats = () => {
5     const { watchlist, watched } = useMovieContext();
6
7     // Calculate average rating
8     const ratedMovies = watched.filter(movie => movie.rating !== null);
9     const averageRating = ratedMovies.length > 0
10      ? (ratedMovies.reduce((sum, movie) => sum + movie.rating, 0) / ratedMovies.length).toFixed(1)
11      : 0;
12
13    return (
14      <div className="stats">
15        <div className="stat-item">
16          <span className="stat-number">{watchlist.length}</span>
17          <div className="stat-label">In Watchlist</div>
18        </div>
19        <div className="stat-item">
20          <span className="stat-number">{watched.length}</span>
21          <div className="stat-label">Watched</div>
22        </div>
23        <div className="stat-item">
24          <span className="stat-number">{ratedMovies.length}</span>
25          <div className="stat-label">Rated</div>
26        </div>
27        <div className="stat-item">
28          <span className="stat-number">{averageRating}</span>
29          <div className="stat-label">Avg Rating</div>
30        </div>
31      </div>
32    );
33  };
34
35  export default Stats;
36
```

5. OUTPUT:

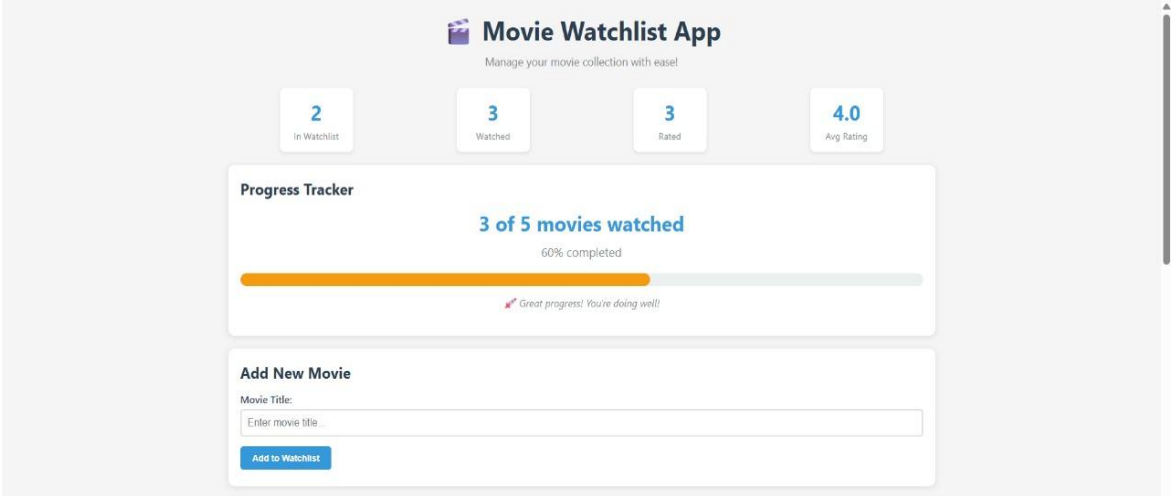

Fig1.

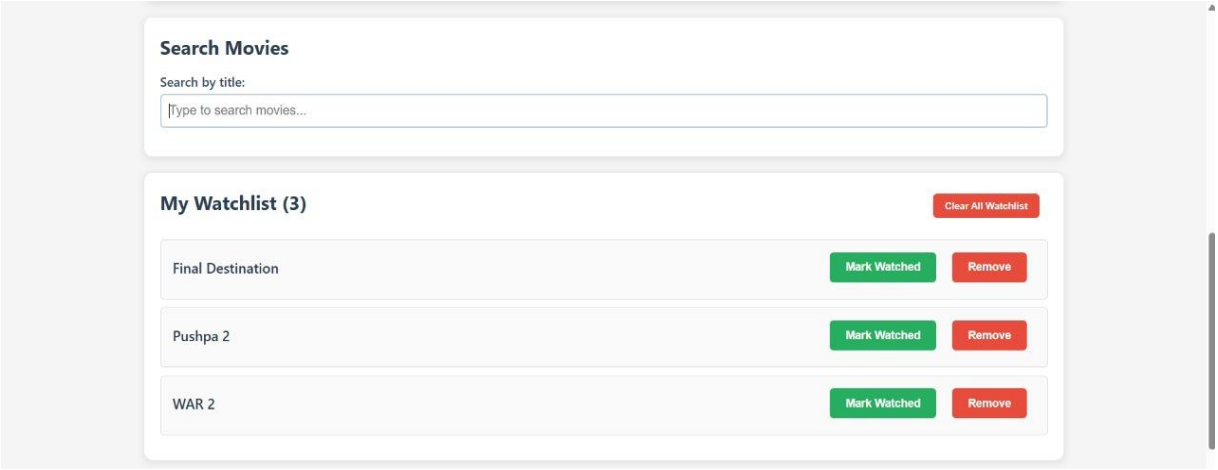

Fig2.

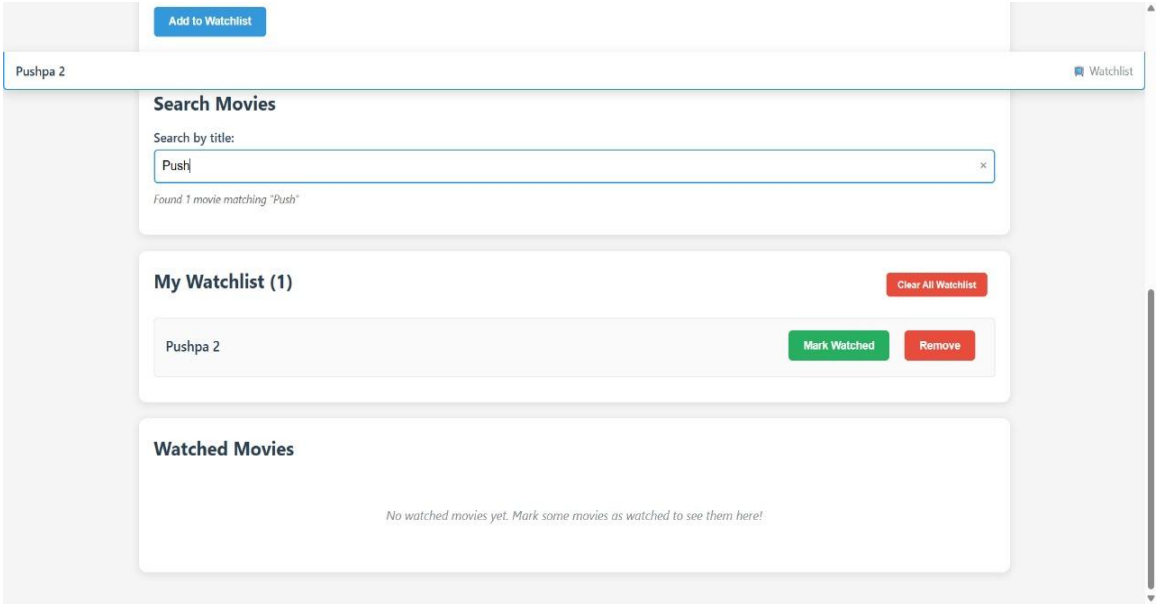
Add to Watchlist
Pushpa 2
Watchlist
Search Movies
Search by title:
Push
Found 1 movie matching "Push"
My Watchlist (1)
Clear All Watchlist
Pushpa 2
Mark Watched
Remove
Watched Movies
No watched movies yet. Mark some movies as watched to see them here!

Fig3.

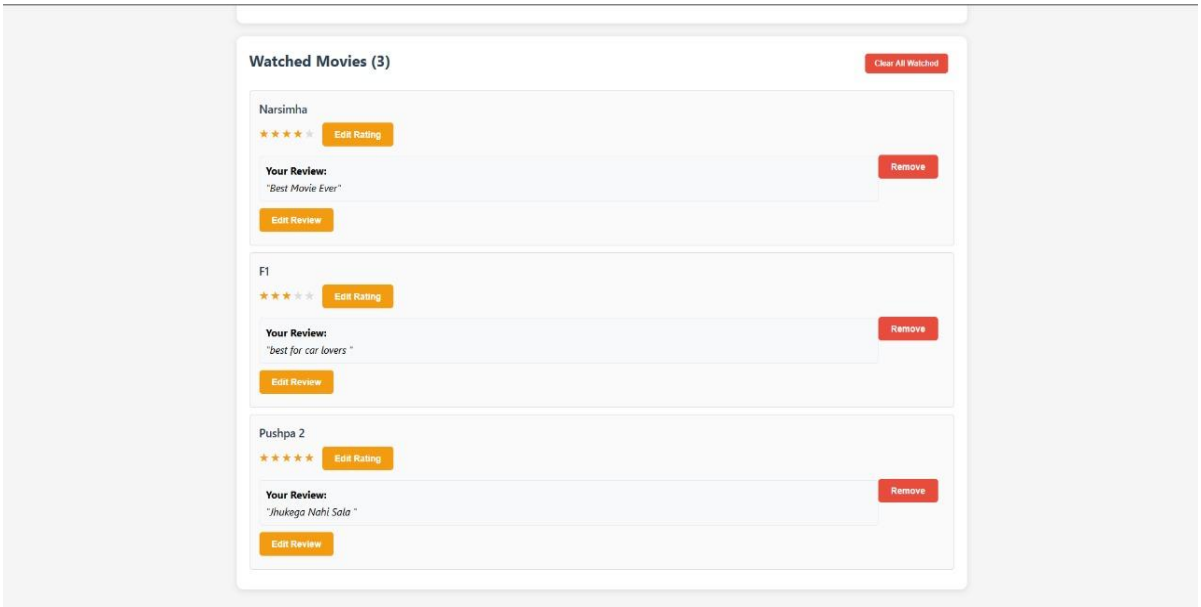
Watched Movies (3)
Clear All Watched
Narsimha
Edit Rating
Your Review:
"Best Movie Ever"
Remove
Edit Review
F1
Edit Rating
Your Review:
"best for car lovers "
Remove
Edit Review
Pushpa 2
Edit Rating
Your Review:
"Jhukega Nahi Sala "
Remove
Edit Review

Fig4.

6. Conclusion:

This experiment shows that using Context API with useReducer effectively manages complex state in React without external libraries like Redux. The Movie Watchlist App highlights how React's built-in tools handle nested data, real-time updates, user interactions, and persistence. While Redux is useful for very large apps, Context API offers simplicity, zero bundle size impact, and a smoother learning curve for most projects. The implementation demonstrates modern React practices, scalable architecture, and best practices. The added 30% features further showcase practical, production-ready applications of these state management concepts.