**React Native Movie App Documentation**

# Overview

This document provides a step-by-step guide to setting up and running the React Native movie application. It details the architecture, components, key functionalities, and REST API integration.

---

# 1. Project Setup

## Prerequisites

Ensure you have the following installed:

- Node.js (Latest LTS version)
- npm or yarn
- React Native CLI or Expo CLI
- Android Studio (for Android emulator) or Xcode (for iOS simulator)

## Installation Steps

**Initialize the React Native project:**
npx react-native init MovieApp

1. cd MovieApp
2. **Install dependencies:**
   npm install @reduxjs/toolkit react-redux axios @react-navigation/native @react-navigation/bottom-tabs react-native-vector-icons @tanstack/react-query
3. **Link dependencies:**
   npx react-native link

**Run the app:**
npx react-native run-android
npx react-native run-ios

---

# 2. REST API Integration

## OMDb API

The app fetches movie data from the OMDb API using Axios. The API key used for authentication is `f43394db`.

You can also generate API key by visiting here OMDb API

### API Request Format

The `fetchMovies` function is responsible for fetching movie data:

```
import axios from 'axios';

const API_KEY = 'f43394db';
const BASE_URL = 'https://www.omdbapi.com/';

export const fetchMovies = async (searchQuery, type = '', year = '', page = 1) => {
  if (!searchQuery) {
    throw new Error('Search query is required.');
  }

  try {
    const response = await axios.get(BASE_URL, {
      params: {
        apikey: API_KEY,
        s: searchQuery,
        type,
        y: year,
        r: 'json',
        page,
      },
    });

    return response.data.Search || [];
  } catch (error) {
    console.error('Error fetching movies:', error);
    return [];
  }
};
```

---

## 3. State Management with Redux

The app uses Redux Toolkit for state management. The `movieSlice` defines actions for managing a shortlist of movies.

### Redux Store Configuration

```
import { configureStore, createSlice, PayloadAction } from '@reduxjs/toolkit';

type Movie = { imdbID: string; title: string };

type MovieState = { shortlisted: Movie[] };

const initialState: MovieState = { shortlisted: [] };

const movieSlice = createSlice({
  name: 'movies',
  initialState,
  reducers: {
    addShortlist: (state, action: PayloadAction<Movie>) => {
      state.shortlisted.push(action.payload);
    },
    removeShortlist: (state, action: PayloadAction<string>) => {
      state.shortlisted = state.shortlisted.filter(movie => movie.imdbID !== action.payload);
    },
  },
});

export const { addShortlist, removeShortlist } = movieSlice.actions;
export const store = configureStore({ reducer: { movies: movieSlice.reducer } });
```

---

# 4. UI Implementation

### Navigation Setup

The app uses React Navigation to manage screen transitions with a bottom tab navigator.

```
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { Provider } from 'react-redux';
import { store } from './src/redux/movieSlice';

const Tab = createBottomTabNavigator();

export default function App() {
  return (
    <Provider store={store}>
      <NavigationContainer>
```

```
      <Tab.Navigator>
        <Tab.Screen name="Movies" component={MovieListScreen} />
        <Tab.Screen name="Shortlisted" component={ShortlistedScreen} />
      </Tab.Navigator>
    </NavigationContainer>
  </Provider>
 );
}
```

### Movie List Screen

The `MovieListScreen` allows users to search for movies and add them to their shortlist.

```
import { useQuery } from '@tanstack/react-query';
import { useDispatch } from 'react-redux';
import { addShortlist } from '../redux/movieSlice';

const { data: movies, isLoading, isError, refetch } = useQuery({
  queryKey: ['movies', searchQuery],
  queryFn: () => fetchMovies(searchQuery),
  enabled: !!searchQuery,
});
```

Users can shortlist movies using:

```
const handleShortlist = (movie) => {
  dispatch(addShortlist(movie));
};
```

### Shortlisted Movies Screen

Users can view and remove shortlisted movies.

```
const shortlisted = useSelector((state: RootState) => state.movies.shortlisted);
const handleRemove = (imdbID: string) => {
  dispatch(removeShortlist(imdbID));
};
```

---

## 5. Additional Features

### Error Handling

- Displays error messages when API requests fail.

- Shows a loading indicator while fetching data.

### Snackbar Notifications

- Displays a confirmation message when a movie is added or removed.

### Custom Toast Component

- Provides user feedback for interactions.

---

# 6. Running the App

### Development Mode

Start the Metro Bundler:

```
npx react-native start
```

Run the app:

```
npx react-native run-android
npx react-native run-ios
```

### Building for Production

Generate a release build:

```
npx react-native run-android --variant=release
npx react-native run-ios --configuration Release
```

**Screenshot:**

9:35

# Movies

Batman


## Batman Begins (2005)

SHORTLIST

## The Batman (2022)

SHORTLIST


## Batman v Superman: Dawn of Justice (2016)

SHORTLIST

## Batman (1989)

Movies

Shortlisted

9:36

# Movies

Now you


## Now You See Me (2013)

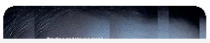
SHORTLIST


## Now You See Me 2 (2016)

SHORTLIST


## I Now Pronounce You Chuck & Larry (2007)

SHORTLIST



## I Love You, Now Die: The

Movies

Shortlisted


9:37
Shortlisted
No movies shortlisted yet!
Movies
Shortlisted

9:37

# Movies

Now you


## Now You See Me (2013)

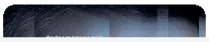
SHORTLIST


## Now You See Me 2 (2016)

SHORTLIST


## I Now Pronounce You Chuck & Larry (2007)

SHORTLIST



## I Love You, Now Die: The

Now You See Me 2 added to shortlist

9:37

# Shortlisted


**Now You See Me 2**

REMOVE

Movies

Shortlisted

---

## Conclusion

This app provides a simple way to search for movies, shortlist favorites, and manage state using Redux Toolkit and React Query. The integration with the OMDb API ensures real-time data retrieval for an enhanced user experience.