Movie Browser App - Theme Implementation Documentation ## Overview This document provides a comprehensive overview of the theme implementation and key features in the Movie Browser App, a React Native application built with Expo. The implementation focuses on creating a seamless, user-friendly experience while maintaining high performance and code quality. ## Theme Implementation ### Core Theme Features #### 1. Dynamic Theme Switching - System theme detection using `useColorScheme` - Manual theme toggle between light, dark, and system modes - Persistent theme preferences using AsyncStorage - Smooth transitions between themes using Animated API #### 2. Theme Colors | Mode | Background | Surface | Text | |------|------------|---------|------| | Dark Theme | `#121212` | `#1e1e1e` | `#ffffff` | | Light Theme | `#e0e0e0` | `#f5f5f5` | `#000000` | ### Implementation Details #### ThemeContext Architecture ```typescript interface ThemeContextType { theme: 'light' | 'dark' | 'system'; toggleTheme: () => void; isDarkMode: boolean; paperTheme: typeof MD3DarkTheme; navigationTheme: typeof DarkTheme; } ``` > The ThemeContext provides a centralized state management solution for theme-related functionality across the application. #### Theme Transitions The app implements smooth transitions between themes using React Native's Animated API: ```typescript React.useEffect(() => { Animated.sequence([Animated.timing(fadeAnim, { toValue: 0, duration: 150, useNativeDriver: true, }), Animated.timing(fadeAnim, { toValue: 1, duration: 150, useNativeDriver: true, }),]).start(); }, [isDarkMode]); ``` ## UI Components ### MovieCard Component #### Visual Design Features 1. **Responsive Layout** - Dynamic width calculation based on screen size - Adaptive grid system - Platform-specific shadows 2. **Animations** - Spring-based press animation - Scale transform (0.95) - Native driver enabled #### Theme Integration ```typescript const cardStyle = [styles.container, { backgroundColor: paperTheme.colors.surface, transform: [{ scale }], }, isDarkMode ? styles.darkCard : styles.lightCard,]; ``` ### Screen Implementations #### HomeScreen Features - Grid layout for movie cards - Search functionality with history - Pull-to-refresh implementation - Infinite scrolling - Theme-aware components #### MovieDetailsScreen Features - Rich movie information display - Dynamic backdrop and poster images - Theme-aware text and background colors - Favorite functionality integration ## Technical Implementation ### Best Practices 1. **Performance Optimization** - Native driver for animations - Memoized theme values - Efficient re-rendering strategies 2. **Code Organization** - Separation of concerns - Modular component structure - Consistent styling patterns 3. **User Experience** - Smooth transitions - Responsive layouts - Platform-specific adaptations ### Configuration The app's theme configuration in `app.json`: ```json { "expo": { "ios": { "userInterfaceStyle": "automatic" }, "android":

{ "userInterfaceStyle": "automatic" } } } ``` ## Future Enhancements ### 1. Theme Customization - User-defined color schemes - Custom theme presets - Advanced animation options ### 2. Performance Optimization - Cached theme values - Optimized re-renders - Reduced animation workload ### 3. Accessibility - Enhanced contrast ratios - Dynamic text sizing - Voice-over support ## Conclusion The theme implementation in the Movie Browser App demonstrates a robust, performant, and user-friendly approach to handling dynamic themes in React Native. The combination of system theme detection, smooth animations, and persistent preferences creates a polished user experience while maintaining code quality and maintainability.