100 React Native Technical Practice Questions

State Management with useState & useEffect

1. Display a list of users from an array using useState and make it scrollable with FlatList
2. Create a counter app that increments/decrements using useState with animation effects
3. Build a todo list where you can add, delete, and mark items as complete using useState
4. Implement a search functionality that filters a list of products in real-time using useState
5. Create a form with multiple input fields that updates state on each keystroke
6. Build a toggle switch that changes the theme (dark/light mode) using useState
7. Implement a timer that starts, stops, and resets using useState and useEffect
8. Create a shopping cart where you can add/remove items and calculate total price
9. Build an image gallery that displays different images based on selected category using useState
10. Implement a pagination component that loads more data when reaching the end of the list

Redux Toolkit Integration

11. Set up Redux Toolkit store and create a slice for managing user authentication state
12. Implement a global loading state using Redux Toolkit that shows/hides across different screens
13. Create a Redux slice for managing a shopping cart with add, remove, and update quantity actions
14. Build a user profile management system using Redux Toolkit with async thunks for API calls
15. Implement a notification system using Redux Toolkit that manages multiple toast messages
16. Create a favorites list functionality where users can add/remove items using Redux
17. Build a multi-step form that saves progress in Redux store between screen navigations
18. Implement a global error handling system using Redux Toolkit
19. Create a settings screen that saves user preferences to Redux store and AsyncStorage
20. Build a real-time chat message system using Redux Toolkit with WebSocket integration

Navigation & Screen Management

21. Implement stack navigation with custom headers and navigation parameters
22. Create a bottom tab navigator with badge counts that update based on Redux state
23. Build a drawer navigation with user profile information displayed in the drawer
24. Implement deep linking that navigates to specific screens with parameters
25. Create a modal screen that passes data back to the previous screen when closed
26. Build a nested navigation structure (Stack inside Tab inside Drawer)

27. Implement screen transitions with custom animations between different screens
28. Create a splash screen that checks authentication status before navigating to main app
29. Build a onboarding flow with multiple screens and skip/next functionality
30. Implement conditional navigation based on user roles (admin vs regular user)

API Integration & Data Fetching

31. Fetch data from REST API using useEffect and display it in a FlatList with loading states
32. Implement infinite scrolling that loads more data when user reaches the end
33. Create a search functionality that debounces API calls to avoid excessive requests
34. Build offline data caching using AsyncStorage when API calls fail
35. Implement pull-to-refresh functionality for updating data from API
36. Create a retry mechanism for failed API calls with exponential backoff
37. Build a image upload functionality with progress indicator using FormData
38. Implement optimistic updates that immediately show changes before API confirmation
39. Create a data synchronization system that handles conflicts between local and remote data
40. Build a real-time data updates system using WebSocket or Server-Sent Events

Custom Components & Reusability

41. Create a reusable custom button component with different variants (primary, secondary, outline)
42. Build a custom input component with validation, error states, and different input types
43. Implement a reusable modal component that can display different content types
44. Create a custom loading spinner component with different sizes and colors
45. Build a reusable card component that adapts its layout based on content type
46. Implement a custom slider component for image carousels with dot indicators
47. Create a reusable dropdown/picker component with search functionality
48. Build a custom tab component that works independently of navigation
49. Implement a reusable progress bar component with animation and customizable styles
50. Create a custom calendar component with date selection and event marking

Performance Optimization

51. Optimize FlatList performance using getItemLayout, keyExtractor, and removeClippedSubviews
52. Implement lazy loading for images in a gallery using React.lazy or custom logic
53. Use React.memo to prevent unnecessary re-renders of list items
54. Implement virtual scrolling for large datasets to improve memory usage

55. Optimize Redux selectors using createSelector to prevent unnecessary re-renders
56. Build a image caching system that stores images locally after first load
57. Implement code splitting to reduce initial bundle size
58. Use useMemo and useCallback to optimize expensive calculations and functions
59. Create a debounced search input that minimizes API calls and state updates
60. Implement background task handling for data synchronization

Animations & Gestures

61. Create smooth transitions using React Native Animated API for button press effects
62. Implement a swipe-to-delete functionality for list items using PanGestureHandler
63. Build a animated drawer that slides in/out with gesture controls
64. Create a parallax scrolling effect for a profile screen header
65. Implement a pull-down animation for custom refresh functionality
66. Build a animated bottom sheet that responds to drag gestures
67. Create smooth page transitions using shared element animations
68. Implement a animated progress indicator for multi-step processes
69. Build a custom animated loading screen with multiple moving elements
70. Create gesture-based image zoom and pan functionality

Storage & Data Persistence

71. Implement user preferences storage using AsyncStorage with JSON serialization
72. Create a offline-first todo app that syncs with server when connection is available
73. Build a caching layer that stores API responses with expiration times
74. Implement secure storage for sensitive data like authentication tokens
75. Create a data backup and restore functionality using cloud storage
76. Build a database layer using SQLite for complex relational data
77. Implement automatic data cleanup to prevent storage from growing indefinitely
78. Create a migration system for handling app updates with data structure changes
79. Build a conflict resolution system for data synchronization between devices
80. Implement a audit log system that tracks user actions locally

Testing & Error Handling

81. Write unit tests for custom hooks using React Native Testing Library
82. Create integration tests for Redux actions and reducers

83. Implement comprehensive error boundaries that catch and display user-friendly errors
84. Build a crash reporting system that logs errors with user context
85. Create mock API responses for testing different data scenarios
86. Implement form validation with real-time error messages and field highlighting
87. Build a network connectivity detector that handles offline scenarios gracefully
88. Create automated tests for navigation flows between screens
89. Implement screenshot testing for visual regression detection
90. Build a performance monitoring system that tracks app metrics

Advanced Features & Integrations

91. Implement push notifications with different types and action handling
92. Create a biometric authentication system using device fingerprint/face recognition
93. Build a camera integration with photo capture, gallery access, and image processing
94. Implement geolocation tracking with map integration and location-based features
95. Create a file system integration for document viewing and sharing
96. Build a social media sharing integration with multiple platforms
97. Implement in-app purchase functionality with receipt validation
98. Create a background job scheduler for periodic data synchronization
99. Build a custom keyboard extension or input method integration
100. Implement a comprehensive analytics system that tracks user behavior and app performance

Bonus Challenge Questions

Each question above can be extended with these additional challenges:

- Add proper TypeScript typing for all components and functions
- Implement comprehensive error handling with user-friendly messages
- Add loading states and skeleton screens for better user experience
- Include accessibility features (screen reader support, keyboard navigation)
- Optimize for both iOS and Android platform-specific behaviors
- Add proper logging and debugging capabilities
- Implement proper memory management to prevent leaks
- Add comprehensive unit and integration tests