React Native Coding Challenges

Interview Questions and Answers

Core Concepts

This section focuses on fundamental principles and advanced concepts that an experienced developer should master.

1. Implement a custom hook for handling deep linking in React Native with TypeScript

Solution:

```
const useDeepLink = () => {
  const [link, setLink] = useState(null);
  useEffect(() => {
    const handleLink = ({url}: {url: string}) => setLink(url);
    Linking.addEventListener('url', handleLink);
    return () => Linking.removeEventListener('url', handleLink);
  }, []);
  return link;
}
```

Key points:

- Handles deep linking events
- Properly typed with TypeScript
- Cleanup on unmount

2. Create a performant infinite scroll implementation with React Native FlatList

Solution:

```
const InfiniteList = () => {
  const [data, setData] = useState([]);
  const loadMore = useCallback(() => {
    fetchMoreData().then(newItems =>
      setData(prev => [...prev, ...newItems]));
  }, []);
  return ;
}
```

3. Implement a custom gesture handler for a swipeable card component

Solution:

```
const SwipeableCard = () => {
  const pan = useRef(new Animated.ValueXY()).current;
  const panResponder = PanResponder.create({
    onMoveShouldSetPanResponder: () => true,
    onPanResponderMove: Animated.event(
      [null, {dx: pan.x, dy: pan.y}],
      {useNativeDriver: false}
    )
  });
```

4. Design a reusable form validation hook with error handling

Solution:

```
const useFormValidation = (initialState, validationRules) => {
  const [values, setValues] = useState(initialState);
  const [errors, setErrors] = useState({});
  const validate = useCallback(() => {
    const newErrors = {};
```

```
    Object.keys(validationRules).forEach(key => {
      const error = validationRules[key](values[key]);
      if (error) newErrors[key] = error;
    });
    setErrors(newErrors);
  }, [values]);
```

5. Create a custom navigation animation using React Native Reanimated

Solution:

```
const CustomTransition = () => {
  const progress = useSharedValue(0);
  const animatedStyle = useAnimatedStyle(() => ({
    transform: [{
      translateX: interpolate(progress.value,
        [0, 1], [300, 0])
    }]
  }));
```

6. Implement a performant real-time search with debouncing

Solution:

```
const useDebounceSearch = (searchFn, delay = 300) => {
  const [query, setQuery] = useState('');
  useEffect(() => {
    const handler = setTimeout(() => searchFn(query), delay);
    return () => clearTimeout(handler);
  }, [query, delay, searchFn]);
  return [query, setQuery];
};
```

7. Create a custom hook for handling offline storage synchronization

Solution:

```
const useSyncStorage = () => {
  const [syncQueue, setSyncQueue] = useState([]);
  const sync = useCallback(async () => {
    const connected = await NetInfo.fetch();
    if (connected.isConnected) {
      await Promise.all(syncQueue.map(item => syncItem(item)));
      setSyncQueue([]);
    }
  }, [syncQueue]);
```

8. Implement a custom image lazy loading component with placeholder

Solution:

```
const LazyImage = ({ uri, placeholder }) => {
  const [loaded, setLoaded] = useState(false);
  const opacity = useRef(new Animated.Value(0)).current;
  const onLoad = () => {
    Animated.timing(opacity, {
      toValue: 1,
      duration: 300,
      useNativeDriver: true
    }).start();
  };
```

9. Create a custom hook for handling app state and background tasks

Solution:

```
const useAppState = (onBackground, onForeground) => {
```

```
  useEffect(() => {
    const subscription = AppState.addEventListener(
      'change',
      nextState => {
        if (nextState === 'background') onBackground();
        if (nextState === 'active') onForeground();
      }
    );
```

10. Implement a custom navigation guard with authentication check

Solution:

```
const useAuthGuard = (navigation) => {
  const { isAuthenticated } = useAuth();
  useEffect(() => {
    if (!isAuthenticated) {
      navigation.reset({
        index: 0,
        routes: [{ name: 'Login' }]
      });
    }
  }, [isAuthenticated]);
```

Data Structures and Algorithms

Questions in this section test your understanding of how to work with and manipulate data efficiently.

1. Implement an LRU Cache in React Native with a fixed size of N

Solution:

We can implement an LRU Cache using a Map and maintain size:

```
class LRUCache {
  constructor(capacity) {
    this.cache = new Map();
    this.capacity = capacity;
  }
  get(key) {
    if (!this.cache.has(key)) return -1;
    const value = this.cache.get(key);
    this.cache.delete(key);
    this.cache.set(key, value);
    return value;
  }
  put(key, value) {
    if (this.cache.has(key)) this.cache.delete(key);
    else if (this.cache.size >= this.capacity) {
      this.cache.delete(this.cache.keys().next().value);
    }
    this.cache.set(key, value);
  }
}
```

Time Complexity: O(1) for both get and put operations

2. Implement a function to find pairs in an array that sum to a target value

Solution:

```
const findPairs = (arr, target) => {
  const seen = new Set();
  const result = [];
  arr.forEach(num => {
    const complement = target - num;
    if (seen.has(complement)) result.push([num, complement]);
    seen.add(num);
  });
  return result;
}
```

Time Complexity: O(n) where n is the array length

Space Complexity: O(n) for the hash set

3. Implement a sliding window maximum for a given array and window size

Solution:

```
const maxSlidingWindow = (nums, k) => {
  const result = [];
  const deque = [];
  for (let i = 0; i < nums.length; i++) {
```

```
    while (deque.length && nums[deque[deque.length-1]] <= nums[i])
      deque.pop();
    deque.push(i);
    if (deque[0] <= i - k) deque.shift();
    if (i >= k - 1) result.push(nums[deque[0]]);
  }
  return result;
}
```

Time Complexity: O(n)

4. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time

Solution:

```
class MinStack {
  constructor() {
    this.stack = [];
    this.minStack = [];
  }
  push(x) {
    this.stack.push(x);
    if (!this.minStack.length || x <= this.minStack[this.minStack.length-1])
      this.minStack.push(x);
  }
  pop() {
    if (this.stack.pop() === this.minStack[this.minStack.length-1])
      this.minStack.pop();
  }
  top() { return this.stack[this.stack.length-1]; }
  getMin() { return this.minStack[this.minStack.length-1]; }
}
```

Time Complexity: O(1) for all operations

5. Implement a function to detect a cycle in a linked list

Solution:

```
const hasCycle = head => {
  let slow = head, fast = head;
  while (fast && fast.next) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow === fast) return true;
  }
  return false;
}
```

Time Complexity: O(n)

Space Complexity: O(1)

Uses Floyd's Cycle-Finding Algorithm (tortoise and hare)

6. Implement a function to serialize and deserialize a binary tree

Solution:

```
const serialize = root => {
  if (!root) return 'null';
  return `${root.val},${serialize(root.left)},${serialize(root.right)}`;
};
const deserialize = data => {
  const nodes = data.split(',');
  let index = 0;
  const dfs = () => {
```

```
    if (nodes[index] === 'null') { index++; return null; }
    const node = { val: parseInt(nodes[index++]) };
    node.left = dfs();
    node.right = dfs();
    return node;
  };
  return dfs();
}
```

Time Complexity: O(n) for both operations

7. Implement a trie (prefix tree) with insert, search, and startsWith methods

Solution:

```
class Trie {
  constructor() {
    this.root = {};
  }
  insert(word) {
    let node = this.root;
    for (let char of word) {
      node[char] = node[char] || {};
      node = node[char];
    }
    node.isEnd = true;
  }
  search(word) {
    const node = this.traverse(word);
    return node !== null && node.isEnd === true;
  }
  startsWith(prefix) {
    return this.traverse(prefix) !== null;
  }
  traverse(word) {
    let node = this.root;
    for (let char of word) {
      if (!node[char]) return null;
      node = node[char];
    }
    return node;
  }
}
```

Time Complexity: O(m) where m is the length of the word

8. Implement a function to find the longest palindromic substring

Solution:

```
const longestPalindrome = s => {
  let start = 0, maxLength = 1;
  const expandAroundCenter = (left, right) => {
    while (left >= 0 && right < s.length && s[left] === s[right]) {
      const currLength = right - left + 1;
      if (currLength > maxLength) {
        start = left;
        maxLength = currLength;
      }
      left--; right++;
    }
  };
  for (let i = 0; i < s.length; i++) {
    expandAroundCenter(i, i);
    expandAroundCenter(i, i + 1);
  }
  return s.substring(start, start + maxLength);
}
```

Time Complexity: O(n²)

9. Implement a function to merge k sorted linked lists

Solution:

```
const mergeKLists = lists => {
  if (!lists.length) return null;
  const merge = (l1, l2) => {
    if (!l1 || !l2) return l1 || l2;
    const dummy = { next: null };
    let curr = dummy;
    while (l1 && l2) {
      if (l1.val <= l2.val) {
        curr.next = l1;
        l1 = l1.next;
      } else {
        curr.next = l2;
        l2 = l2.next;
      }
      curr = curr.next;
    }
    curr.next = l1 || l2;
    return dummy.next;
  };
  while (lists.length > 1) {
    const merged = [];
    for (let i = 0; i < lists.length; i += 2) {
      merged.push(merge(lists[i], lists[i + 1] || null));
    }
    lists = merged;
  }
  return lists[0];
}
```

Time Complexity: O(N log k) where N is total nodes and k is number of lists

10. Implement a function to find the kth largest element in an array

Solution:

```
const findKthLargest = (nums, k) => {
  const quickSelect = (left, right, k) => {
    const pivot = nums[right];
    let p = left;
    for (let i = left; i < right; i++) {
      if (nums[i] <= pivot) {
        [nums[p], nums[i]] = [nums[i], nums[p]];
        p++;
      }
    }
    [nums[p], nums[right]] = [nums[right], nums[p]];
    if (p === nums.length - k) return nums[p];
    if (p > nums.length - k) return quickSelect(left, p - 1, k);
    return quickSelect(p + 1, right, k);
  };
  return quickSelect(0, nums.length - 1, k);
}
```

Time Complexity: Average O(n), Worst O(n²)

System Design

These questions evaluate your ability to think about the bigger picture, including architecture, scalability, and performance.

1. Design a scalable real-time chat system for React Native. What architecture and components would you use?

Key Components:

- **Frontend:** React Native app with WebSocket connection
- **Backend:** Node.js with Socket.io for real-time communication
- **Database:** MongoDB for messages, Redis for user sessions
- **Message Queue:** RabbitMQ for handling high message volumes

Architecture:

- WebSocket connection maintained between app and server
- Messages stored in MongoDB with TTL index
- Redis pub/sub for scaling across multiple nodes
- Load balancer with sticky sessions
- Message encryption using end-to-end encryption

```
const socket = io('wss://chat.example.com');
socket.on('message', (data) => {
  dispatch({ type: 'NEW_MESSAGE', payload: data });
});
```

2. How would you design an offline-first React Native app with data synchronization?

Key Components:

- **Local Storage:** AsyncStorage or SQLite
- **State Management:** Redux + Redux Persist
- **Sync Queue:** Background queue for pending operations
- **Conflict Resolution:** CRDT or version vectors

Implementation:

- Store data locally first, then sync to server
- Queue offline changes for later sync
- Use optimistic UI updates
- Implement retry mechanism with exponential backoff

```
const syncQueue = new Queue({
  storage: AsyncStorage,
  retry: { attempts: 3, backoff: 'exponential' }
});
```

3. Design a scalable image processing and caching system for a React Native social media app

Architecture Components:

- **CDN:** CloudFront for image delivery
- **Storage:** S3 for original images
- **Processing:** Lambda functions for resizing
- **Cache:** Multi-level caching strategy

Implementation:

- Generate multiple image sizes on upload
- Use progressive loading technique
- Implement local image caching
- Use blur hash for placeholders

```
const ImageCache = {
  get: async (url, size) => {
    const cached = await AsyncStorage.getItem(`img_${url}_${size}`);
    return cached || fetchAndCache(url, size);
  }
};
```

4. How would you implement a reliable push notification system for a React Native app at scale?

System Components:

- **Push Services:** Firebase Cloud Messaging (FCM)
- **Queue:** SQS for notification processing
- **Database:** DynamoDB for device tokens
- **Analytics:** Track delivery and open rates

Key Features:

- Token refresh mechanism
- Batch processing for mass notifications
- Fallback to in-app notifications
- Rate limiting and throttling

```
const pushService = {
  send: async (tokens, message) => {
    return await fcm.sendMulticast({ tokens, ...message });
  }
};
```

5. Design a performant infinite scroll feed system for React Native

Architecture:

- **Frontend:** FlatList with virtualization
- **Backend:** Cursor-based pagination
- **Cache:** Redis for feed caching
- **Database:** Sharded MongoDB

Optimization Techniques:

- Windowing for memory efficiency
- Pre-fetching next page
- Placeholder loading states
- Debounced scroll events

```
const Feed = () => (

);
```

6. How would you implement a secure authentication system for a React Native app?

Security Components:

- **Auth Flow:** OAuth 2.0 with PKCE
- **Storage:** Encrypted KeyChain/KeyStore
- **Tokens:** JWT with refresh mechanism
- **Biometrics:** Face ID/Touch ID integration

Implementation:

- Secure token storage
- Certificate pinning

- Automatic token refresh
- Session management

```
const secureStore = {
  save: async (key, value) => {
    await Keychain.setGenericPassword(key, value, {
      accessControl: ACCESS_CONTROL.BIOMETRY_ANY
    });
  }
};
```

7. Design a real-time location tracking system for a React Native delivery app

System Components:

- **Location Updates:** Background location services
- **Real-time:** WebSocket/MQTT
- **Storage:** MongoDB with geospatial indexes
- **Processing:** Redis for active deliveries

Features:

- Efficient battery usage
- Geofencing alerts
- Offline support
- Distance calculation

```
const locationTracker = {
  start: async () => {
    return await Location.startLocationUpdatesAsync(TASK_NAME, {
      accuracy: Location.Accuracy.BestForNavigation,
      timeInterval: 10000
    });
  }
};
```

8. How would you implement a React Native app state management system for complex enterprise applications?

Architecture:

- **Core:** Redux + Redux Toolkit
- **Side Effects:** Redux Saga
- **Persistence:** Redux Persist
- **Performance:** Reselect for memoization

Features:

- Module federation
- State normalization
- Action logging
- Time-travel debugging

```
const store = configureStore({
  reducer: rootReducer,
  middleware: [...getDefaultMiddleware(), sagaMiddleware],
  devTools: process.env.NODE_ENV !== 'production'
});
```

9. Design a scalable file upload system for a React Native app

System Components:

- **Storage:** S3 with presigned URLs
- **Upload:** Chunked multipart uploads
- **Processing:** Lambda for post-processing
- **CDN:** CloudFront distribution

Features:

- Resume broken uploads
- Progress tracking
- Compression
- Virus scanning

```
const upload = async (file) => {
  const chunks = await splitIntoChunks(file, 5 * 1024 * 1024);
  const uploads = chunks.map(chunk => uploadChunk(chunk));
  return Promise.all(uploads);
};
```

10. How would you implement a React Native app analytics and monitoring system?

Components:

- **Analytics:** Firebase Analytics + Custom events
- **Error Tracking:** Sentry/Crashlytics
- **Performance:** Custom metrics collection
- **Monitoring:** Real-time dashboards

Features:

- Custom event tracking
- Performance monitoring
- Crash reporting
- User flow analysis

```
const analytics = {
  trackEvent: (name, params) => {
    Analytics.logEvent(name, params);
    CustomAnalytics.track(name, params);
  }
};
```

Coding and Debugging

This section presents practical coding challenges and questions about debugging techniques.

1. Write a function to flatten a deeply nested array in React Native while maintaining order.

Solution:

Here's an efficient recursive approach:

```
const flattenArray = (arr) => {
  return arr.reduce((flat, item) =>
    flat.concat(Array.isArray(item) ? flattenArray(item) : item),
  []);
}
```

Example usage:

```
const nested = [1, [2, 3, [4, 5]], 6];
console.log(flattenArray(nested)); // [1, 2, 3, 4, 5, 6]
```

2. Implement a custom debounce function for handling frequent events in React Native.

Implementation:

```
const debounce = (func, delay) => {
  let timeoutId;
  return (...args) => {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func.apply(null, args), delay);
  };
}
```

Usage example:

```
const debouncedSearch = debounce((text) => {
  // API call or heavy computation
}, 300);
```

3. Create a memory leak detector for React Native components.

Solution:

```
class MemoryLeakDetector {
  constructor(componentName) {
    this.componentName = componentName;
    this.timeoutId = setTimeout(() => {
      console.warn(`Possible memory leak in ${componentName}`);
    }, 5000);
  }
  componentWillUnmount() {
    clearTimeout(this.timeoutId);
  }
}
```

Key points:

- Helps identify components that aren't properly unmounting
- Should be used in development only
- Can be extended with more sophisticated detection logic

4. Implement a custom hook for handling infinite scroll in a React Native FlatList.

Implementation:

```
const useInfiniteScroll = (fetchData, initialPage = 1) => {
  const [page, setPage] = useState(initialPage);
  const [loading, setLoading] = useState(false);
  const loadMore = async () => {
    if (!loading) {
      setLoading(true);
      await fetchData(page);
      setPage(p => p + 1);
      setLoading(false);
    }
  };
  return { loadMore, loading, page };
}
```

5. Write a function to deep clone a React Native component's props while preserving methods.

Solution:

```
const deepCloneProps = (props) => {
  if (props === null || typeof props !== 'object') return props;
  const clone = Array.isArray(props) ? [] : {};
  for (let key in props) {
    clone[key] = typeof props[key] === 'function'
      ? props[key]
      : deepCloneProps(props[key]);
  }
  return clone;
}
```

6. Implement a custom error boundary for React Native with error reporting.

Implementation:

```
class CustomErrorBoundary extends React.Component {
  state = { hasError: false, error: null };
  static getDerivedStateFromError(error) {
    return { hasError: true, error };
  }
  componentDidCatch(error, errorInfo) {
    // Send to error reporting service
    reportError(error, errorInfo);
  }
}
```

7. Create a function to detect circular references in React Native props.

Solution:

```
const detectCircular = (obj, seen = new WeakSet()) => {
  if (typeof obj !== 'object' || obj === null) return false;
  if (seen.has(obj)) return true;
  seen.add(obj);
  return Object.values(obj).some(val => detectCircular(val, seen));
}
```

Usage:

```
const props = { a: {} };
props.a.b = props;
console.log(detectCircular(props)); // true
```

8. Implement a performance monitoring HOC for React Native components.

Implementation:

```
const withPerformanceMonitoring = (WrappedComponent) => {
  return class extends React.Component {
    componentDidMount() {
      this.renderStart = performance.now();
    }
    componentDidUpdate() {
      console.log(`Render time: ${performance.now() - this.renderStart}ms`);
    }
    render() {
      return ;
    }
  };
}
```

9. Write a custom hook for handling keyboard events with cleanup in React Native.

Solution:

```
const useKeyboard = () => {
  const [keyboardHeight, setKeyboardHeight] = useState(0);
  useEffect(() => {
    const showSubscription = Keyboard.addListener('keyboardDidShow',
      (e) => setKeyboardHeight(e.endCoordinates.height));
    const hideSubscription = Keyboard.addListener('keyboardDidHide',
      () => setKeyboardHeight(0));
    return () => {
      showSubscription.remove();
      hideSubscription.remove();
    };
  }, []);
```

10. Implement a custom navigation gesture handler for React Native screens.

Implementation:

```
const useNavigationGesture = () => {
  const pan = useRef(new Animated.ValueXY()).current;
  const panResponder = PanResponder.create({
    onMoveShouldSetPanResponder: () => true,
    onPanResponderMove: Animated.event([null, {
      dx: pan.x,
      dy: pan.y
    }]),
    onPanResponderRelease: handleRelease
  });
```