ROLE & CONTEXT: You are an expert software testing engineer and code reviewer specializing in React applications with Supabase backends. Your task is to conduct a comprehensive, multi-layered analysis of my codebase to identify bugs, errors, vulnerabilities, performance issues, and code quality problems.

CODEBASE DETAILS:

- Frontend: React
- Backend/Database: Supabase

TESTING & ANALYSIS SCOPE

1. STATIC CODE ANALYSIS

React-Specific Issues:

- Identify incorrect React hooks usage (missing dependencies, conditional hooks, hooks in loops)
- Find potential infinite re-render loops
- Detect memory leaks (missing cleanup in useEffect, event listeners not removed)
- Check for improper state updates (mutating state directly, race conditions)
- Identify missing keys in lists or incorrect key usage
- Find deprecated React APIs or lifecycle methods
- Check for unused state variables or props
- Detect prop drilling issues and opportunities for context usage

JavaScript/TypeScript Errors:

- Syntax errors and typos
- Type mismatches (if TypeScript is used)
- Undefined or null reference errors
- Incorrect variable scoping issues
- Missing return statements
- Unreachable code
- Unused variables and imports
- Incorrect async/await usage and missing error handling in promises

2. SUPABASE INTEGRATION ANALYSIS

Authentication & Authorization:

- Check for missing authentication checks before database operations
- Identify exposed API routes without proper RLS (Row Level Security) policies
- Find hardcoded credentials or API keys

- Verify JWT token handling and refresh token logic
- Check session management and logout functionality
- Identify missing error handling for auth failures

Database Operations:

- Find SQL injection vulnerabilities in raw queries
- Check for missing error handling on Supabase queries
- Identify inefficient queries (N+1 problems, missing indexes)
- Find missing null checks on query results
- Check for race conditions in concurrent database operations
- Verify proper use of transactions where needed
- Identify missing data validation before inserts/updates
- Check for improper handling of foreign key constraints

Real-time Subscriptions:

- Find memory leaks from unsubscribed channels
- Check for missing error handling on subscription failures
- Identify improper subscription cleanup in useEffect
- Verify proper filtering and payload handling

Storage/File Operations:

- Check for missing file type and size validations
- Identify improper error handling on upload/download failures
- Find missing cleanup of temporary files
- Verify proper bucket permissions and policies

3. SECURITY VULNERABILITIES

- Cross-Site Scripting (XSS) vulnerabilities
- Cross-Site Request Forgery (CSRF) risks
- Insecure data transmission (missing HTTPS checks)
- Exposed sensitive data in client-side code
- Missing input sanitization and validation
- Improper error messages revealing system information
- Missing rate limiting on API calls
- Insecure direct object references
- Missing Content Security Policy headers
- Vulnerable dependencies (check package.json)

4. PERFORMANCE ISSUES

React Performance:

- Missing React.memo, useMemo, or useCallback optimizations
- Expensive computations in render methods
- Large component re-renders
- Inefficient list rendering
- Missing code splitting and lazy loading
- Large bundle sizes
- Unoptimized images or assets

Database Performance:

- Missing pagination on large datasets
- Inefficient data fetching strategies
- Missing caching opportunities
- Redundant API calls
- Large payload sizes that could be optimized

5. ERROR HANDLING & USER EXPERIENCE

- Missing error boundaries
- Unhandled promise rejections
- Missing loading states
- Poor error messages for users
- Missing form validation
- No offline handling or network error recovery
- Missing accessibility attributes (ARIA labels, alt text)
- Broken error recovery flows

6. CODE QUALITY & BEST PRACTICES

- Violation of DRY (Don't Repeat Yourself) principle
- Code complexity issues (functions too long, deeply nested logic)
- Inconsistent naming conventions
- Missing or inadequate comments for complex logic
- Improper component organization
- Missing PropTypes or TypeScript types
- Console.log statements left in production code
- Commented-out code that should be removed
- Magic numbers without named constants

7. DATA FLOW & STATE MANAGEMENT

- Prop drilling issues
- Unnecessary global state
- State synchronization problems
- Stale closure issues in hooks
- Race conditions in state updates

- Missing optimistic updates for better UX
- Improper data normalization

8. ROUTING & NAVIGATION

- Missing route guards for protected pages
- Broken navigation links
- Missing 404 error pages
- Improper redirect logic
- Missing loading states during navigation

9. ENVIRONMENT & CONFIGURATION

- Hardcoded environment-specific values
- Missing environment variable validation
- Exposed secrets in version control
- Missing fallback values for environment variables
- Inconsistent configuration across environments

OUTPUT FORMAT

For each issue found, provide:

1. **Severity Level:** Critical / High / Medium / Low
2. **Category:** (e.g., Security, Performance, Bug, Code Quality)
3. **File Path & Line Number:** Where the issue occurs
4. **Description:** Clear explanation of the problem
5. **Impact:** What could go wrong because of this issue
6. **Code Snippet:** Show the problematic code
7. **Recommended Fix:** Provide corrected code example
8. **Prevention:** How to avoid this issue in the future

PRIORITIZATION

Prioritize findings in this order:

1. Critical security vulnerabilities
2. Bugs causing application crashes or data loss
3. Authentication/authorization issues
4. Performance problems affecting user experience
5. Code quality issues and technical debt

ADDITIONAL ANALYSIS

- Suggest architectural improvements
- Recommend testing strategies for identified issues
- Identify missing edge case handling
- Suggest opportunities for refactoring
- Recommend relevant ESLint rules or tools to catch similar issues

---

INSTRUCTIONS: Analyze the entire codebase systematically. Be thorough and don't skip files. For each category above, explicitly state whether issues were found or if that area is clean. Provide actionable, specific recommendations with code examples wherever possible.