**What Worked?**

* **Basic test structure** was solid - proper imports, mocking setup, and Jest configuration
* **Unit tests for business logic** (vote recording) worked excellently with comprehensive edge cases
* **Simple component tests** passed when aligned with actual component behavior
* **Mock setup patterns** were correct and followed testing best practices
* **Test organization** with describe blocks and clear test names was well-structured

**What Didn't?**

* **Assumptions about UI behavior** - AI generated tests expecting error messages and loading states that didn't exist in the component
* **Complex user interactions** - Tests assumed features like remove buttons with "×" text when the component used SVG icons
* **Environment setup** - Required extensive configuration fixes (React imports, Supabase env vars, TextEncoder polyfills)
* **Component-test mismatch** - AI couldn't "see" the actual component implementation, leading to unrealistic expectations

**What Surprised You?**

* **How close the business logic tests were** - The vote recording tests were nearly perfect and comprehensive
* **The amount of setup required** - Simple React component tests needed significant Jest configuration
* **AI's assumption of standard patterns** - Expected common UI patterns (error messages, loading states) that weren't implemented
* **The need for manual refinement** - Even well-structured tests required significant adjustments to match reality

**Key Takeaway:** AI excels at generating test structure and business logic scenarios but struggles with UI implementation details without seeing the actual code. Manual verification and adjustment are essential.