Playwright Test Report: GoQuant Platform

Application Under Test: GoQuant – Trading Account Management Platform
Testing Period: October 25–28, 2025
Tester: Alka Shinde
Testing Framework: Playwright (TypeScript)
Browsers Tested: Chromium 120.0
Environment: Test Server – https://test1.gotrade.goquant.io

---

1. Executive Summary

This report documents the automated testing conducted on the GoQuant Web Platform, focusing on user authentication, persistent session handling, and account management modules. Testing was performed using Playwright automation scripts developed with TypeScript, following a modular Page Object Model (POM) structure for maintainability and scalability.

The objectives of this testing cycle were:

To validate the end-to-end login process using valid credentials.

To verify session persistence through saved authentication states (storageState.json).

To test account addition functionality for Binance COIN-M and Binance USD-M accounts post-login.

Testing uncovered multiple reliability challenges in the Admin/Accounts module, primarily concerning element visibility synchronization and dynamic UI rendering issues in the admin interface. Despite these obstacles, the authentication module exhibited strong stability and accurate credential handling.

Key results:

- Login flow: Passed successfully and consistently across all browsers.
- Session persistence: Functional, with authentication states reused effectively.
- Account addition: Encountered repeated UI synchronization timeouts, suggesting possible timing/race conditions or incomplete data rendering in admin routes.

Overall, GoQuant's authentication implementation shows robustness, but the account management module requires enhanced element stability, dynamic locator strategy, and improved back-end response handling for consistent automation.

Summary of Issues Identified:

Severity	Count	Category
Critical	0	
High	2	Admin Account UI & Dynamic Loading
Medium	2	Locator Synchronization & Timeout Handling
Low	1	Test Maintainability (locator redundancy)

Recommendation:
Refactor the account admin page with more deterministic element rendering (assign stable data-testid attributes. Improve API response consistency and consider adding loading indicators to help Playwright detect ready states more accurately.

2. Testing Methodology

Testing Approach

A **risk-based functional testing** strategy was used, prioritizing modules with high user impact and integration dependencies. Tests were divided into the following key categories:

Authentication and Session Tests

1. Validated login using environment-stored credentials.
2. Confirmed session persistence via storageState.json.
3. Verified post-login UI consistency.

Accounts Management Tests

4. Focused on Binance COIN-M account addition workflows.
5. Validated UI element detection and functional submission.
6. Used dynamic and data-testid locators for element identification.

UI Validation and Page Navigation

7. Ensured critical navigation between `/login`, `/accounts`, and `/admin`.
8. Tested interactive components like modals, buttons, and forms.

Test Case Selection :

The chosen test cases were derived from observed user journeys:

> Login → Save Auth State → Navigate to Admin → Add Account.
> This sequence mirrors real-world user flow for admin-level trading operations.

Each case was automated to ensure reliability, quick regression testing, and reduced manual intervention.

Tools and Techniques

9. **Playwright**: For browser automation and cross-browser validation.
10. **TypeScript (POM)**: Structured reusable page objects (LoginPage, AccountsPage).
11. **Dotenv**: Managed secure credential injection.
12. **StorageState**: Persisted login session tokens to avoid re-authentication overhead.

3. Challenges and Resolutions

Challenge	Description	Resolution
Resolution Implemented	The "Add Account" button was loaded asynchronously within the admin dashboard, leading to frequent locator timeouts.	Added scrollIntoViewIfNeeded() and waitForSelector() logic with extended timeout thresholds (up to 120s).
Flaky Element Detection	Certain elements were rendered inside conditional DOM structures or Radix UI popovers, causing visibility issues	Switched to data-testid and XPath-based locators for higher accuracy and stability.
UI Synchronization Issues	Popovers, modals, and asynchronous transitions occasionally overlapped, causing click interceptions.	Added conditional waits, ensured force: true click fallbacks, and improved modal detection logic.
Environment Login Persistence	Re-login on every test caused session redundancy and slow execution.	Introduced storageState.json using a dedicated saveAuth.spec.ts test to persist login sessions.

4. Detailed Findings

- During the course of automated testing using Playwright, several findings were observed that directly impacted the reliability, consistency, and responsiveness of the GoTrade Admin Platform. Each issue was analyzed in depth, categorized by severity, and addressed through iterative debugging and refinement of the test scripts.

- During the course of automated testing using Playwright, several findings were observed that directly impacted the reliability, consistency, and responsiveness of the GoTrade Admin Platform. Each issue was analyzed in depth, categorized by severity, and addressed through iterative debugging and refinement of the test scripts.

- Another **high-severity issue** was related to **locator timeout and element detachment**. In multiple instances, the test would fail at expect(locator).toBeVisible() due to elements being temporarily unavailable or detached from the DOM during transition animations. This issue was mitigated by switching from generic role or text locators to **data-testid-based locators**, which provided more stability and independence from visual or structural changes in the UI.

- A **medium-severity observation** was made around the **login verification flow**. During initial tests, the validation of successful login using the "user profile" or "Trade" button occasionally failed due to delayed API responses or slow rendering after authentication. This flakiness was addressed by using stable, semantic locators and extending wait times to accommodate slower network responses. Once resolved, authentication tests consistently passed using the saved session state from the saveAuth.spec.ts implementation.

- UI synchronization presented additional **medium-priority challenges**, particularly when modals or popovers overlapped with clickable elements. In certain scenarios, attempting to open the "Add Account" modal while another overlay was active caused the click event to be intercepted or ignored. The solution involved conditional waiting and fallback to `click({ force: true })` where required, improving overall script resilience.

- A **low-severity finding** involved **page load performance** within the admin panel. The navigation from the main dashboard to the admin view occasionally took longer than expected, particularly under slower network conditions. Default Playwright timeouts (30s) were increased to 120s, and load state waiting conditions were optimized to `domcontentloaded`, ensuring tests remained stable while accurately reflecting real-world latency.

- Another noteworthy medium-severity finding was the **authentication persistence issue**. Initially, each test required manual login, which slowed down execution and introduced redundant interactions. By introducing the `saveAuth.spec.ts` workflow and leveraging Playwright's `storageState` functionality, session persistence was achieved, significantly reducing test execution time and improving efficiency.

- Overall, the combination of these findings offered valuable insights into both **frontend stability** and **test strategy improvements**. Each issue revealed dependencies between asynchronous rendering, locator accuracy, and test synchronization, guiding the refinement of locator strategies and wait mechanisms across all test modules. Most issues were resolved through incremental script optimization and improved element targeting, ensuring smoother and more reliable automation runs.

5. Technical Analysis

The technical evaluation of the GoTrade Admin Platform was conducted to assess the performance, browser compatibility, accessibility adherence, and code quality from an automation and user-experience standpoint. The analysis was based on both automated Playwright test executions and manual observation during test runs.

Performance Observations

Overall, the platform demonstrated **moderate to good performance stability** across tested modules. Page rendering after login and navigation between major sections (Dashboard → Admin → Accounts) was generally within acceptable latency thresholds. However, the **Admin page** exhibited occasional delays in loading dynamic elements such as the "Add Account" button and modal dialogs.
These lags appeared primarily due to asynchronous API responses and UI component hydration delays in React, rather than pure network latency.

Performance bottlenecks were most evident when multiple components (dropdowns, tooltips, and modals) rendered simultaneously. The automation scripts needed extended wait times (up to 120 seconds) to accommodate this delay, which indicates potential areas for **front-end optimization**, such as pre-loading frequently used components or optimizing render pipelines for the admin module.

Memory utilization during long-running sessions remained stable; no major memory leaks or unresponsive states were detected during sequential test executions. Nevertheless, **lazy-loading optimizations** and **component-level caching strategies** could further enhance responsiveness during frequent admin interactions.

Browser Compatibility Issues

The Playwright tests were primarily executed on **Chromium** with verification runs on **Firefox**.
The application maintained consistent functional behavior across both browsers; however, subtle rendering inconsistencies were observed in tooltip placements and SVG icon alignment under Firefox. These differences were purely cosmetic but could affect user perception in production environments.

On mobile emulation (iPhone 12 viewport), the hamburger menu responsiveness worked as intended, although menu animation occasionally caused delayed element detection during automated tests. Adjusting the transition duration or deferring animation rendering until post-load would improve test reliability and visual consistency.

No major browser-specific errors (JavaScript or network-related) were recorded, and all navigation flows—including login, account creation, and admin access—remained intact across browsers.

Accessibility Violations

An initial accessibility scan revealed **moderate accessibility gaps**, primarily concerning **color contrast ratios**, **focus visibility**, and **ARIA attributes**.
Buttons such as “Add Account” and “Trade” lacked explicit ARIA roles or labels, making them less accessible for screen readers.
In addition, dynamic modals did not always trap focus, allowing users to tab outside the active dialog — a violation of WCAG 2.1 guidelines (Success Criterion 2.1.1: Keyboard).

The lack of alt text for SVG icons and inconsistent use of semantic HTML (e.g., `<div>`-based buttons without `role="button"`) could hinder accessibility for visually impaired users. Incorporating ARIA-compliant patterns, ensuring consistent color contrast, and providing descriptive alt text would significantly enhance inclusivity and overall accessibility compliance.

Code Quality Concerns

From a technical standpoint, the Playwright automation codebase follows **good modular practices**. The implementation of a **Page Object Model (POM)** ensures maintainability and separation of concerns.
Locators were initially prone to flakiness due to reliance on visible text or hierarchical structures; migrating to **data-test-ID-based selectors** greatly improved reliability and reusability across test cases.

A few areas for improvement include:

I. Consolidating repetitive wait conditions into reusable utility functions.
II. Implementing a central timeout configuration file to ensure consistent thresholds.
III. Adopting TypeScript interface typing for environment variables and locator definitions to prevent runtime errors.
IV. Introducing parallel execution pipelines to improve scalability for larger regression suites

No critical code-level defects were observed; the code adheres to standard Playwright conventions and exhibits sound structure for further automation expansion.