Test Report: GoTrade Web Platform

Application Under Test: GoTrade – Online Trading and Account Management Platform
Test Period: October 29, 2025
Tester: Akul Duggal
Testing Framework: Playwright with TypeScript
Browsers Tested: Chromium , Firefox , WebKit

Executive Summary

This project focuses on the testing and automation of the **GoTrade web platform**, accessible at https://test1.gotrade.goquant.io/gotrade. The application functions as an **online trading interface**, enabling users to monitor and manage trading-related activities through a browser-based environment.

The testing initiative began with comprehensive **manual validation** of the platform's user interface and core functionality. The primary objective was to ensure that all UI components were user-friendly, visually consistent, and functionally accessible across different sections of the application. This phase provided a deeper understanding of the system's usability, layout behavior, and areas that could negatively impact the user experience.

Following manual validation, **UI automation** was developed using **Playwright with TypeScript**. The automated tests focused on verifying the functionality of key UI elements, including button interactions, form behaviors, and navigation flow, to confirm that core user actions were performing as intended. In total, **13 automated test cases** were implemented to cover essential workflows and validate the system's basic interactive stability.

While API-level and regression testing were not within the current scope, this phase effectively establishes a **foundation for future automation**, setting the groundwork for broader testing coverage such as functional, integration, and performance-level validation in upcoming cycles.

Testing Methodology

Testing Approach

A **hybrid exploratory and automated testing approach** was adopted to assess the GoTrade web platform. The process began with exploratory testing to understand the application's structure, available modules, and user interactions. This hands-on exploration helped uncover usability gaps, missing validations, and visual inconsistencies from a real user's perspective.

Once the core behavior and navigation flow were well understood, **automation testing** was introduced using **Playwright with TypeScript**, structured under the **Page Object Model (POM)** framework. This allowed efficient replication of key UI actions, navigation paths, and validation scenarios across multiple browsers.

Due to functional restrictions within certain modules, the focus remained primarily on **UI-level validation** , ensuring that all visible components were functional, responsive, and correctly aligned. The combined manual and automated testing approach provided both depth and coverage, enabling comprehensive evaluation of GoTrade's visual stability, navigation, and overall usability.

Test Case Selection Rationale

From the complete user journey, I selected **test cases** that covered the most repetitive and essential actions performed by users. Automating these flows ensured that critical operations could be verified quickly and consistently, saving manual effort and improving regression confidence.

Test case selection was guided by the need to **automate repetitive workflows** (such as navigation and basic form interactions) and validate **core UI functionality**. Some features, particularly **trading-related and order placement functions**, were intentionally excluded from the automation scope due to restricted access or incomplete implementation in the test environment. The automation scripts were designed directly based on the **manual test cases** I had earlier created during exploratory testing, maintaining consistency between both phases.

Tools and Techniques Employed

The automation framework was built using **Playwright with TypeScript**, chosen for its modern syntax, fast execution, and built-in cross-browser capabilities. Tests were executed sequentially through the Playwright Test Runner, leveraging direct **locator strategies** such as text selectors and CSS identifiers for identifying UI elements.

A **Page Object Model (POM)** structure was implemented to enhance maintainability and scalability, ensuring better separation of page-specific actions from test logic. Each module's interactions and locators were encapsulated in dedicated page files, improving readability and reusability across test cases.

The scripts focused on ensuring that key elements like buttons, forms, and navigational links behaved as expected during interaction. No external libraries or data generators were used in this phase, as the emphasis was primarily on validating **UI flow and interaction consistency** rather than data-driven or API-level testing.

Challenges Encountered and Solutions Implemented

One of the primary challenges encountered during this project was the **initial learning curve associated with both TypeScript and Playwright**, as I had no prior experience with these technologies. Transitioning from manual testing to automation required a deep understanding of asynchronous handling, selectors, and the Page Object Model in a TypeScript-based framework. To overcome this, I dedicated time to studying Playwright's architecture, test structure, and best practices through documentation and experimentation before implementing the scripts.

To strengthen my understanding, I also **used Selenium as a reference point** to compare its automation flow and command structure with Playwright's. This comparative approach helped me better grasp how Playwright's asynchronous model and auto-waiting mechanisms differ from Selenium's more traditional, synchronous execution style. By observing these differences, I was able to adopt more efficient locator strategies, minimize flakiness, and design cleaner, more maintainable scripts in Playwright.

Another major limitation was that **order placement and several trading-related functionalities were restricted or non-operational** in the test environment. These limitations prevented in-depth testing of transactional processes and order management workflows. To address this, I redirected my efforts toward **accessible modules** such as navigation, login, and general UI validation. By focusing on the available functionality, I ensured that all accessible elements and workflows were thoroughly tested and automated, allowing for meaningful coverage and stable reporting despite environmental constraints.

Detailed Finding

Critical Issues

CRIT-001: Placed Orders Not Reflected in Working Orders
Severity: Critical
Description: When users place an order using any available algorithm, the system confirms with a success message ("Order accepted") but fails to display the order in the Working Orders table.
Steps to Reproduce:

1. Go to the GoTrade page.
2. Place an order using any available algorithm.

3. Open the Order History or Working Orders table.
4. Observe that the placed order is not listed.

Expected Behavior: The placed order should immediately appear in the Working Orders table once confirmed.
Actual Behavior: The system displays a success message, but the order is missing from the Working Orders table.

CRIT-002: Unauthorized Access to Other User Data
Severity: Critical
Description: Users can view reconciliation and order information belonging to other user accounts.
Steps to Reproduce:

1. Log in using valid user credentials.
2. Navigate to GoOps → Reconciliation Details.
3. Observe the "Account" field and note that entries for other users are visible.

Expected Behavior: Each user should only be able to view their own account data.
Actual Behavior: The user can see reconciliation data for other accounts, exposing sensitive information and violating access control.

CRIT-003: Login Failure with Valid Credentials
Severity: Critical
Description: Login attempts using valid credentials fail, preventing users from accessing the platform.
Steps to Reproduce:

1. Open the GoTrade application or web page.
2. Enter valid username/email and password.
3. Click Login / Sign In.
4. Observe that the login fails despite valid credentials.

Expected Behavior: The user should be authenticated successfully and redirected to the GoTrade dashboard.
Actual Behavior: Login fails, and the user remains on the login page despite using valid credentials.

High-Priority Issues

HIGH-001: GoMarket Page Freezes on Access
Severity: High
Description: The GoMarket page becomes unresponsive immediately upon loading, preventing

user interaction or navigation.
Steps to Reproduce:

1. Navigate to the GoMarket page.
2. Wait for the page to load.
3. Attempt to interact or exit the page.

Expected Behavior: The GoMarket page should load smoothly, allowing users to view and interact with all sections.
Actual Behavior: The page freezes immediately after loading, and no further actions can be performed.

HIGH-002: File Generation in GoOps Produces No Downloadable Output
Severity: High
Description: When attempting to generate a report file, a confirmation message appears but no downloadable file is provided.

Steps to Reproduce:

1. Navigate to the GoOps page.
2. Click on the Generate button.
3. Observe that a confirmation message appears, but no CSV file is downloaded.

Expected Behavior: A CSV file should be generated and downloaded successfully after confirmation.
Actual Behavior: The system shows a confirmation message, but no file is generated or downloaded.

HIGH-003: Liquidate Position Action Returns No Response
Severity: High
Description: Clicking the Liquidate Position button produces no response, displaying a UDP server error instead.
Steps to Reproduce:

1. Navigate to the GoTrade page.
2. Select any active position under Open Positions.
3. Click Liquidate Position.
4. Observe the system response.

Expected Behavior: The selected position should be liquidated, and a success or confirmation message should be displayed.
Actual Behavior: No action occurs, and a UDP server error appears on screen.

HIGH-004: Added Accounts Not Visible on Admin Page
Severity: High
Description: Newly added accounts fail to appear in the Admin page's accounts list, even after successful addition.

Steps to Reproduce:

1. Log in with valid admin credentials.
2. Navigate to the Admin page.
3. Add one or more accounts using valid API credentials.
4. Check the Accounts List section.

Expected Behavior: All added accounts should appear immediately in the Accounts List.
Actual Behavior: Added accounts do not appear, giving the impression that the operation failed.

Low-Priority Issues

LOW-001: Broken Documentation Link in Algorithm Info
Severity: Low
Description: The documentation link provided under certain algorithms leads to a non-existent page.
Steps to Reproduce:

1. Navigate to the GoTrade page.
2. Select the Ratio Trade algorithm.
3. Hover over the **(i)** information icon and click the documentation link.

Expected Behavior: The link should redirect the user to the correct documentation page for the selected algorithm.
Actual Behavior: A "Page Not Found" or "The page does not exist" error message is displayed instead of the documentation page.

LOW-002: Missing Documentation Links for Specific Algorithms
Severity: Low
Description: The information pop-up for certain trading algorithms does not include the expected documentation link.
Steps to Reproduce:

1. Navigate to the GoTrade page.
2. Select any of the following algorithms: Market, Limit, TWAP, or VWAP.
3. Hover over the **(i)** icon in the algorithm's information box.

Expected Behavior: Each algorithm's info pop-up should contain a documentation link for user reference.
Actual Behavior: The documentation link is missing from the pop-up, leaving users without a direct reference to the algorithm's details.

LOW-003: Font Size Changes After Toggling Discovery Mode
Severity: Low
Description: The font size of the coin name changes unexpectedly after toggling the Discovery Mode option.
Steps to Reproduce:

1. Navigate to the Order Book section on the GoTrade page.
2. Toggle the Discovery Mode option on and off.
3. Observe the font size of the coin name.

Expected Behavior: Toggling Discovery Mode should only change functional behavior without affecting the UI design.
Actual Behavior: The font size of the coin name reduces significantly, affecting visual consistency.

Technical Analysis

Performance Observations

The application exhibited **significant performance degradation** as user load increased. When multiple users accessed the platform simultaneously, **critical operations such as login and order placement frequently failed**, disrupting the overall workflow. Several instances of **"UDP server error"** and repeated **authentication failures** were observed, while certain pages remained unresponsive and failed to display any data even after extended waiting periods.

Although minor waits were added during automation to account for slow rendering, the root cause appeared to be **backend inefficiencies and server-side instability** rather than frontend synchronization delays. Overall, the system displayed **sluggish responsiveness**, recurring blockers, and noticeable slowdowns during concurrent sessions   all of which negatively impacted usability, reliability, and the overall test experience.

Browser Compatibility Issues

The majority of testing and automation were performed on the **Chromium browser**, where most test cases executed successfully and produced consistent results. Cross-browser testing on **Firefox** and **WebKit** also completed without major functional failures, indicating a reasonable level of compatibility across environments. However, noticeable differences were observed in **UI responsiveness and rendering speed** between browsers.

In certain cases, the site would **take significantly longer to respond or render elements** on Firefox or WebKit compared to Chromium. This inconsistency introduced challenges during automation, as **using predefined wait times often led to premature interactions or test failures**. Interactive elements such as buttons and dropdowns sometimes required multiple attempts to register clicks or selections, highlighting **timing and rendering disparities** that can directly impact both test reliability and real user experience across different browsers.

Accessibility Violations

A major **accessibility and data security issue** was discovered during testing   users were able to **view orders belonging to other accounts**. This represents a significant **privacy and access-control vulnerability**, as sensitive user data should be strictly confined to the authenticated session. In addition to this, several **minor accessibility concerns** were noted, including inconsistent visibility of certain UI elements and unclear error or feedback messages during failed operations. These issues could lead to **user confusion** and disrupt seamless navigation across the platform.

Code Quality and Functional Concerns

While no direct code-level issues were analyzed, multiple **functional bugs** were identified during testing. The most frequent problems included the **inability to place orders**, **viewing of other users' orders**, **pages failing to load data**, and **UI functionality not responding as expected**. These findings suggest that the current build requires further stabilization and backend optimization before it can support real-world trading scenarios or concurrent user sessions effectively.

Test Execution Summary

Automated testing was conducted using **Playwright with TypeScript**, with each test executed across **Chromium, Firefox, and WebKit** browsers to ensure consistent performance and cross-browser reliability. The test suite primarily focused on validating **UI interactions, navigation flow, and feature stability** within the GoTrade platform.

Approximately **80% of the executed tests passed successfully**, demonstrating that the core user interface and navigational elements perform reliably under stable site conditions. The remaining **20% of tests failed**, primarily due to issues such as **login instability, delayed page loads**, and **restricted functionality** within certain modules like order placement and manipulation.

The **login module** exhibited intermittent authentication failures, particularly under slow response conditions. Additionally, the **GoRisk** and **Post Trade Analysis** sections showed inconsistencies in data rendering and updates. In contrast, the **GoTrade** and **GoOps** modules maintained consistent stability and responsiveness across all browsers.

Overall, the GoTrade application can be considered **moderately stable** , with critical UI and navigation features functioning as intended , while **backend responsiveness, data synchronization, and authentication reliability** remain key areas for improvement. Addressing these issues will be essential to enhance the platform's overall performance and production readiness.
