Ferovinum QA Strategy and Implementation Plan

Executive Summary

This document outlines the quality assurance strategy for the Ferovinum trading platform, focusing on automating tests for wine and whisky inventory transactions. The strategy is designed to ensure comprehensive test coverage across UI components, API endpoints, and end-to-end workflows using Playwright.

1. Testing Objectives

The primary objectives of our QA strategy are to:

- Verify the core functionalities of buying and selling stock work correctly
- Ensure data consistency across the UI and API
- Validate form inputs and error handling
- Measure and monitor performance metrics
- Support continuous integration and deployment

2. Testing Scope

In Scope

- User interface testing of the trading platform
- API endpoint verification (GET and POST /stocks)
- End-to-end workflow testing
- Performance and load testing
- Cross-browser compatibility

Out of Scope

- Security testing
- Penetration testing
- Mobile responsiveness testing
- Localisation testing (TBC discussions needed with the business) easily achievable tho through browser stack)

3. Testing Approach

We will implement a layered testing approach:

1. **Unit Tests**: Testing individual components in isolation
2. **Integration Tests**: Testing interactions between components
3. **API Tests**: Verifying API endpoints functionality
4. **UI Tests**: Testing the user interface components
5. **E2E Tests**: Testing complete business flows
6. **Performance Tests**: Measuring response times and system behavior under load

4. Test Environment

- **Development**: Local environment for test development
- **CI/CD**: GitHub Actions pipeline for continuous testing
- **Pre-Production**: Environment that mirrors production for final validation

5. Test Automation Framework

Technologies

- **Testing Framework**: Playwright
- **Programming Language**: TypeScript
- **Reporting**: HTML Reports, CI integration, slack push notifications in dedicated e2e channel
- **Design Pattern**: Page Object Model

Framework Structure - Please enlarge font for easy readability if needed

```
fero-e2e/
├── .github/                        # GitHub Actions CI/CD
configuration
│   └── workflows/
│       └── playwright.yml          # Playwright test workflow
│
├── playwright-report/              # Generated test reports
│   └── index.html                  # HTML report viewer
│
├── test-results/                   # Test execution results
│
├── tests/                          # Test files directory
│   ├── api/                        # API test files
│   │   └── stocks.spec.ts          # Stock API endpoint tests
│   │
│   ├── ui/                         # UI test files
│   │   ├── visual.spec.ts          # Visual/UI verification tests
│   │   ├── buy.spec.ts             # Buy functionality tests
│   │   └── sell.spec.ts            # Sell functionality tests
│   │
│   ├── e2e/                        # End-to-end test files
│   │   └── trading.spec.ts         # Complete trading scenarios
```

```
│   │
│   ├── pages/                  # Page Object Models
│   │   └── trading-page.ts     # Main trading page object
│   │
│   ├── utils/                  # Helper utilities
│   │   ├── api-helper.ts       # API testing utilities
│   │   └── test-helper.ts      # General test helpers
│   │
│   └── fixtures/               # Test data fixtures
│
├── node_modules/               # Project dependencies
│
├── playwright.config.ts        # Playwright configuration
├── package.json                # Project dependencies and
scripts
├── package-lock.json           # Lock file for dependencies
├── .gitignore                  # Git ignore rules
└── README.md                   # Project documentation
```

6. Test Types and Coverage

UI/Visual Tests

- Element visibility and properties
- Layout and appearance
- Table data display
- Form interactions

Functional Tests

- Buy stock with various parameters
- Sell stock with various parameters
- Form validation and error handling
- Table updates after operations

API Tests

- GET endpoint functionality
- POST endpoint functionality
- Request/response validation
- Error handling

Performance Tests

- Page load time
- Operation response time
- Behavior under concurrent load

7. Test Data Management

- Create unique test data for each test run
- Clean up test data after test execution when possible
- Use API to set up test data for UI tests
- Implement data generation utilities

8. Execution Strategy

Automation Execution

- Run smoke tests on each pull request
- Run full regression suite nightly
- Execute performance tests weekly
- Manual exploratory testing for new features

Environments

- Run tests across multiple browsers (Chrome, Firefox, Safari)
- Execute in headless mode for CI/CD
- Run in headed mode for debugging and development

9. Test Prioritisation via Traffic light system approach

Tests will be prioritised using the following criteria:

1. **Critical**: Core business functionality (buying/selling)
2. **High**: Data consistency and validation
3. **Medium**: UI/UX elements
4. **Low**: Edge cases and rare scenarios

10. Defect Management

- Log defects with detailed reproduction steps
- Include screenshots and trace files
- Categorize by severity and priority
- Link to failing test cases

Given the opportunity to deliver such an automation suite for the actual Ferovinum trading platform. Please see the below implementation phases:

11. Implementation Plan

Phase 1: Setup and Basic Tests - R&D

- Set up test framework and environment
- Implement page object models
- Create basic UI tests
- Verify connectivity to application

Phase 2: CI/CD Integration - Deliver benefit to the business from Day 1 for Developers to run locally and through CI/CD triggering automation suite upon commit, PR and environment to environment promotion.

- Set up GitHub Actions workflow
- Configure test reporting
- Implement test result notification
- Document test procedures
- Run basic tests even if they FAIL we can see its delivering value to the business

Phase 3: Core Functionality Tests

- Implement buy operation tests
- Implement sell operation tests
- Add form validation tests
- Create API tests

Phase 4: Advanced Tests

- Implement end-to-end workflows
- Add data-driven tests
- Create performance tests
- Implement visual comparison tests

12. Resources and Roles especially when gathering user requirements via 3 Amigo Process

The aim will be to have myself and 2 more QE SDET engineers on my team to continuously:

1. **Gather testing requirements**
2. **Create tests using Playwright codegen record and playback tool where possible**
3. **Maintain existing tests prone to breakages**
4. **R&D AI testing initiatives - How can we make our lives easier with implementing AI such as PR review AI processes**

Roles

- **QE/SDET**: Implement and maintain tests
- **Developer**: Support test implementation and fix defects
- **Product Owner**: Define test scenarios and acceptance criteria

Resources

- Testing environment
- Application documentation
- API specifications
- Development support

13. Risk Assessment Matrix

Risk	Impact	Probability	Mitigation
Flaky Tests	High	Medium	Implement retries and explicit waits
Environment availability	High	Medium	Set up dedicated test environment
Test data corruption	Medium	Low	Create fresh test data for each run
Browser compatibility	Medium	Medium	Test across multiple browsers which playwright can do

14. Success Criteria

- 90% or higher test coverage for critical functionality
- All tests stable with less than 5% flakiness
- CI/CD pipeline integration complete
- Performance tests established with baselines

15. Reporting

- Generate HTML reports after each test run
- Send notifications for test failures
- Track test metrics and trends
- Regular review of test results

16. Continuous Improvement

- Regular review of test coverage
- Refactoring and optimization of tests
- Keeping dependencies up to date

- Knowledge sharing and documentation
- AI integration within the e2e automated testing pipeline

17. Feedback / Comments / Actions

Best Practices for Document Creation: A Comprehensive Approach

This section serves as a dedicated space for incorporating comments, feedback, and actionable items throughout the document creation process. This practice is strongly encouraged as it fosters transparency, facilitates collaboration, and ensures a well-structured and high-quality final output.

Purpose and Benefits:

- **Centralised Feedback:** This designated area provides a single point of reference for all input and suggestions related to the document's content, structure, and overall effectiveness. This eliminates the need to search through multiple versions or rely on disparate communication channels.
- **Enhanced Collaboration:** By clearly documenting feedback and actions, all stakeholders involved in the document's creation can readily understand the rationale behind changes and contribute effectively. This promotes a collaborative environment and reduces the potential for misunderstandings.
- **Improved Traceability:** Maintaining a record of comments and the corresponding actions taken ensures a clear audit trail of the document's evolution. This is particularly valuable for complex documents or projects requiring multiple revisions.
- **Actionable Insights:** Explicitly outlining action items provides clarity on the tasks that need to be completed and who is responsible for them. This promotes accountability and ensures that feedback is translated into tangible improvements.
- **Knowledge Retention:** Documenting the feedback and decision-making process captures valuable insights and lessons learned during the document creation cycle. This knowledge can be leveraged for future projects, leading to more efficient and effective document development.

Implementation Guidelines:

- **Visibility:** This section should be clearly identifiable and easily accessible within the document. Consider placing it at the beginning or end of the document, or utilizing a dedicated section within a collaborative platform.
- **Formatting:** Employ consistent formatting to distinguish comments, feedback, and action items. Using bullet points, numbered lists, or different font styles can enhance readability and organization.
- **Attribution:** Clearly indicate the source and date of each comment or feedback entry to provide context and facilitate follow-up if necessary.
- **Action Tracking:** For each action item, specify the task, the responsible party, and a target completion date. Regularly update the status of action items to ensure progress.
- **Regular Review:** Periodically review the comments and action items to ensure that all feedback has been addressed and that the document is progressing according to plan.
- **Archiving:** Once the document is finalised, consider archiving this section as a record of the document's development process.

By adhering to these best practices, we can ensure a more streamlined, collaborative, and ultimately more successful document creation process. This dedicated space for feedback and actions will serve as a valuable tool for continuous improvement and the development of high-quality deliverables.