Vlad Georgescu
Cluj-Napoca, Romania

Tokero QA Automated Testing – Project Documentation

Repository: https://github.com/GeorgescuMVlad/Tokero-QA-Automated-Testing
Environment Under Test: https://tokero.dev/en/
Tech Stack: Playwright with .NET (C#) - NUnit Test Project (.NET Core) – Target framework: .NET 8.0, Visual Studio Community 2022

1. Purpose:

The aim of this project was to design and implement automated integration tests for the Tokero platform's staging environment. The tests were developed to simulate real user interactions and ensure that key website flows function correctly across languages and browsers.

2. Testing Strategy

What was tested:

1. Policy Pages Crawl & Validation

- The test navigates to the homepage and removes UI obstructions like modals and cookie banners to ensure clean interaction.
- It identifies all relevant policy links from the footer before navigating to them, avoiding stale DOM issues.
- For each unique policy page, it verifies:
    - That the page loads successfully (HTTP status 200).
    - That the page contains meaningful content, validated through selectors like main, article, or .policy-container, or by checking the raw text length.
- This process simulates real-world user behavior when browsing legal documents and ensures content visibility and accessibility across all pages.

2. Multi-Language Switching

- The test switches between English (/en), French (/fr), and German (/de) using a custom LocalizationHelper.
- For each language variant:
    - The test ensures navigation to the correct localized homepage.
    - It then runs the full policy validation test, confirming that translated legal pages are reachable and not broken.
- This confirms internationalization coverage and ensures consistency across language variants.

3. Multi-Browser Compatibility

- The test reinitializes the Playwright browser context for each of the three engines:
  - Chromium (Google Chrome)
  - Firefox
  - WebKit (Safari)
- For each browser:
  - It runs the full policy validation flow from scratch.
- This cross-browser check ensures Tokero's frontend renders and behaves consistently, catching discrepancies between browser engines.

4. Load Performance Measurement

- The test measures load times of each policy page using Stopwatch around navigation calls.
- It asserts that pages load in under 3 seconds, flagging any performance regressions.
- This provides a lightweight baseline of frontend responsiveness.

3. Design Decisions & Trade-offs

- Pragmatic architecture

- The tests were kept lean and execution-focused to prioritize functionality within time constraints.
- A minimal test framework was used instead of deep abstraction layers or custom runners, making the code easier to follow and modify.

- Modular, but reusable

- Core logic like policy validation is reused across languages and browsers, reducing duplication.
- Helper utilities (e.g., LocalizationHelper) isolate non-core responsibilities like language switching.

- Avoided Login/Authentication

- Authentication tests were deliberately excluded due to reCAPTCHA constraints, as mentioned in the assignment requirements.

- Performance scope

- Full-scale performance or tracing analysis (e.g., lighthouse, Playwright trace viewer) wasn't implemented.
- However, basic load time measurements are included, and future iterations could expand on this with more detailed tracing or synthetic monitoring.

4. Outputs & Reports

- **Reports:** Generated reports with all tests run and passed in .trx and .md formats.
- **Screenshot:** Screenshot with all tests run and passed from Visual Studio Test Explorer.

5. Future improvements

- Add performance benchmarks (load time, time-to-interactive) for key pages.
- Add accessibility (a11y) checks using Playwright plugins.
- Mock login or use test-specific tokens to unlock authenticated flows.
- Integrate tests into CI pipelines (e.g., GitHub Actions).

6. Conclusion

This automated testing suite provides a foundational framework to validate critical functionalities of the Tokero website. By focusing on key user flows and ensuring compatibility across languages and browsers, it aims to enhance the website's reliability and user experience.