**Quality Assurance (QA) and User Acceptance Testing (UAT) Test Suite for Web Applications**

Fernando Djingga

13 June 2025

**Introduction**

I created a lightweight Quality Assurance (QA) and User Acceptance Testing (UAT) test suite for a demo web application login module. My project demonstrates how structured testing artifacts – including test plans, test cases, bug reports, and benchmark metrics – can ensure through coverage, reproducibility, and measurable efficiency compared to ad hoc exploratory testing.

My project deliverables include test artifacts (test\_cases.csv, bug\_report.md), screenshots (visual evidence of test execution and defect reproduction), and benchmark report (benchmark.md, which captures execution coverage, run time, and logged defects). My project simulates the professional QA/UAT process expected in entry-level roles, where the emphasis is not on building a large application but on demonstrating discipline, documentation, and reproducibility.

**High-Level Implementation**

My project focuses on testing a demo login module (index.html) that contains email and password inputs, client-side validation, and auxiliary features (remember-me toggle and forgot password link). Firstly, I must define the scope of the login form functionality, input validation, and basic usability. Next, I have to plan testing by determining the scope, risks, and deliverables. I must then design the 10 test cases covering positive, negative, and edge scenarios (valid login, empty fields, invalid email, weak password, toggle, double-click, etc.). After that, I execute tests by manually running each case, recoding the status of each test case. I then create reproducible bug reports with step-by-step instructions and severity levels, simulating typical QA testing procedures. Finally, I compiled the benchmark results by measuring the total cases executed, coverage percentage, run time, and logged defects.

**Coding Implementation**

The main goal of my project is to test documentation and execution results, not application logic; so the coding implementation is quite minimal.

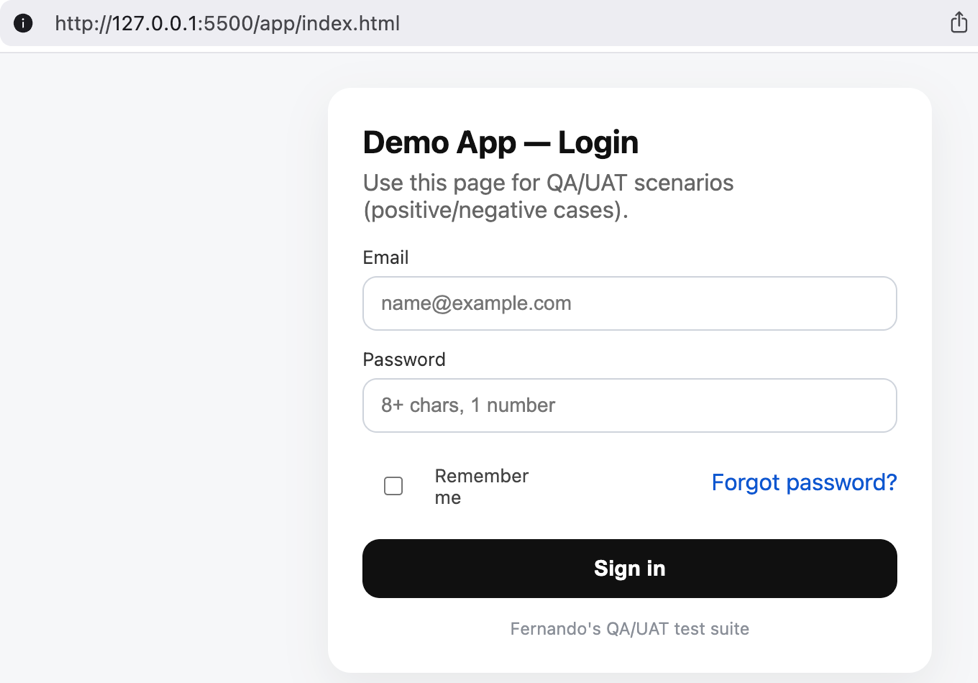
‘index.html’ and ‘script.js’ provides a basic login form with client-side validation for email and password. It checks for empty fields, invalid email formats, and weak passwords; displays success or error messages depending on input; and includes auxiliary features such as the ‘Remember me’ toggle and the ‘Forgot password’ link.

The test artifacts are implemented as static files. ‘test\_cases.csv’ defines the 10 structured test cases with ID, title, priority, preconditions, steps, expected result, and status. ‘bug\_report.md’ logs defects with reproducible steps, severity levels, and screenshot references. ‘benchmark.md’ comprises the execution coverage, run time, and defect detection efficiency.

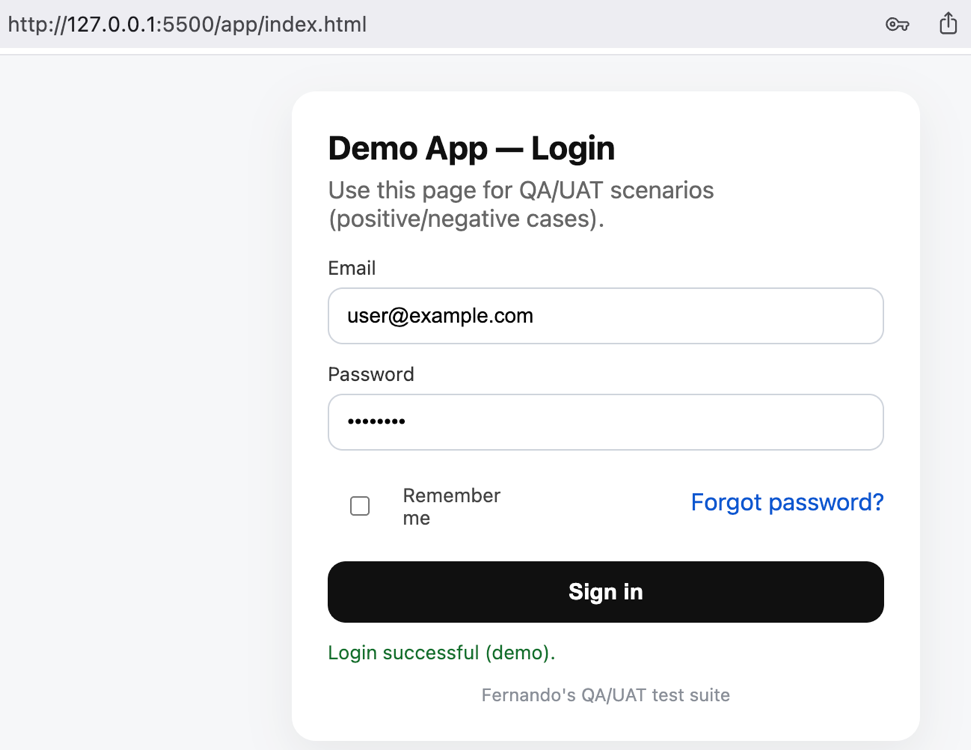
Instructions on how to run the project can be found in README.md.

**Project Results**

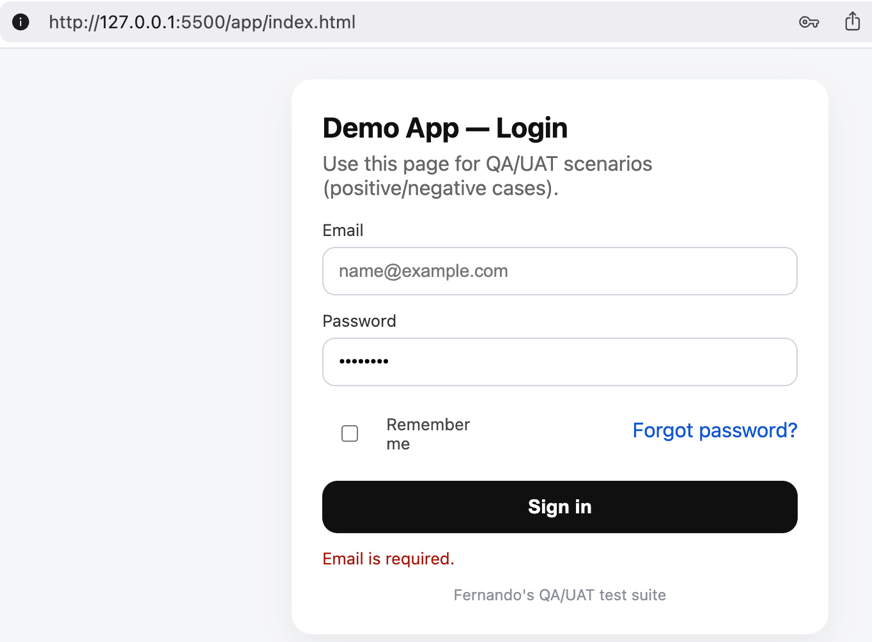
The following are the screenshots from the test cases TC-001 to TC-010. Once the live server is run, the interface can be observed.



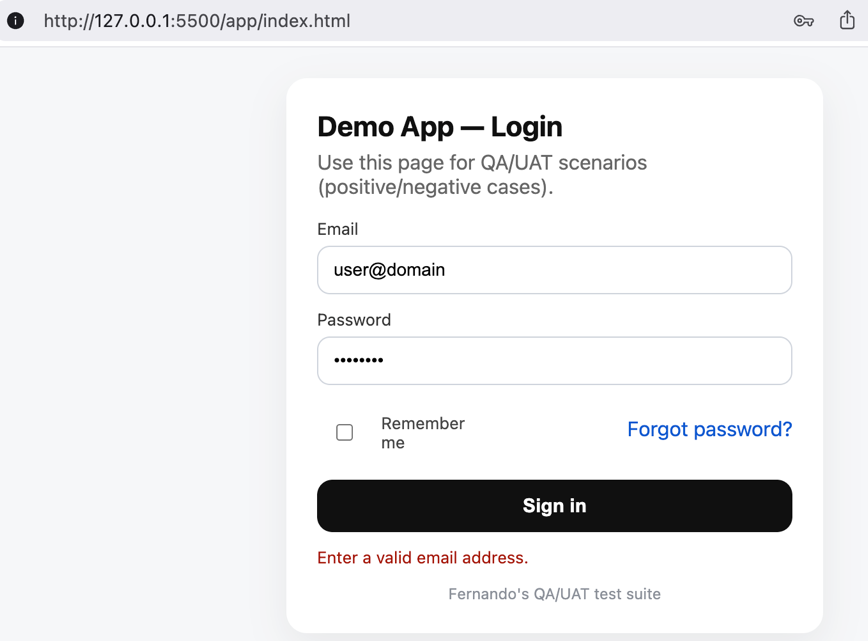
TC-001 verifies that a valid email and strong password combination results in a successful login.



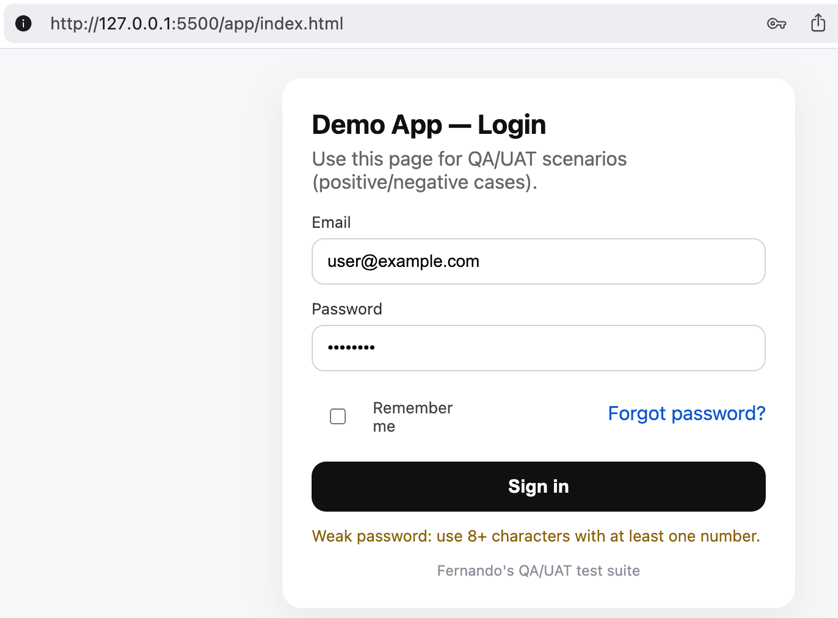
TC-002 checks that the system rejects an attempt to login when the email field is left blank



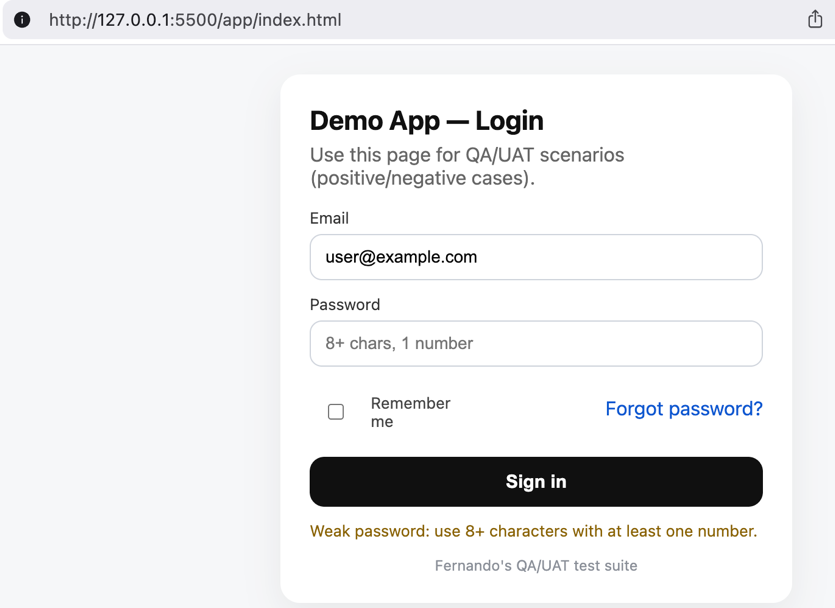
TC-003 ensures that invalid email formats, such as ‘user@domain’ without a top-level domain, are not accepted.



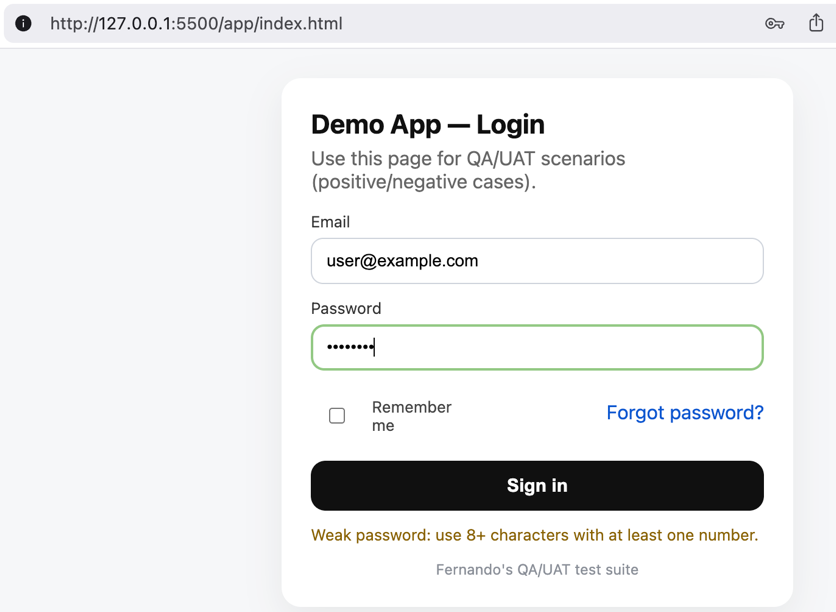
TC-004 confirms that a password with only letters triggers a weak password warning.



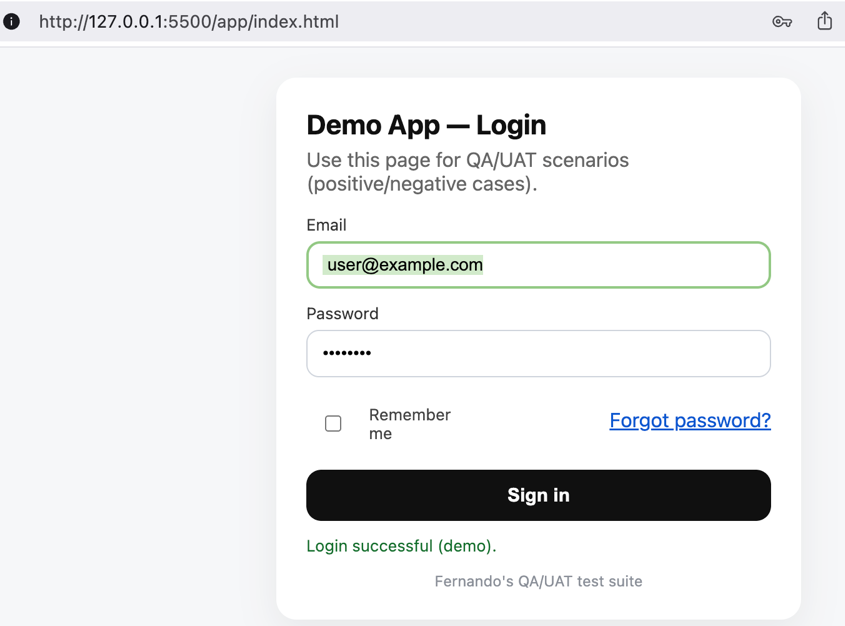
TC-005 validates that login fails if the password fields is empty.



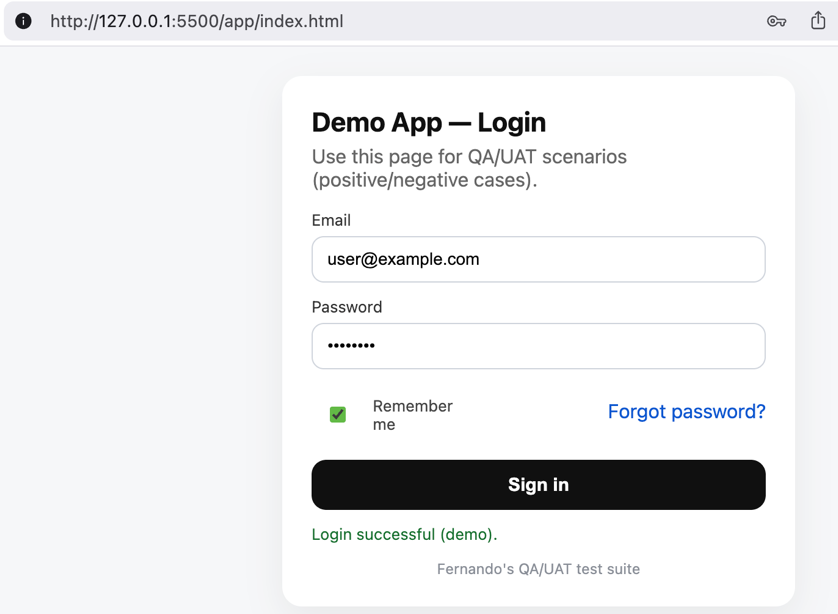
TC-006 checks that numeric-only passwords are flagged as weak.



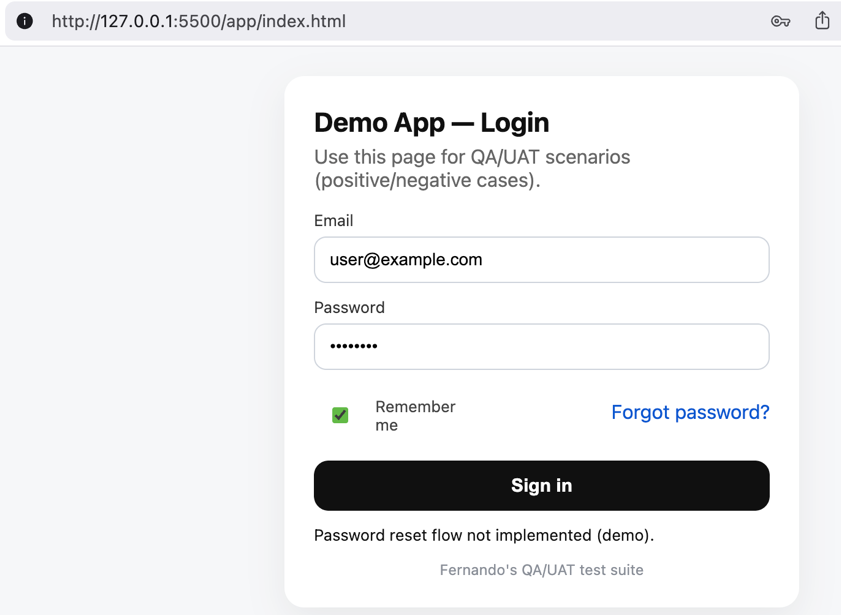
TC-007 tests whether emails entered with leading or trailing spaces are correctly handled.



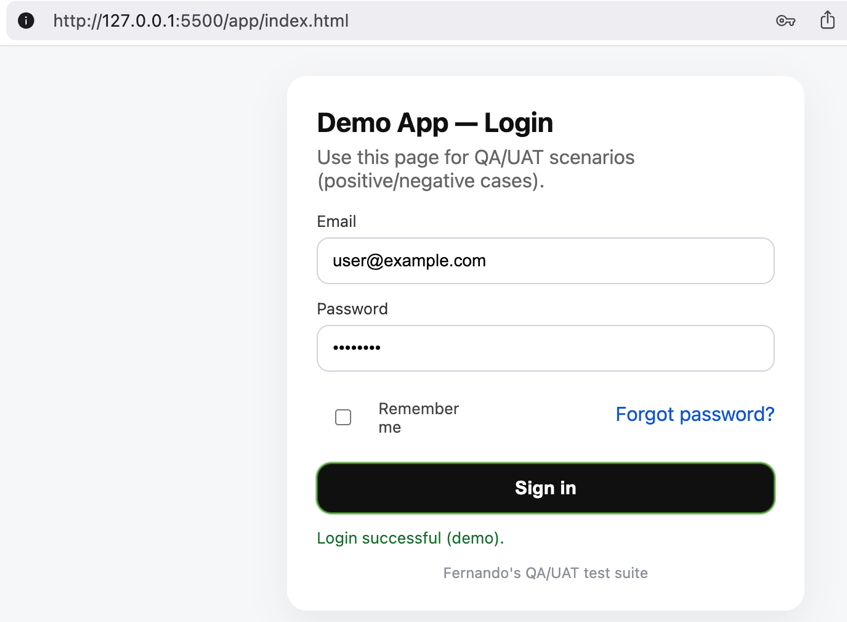
TC-008 verifies that the ‘Remember me’ toggle does not interfere with successful login.



TC-009 ensures that clicking ‘Forgot password’ shows the appropriate informational message.



TC-010 checks that rapid double-clicking the login button does not cause duplicate submissions.



The structured execution of 10 planned cases produced the following outcomes. It executed 10 out of 10 of the test cases, resulting in 100% coverage. In an earlier cycle, there are 2 reproducible bugs identified:

* BUG-001: invalid email format accepted (‘user@domain’)
* BUG-002: success message duplicated on rapid double-click

The defects logged after the bugs were fixed resulted in 0 reproducible bugs, meaning both issues were fixed (by strengthening the email validation code, and adding a submit guard) in this cycle. The execution time of this structured run took about 8 minutes from start to completion.

These results highlight the importance of structured testing. Without a test suite, exploratory testing could easily miss subtle validation and UI timing issues. With a test suite, all planned scenarios were executed consistently, earlier defects were identified and resolved, and the latest cycle validated a clean run.

**Conclusion and Project Significance**

In conclusion, my QA/UAT project demonstrates how a structured test suite ensures complete coverage where all functional and negative scenarios were tested to ensure 100% coverage; improves efficiency where it only took 8 minutes to perform; demonstrates clear defect lifecycles where two real issues were discovered in earlier runs, fixed in the code, and then resolved in another cycle which simulates the complete QA cycle of detection, documentation, and resolution. The project also confirms that the results and evidences documented in test cases, and bug reports are reproducible to enable anyone to rerun and verify the findings.

As a fresh graduate, I may not yet have extensive professional QA experience yet, but I built this project to show that I understand and can apply structured QA/UAT processes. In real-world tech environments (such as SaaS, fintech or IT services), reproducing testing and defect documentation are critical to product quality and team efficiency. My project demonstrates my ability to test functionality, improve, and deliver on metrics that are expected in a QA/UAT role.