SDS Automation QA Assessment: Technical Approach & Solution Highlights

1. Test Strategy

- **End-to-End Coverage:**
  We implemented automation to simulate real user behavior on the SDS website, covering viewing customer/user statistics and submitting demo requests.
- **Scalable, Maintainable Structure:**
  We organized the solution using the Page Object Model (POM), separating all page actions and selectors into dedicated classes. This structure makes the codebase easy to maintain and extend as the application grows.

---

2. Key Features & Best Practices

A. Robust Test Orchestration

- **BDD Approach:**
  We used Behavior-Driven Development (BDD) to express test cases in business-readable language, ensuring clarity and alignment between technical and non-technical stakeholders.
- **Parametrized & Data-Driven Tests:**
  Negative scenarios (e.g., invalid phone numbers, missing fields) are handled through data-driven testing, allowing easy addition of new test cases with minimal code changes.

B. Smart Synchronization & Reliability

- **Stabilized Element Monitoring:**
  When checking customer/user counts, we implemented a “smart wait” to ensure the displayed number has stabilized before asserting its value. This avoids flaky tests caused by dynamic updates.

C. Real User Validation

- **Video Recording:**
  Each test run records a video, auto-named for traceability. This is essential for debugging and providing stakeholders with visual proof of test execution.
- **Comprehensive Reporting:**
  Allure and HTML reports provide clear, actionable insights into test results, including step-by-step logs and links to video files.

D. Modern Coverage: Browsers & Mobile

- **Cross-Browser and Mobile Testing:**
  The test suite can run on Chromium, Firefox, WebKit, and mobile emulations. This ensures functionality across all major platforms and device types.
- **Parallel Execution:**
  All test cases are designed to run independently and in parallel, greatly reducing execution time and increasing scalability.

E. Smart Form Handling

- **End-to-End Submission Flow:**
  The demo request form automation covers both positive and some negative paths, including monitoring API/network requests to ensure backend submission actually occurred and validating the appearance of success/error banners.
- **Validation of Error Handling:**
  Tests not only check successful submissions but also actively trigger and verify error handling for invalid or missing fields, ensuring the form's robustness against bad data.

---

3. Design Rationale

- **Selector Resilience:**
  All selectors are centralized in page objects. This makes maintenance trivial if the UI changes and increases test resilience to minor front-end updates.
- **Extensibility:**
  The modular structure allows for straightforward addition of new test cases, browsers, or environments without significant refactoring.
- **Non-Intrusive & Clean:**
  Tests avoid relying on "sleep" or global timeouts except where waiting for dynamic third-party elements is strictly necessary. Instead, waits are event- or state-based wherever possible.
- **API Monitoring (Where Feasible):**
  The suite intercepts network requests when submitting forms to verify that expected API calls (and responses) take place, catching issues that might not be visible in the UI alone.

4. Continuous Integration (CI) with GitHub Actions

This repository includes a GitHub Actions workflow for automatic test execution.

What It Does

- Triggers on every push and pull request.
- Runs the full Playwright test suite across:
    - Chromium (desktop)
    - Firefox (desktop)
    - WebKit (desktop)
    - Mobile (iPhone 12 emulation)