**TEST AUTOMATION FRAMEWORK BEST PRACTICES**

**🛠️ Project and Language**

* Playwright + TypeScript

**🗂️ Project Structure & Configuration**

1. All implementation code resides under src/ and tests/ is reserved only for test cases.
2. Clear and scalable folder structure using **Page Object Model (POM)**.
3. tsconfig.json configured for strict TypeScript compilation.
4. Use .env files with dotenv for environment setup (dev, uat, prod).
5. Global config under config/ directory:
   * Loads environment variables
   * Resets authentication state
   * Includes resolver/ for handling local vs CI/CD runs
6. Use package.json scripts to:
   * Run tests
   * Check linting
   * Handle authentication and encryption tasks

**🔐 Security & Authentication**

1. **Cryptography**
   * Use Argon2 for hashing
   * Use AES for secure data encryption (at rest & in transit)
2. **Authentication State Management**
   * Store and reuse logged-in session state across tests for efficiency
3. **Encryption Tests**
   * Include tests and README instructions for generating secret keys and encrypting test data

**🧩 Page Object Model (POM)**

* Abstract BasePage for shared methods
* Strongly typed page classes extending BasePage
* Private/protected members; expose only interaction methods
* Use playwright recommended locators. For reference: <https://playwright.dev/docs/locators>
* Reusable component objects for common elements
* Method chaining for readability
* Wait strategies implemented inside POM, not test files
* JSDoc comments for public methods

**🧱 UI Architecture**

1. ui/ directory with:
   * BasePage for common interaction methods
   * Page classes for each screen
2. DI-based fixtures for centralized page management
3. skipBrowserInit option to avoid unnecessary browser launch (e.g., for encryption tests)

**🧪 Test Design**

1. Follow consistent naming: describe('Feature'), test ('should ...')
2. Decide between should vs verify — stay consistent
3. Use Arrange-Act-Assert structure
4. Write atomic tests
5. Use TypeScript interfaces for:
   * API responses
   * Test data
6. Parameterized tests for data-driven coverage
7. Tag tests: @smoke, @regression, etc.
8. Cover positive, negative, and edge case scenarios
9. Group tests by type: encryption/, ui/, api/

**🧼 Code Quality & TypeScript Practices**

* Configure ESLint with TypeScript plugin
* Set up Prettier for code formatting
* Enforce strict typing (minimize any)
* Use interfaces, enums, and utility types
* Add custom types for third-party libs as needed
* Type guards for complex types
* Pre-commit hooks for linting & formatting

**⚙️ Utilities & Helpers**

**✅ AsyncFileManager**

* Modern promise-based file utility
* Safe read/write
* Directory & path validation
* Integrated logging

**✅ DataStore**

* Global in-memory key-value store for test state sharing

**✅ Sanitization Utility**

* Masks sensitive data in logs and API responses

**🧯 Error Handling**

1. Centralized error handler
2. ErrorCategory enum (API, DB, UI, auth, etc.)
3. ErrorProcessor utility:
   * Deduplicates, sanitizes, categorizes, logs
4. API Error Response Builder for REST-friendly output
5. Sanitization to remove stack traces, sensitive paths, headers

**📊 Logging & Reporting**

1. **Logging**
   * Use Winston logger
   * Multiple log levels: info, warn, error, debug
2. **Reporting**
   * Allure or Playwright HTML reporter
   * Screenshot capture on test failure
   * Video recording for failed tests
   * Trace viewer for detailed failure analysis

**🔁 Test Execution & CI/CD**

1. playwright.config.ts configured for:
   * TypeScript
   * Browser matrix (Chrome, Firefox, WebKit)
   * Parallel execution
   * Shared browser contexts
   * Retries for flaky tests
2. CI/CD Integration
   * GitHub Actions / Jenkins pipeline
   * Environment variable handling
   * Artifact storage for screenshots, videos, traces

**🔧 Maintenance & Scalability**

* Onboarding guide and framework documentation
* Regular framework maintenance reviews
* Test data generation utilities
* Data cleanup utilities
* Test coverage reporting
* Fixture sharing and isolation
* Regular dependency updates
* Performance monitoring of test suite