Created by: Lamhot Siagian

Top 111 Selenium Interview Questions & Hints

12 Cards | Questions + Hints | Selenium WebDriver + TestNG/JUnit

1. Selenium Architecture — 10 Qs

- **Explain Selenium components.** Selenium WebDriver (W3C protocol), Grid (distributed runs), IDE (recorder).
- **W3C WebDriver vs JSON Wire.** W3C is the standardized spec; eliminates vendor translation layers.
- **How commands flow?** Client bindings → HTTP request → driver (e.g., ChromeDriver) → browser → response.
- **Role of browser drivers.** Translate WebDriver commands to browser-native automation endpoints.
- **When to use RemoteWebDriver?** For Grid/cloud providers; URI points to Selenium Grid hub or vendor endpoint.
- **Capabilities vs Options.** Use `ChromeOptions` / `FirefoxOptions` which populate capabilities under the hood.
- **Session lifecycle.** Create session → perform actions → quit; avoid reusing stale sessions.
- **Driver-manager tools.** Use WebDriverManager or vendor-managed drivers to avoid manual binaries.
- **Common flakiness sources.** Race conditions, unstable locators, network latency, animations.
- **Selenium limitations.** No native file dialogs, OS windows; use workarounds (Robot/AutoIt) or hooks.

2. Selenium WebDriver Basics — 9 Qs

- **Core navigation APIs?** `get`, `navigate().to/back/forward/refresh`, window maximize.
- **Element interactions.** `click`, `sendKeys`, `clear`, `submit`, `getText/getAttribute`.
- **Find element vs elements.** `findElement` returns first; `findElements` returns list (possibly empty).
- **Handling stale elements.** Re-locate after DOM updates; prefer wait conditions.
- **Screenshots.** Use `TakesScreenshot`; element-level via `getScreenshotAs`.
- **Cookies & storage.** Use `manage().getCookies()`, add/delete cookies for session control.
- **Timeouts config.** `implicitlyWait`, `pageLoadTimeout`, `setScriptTimeout`.
- **Driver cleanup.** Always `quit()` in teardown to release resources.
- **Thread safety.** Don't share WebDriver between threads; use one driver per test/thread.

3. Locators Strategy — 10 Qs

- **Preferred locator order?** ID → Name → CSS → XPath; use test IDs when available.
- **Robust CSS patterns.** Attribute selectors, starts-with/contains (`^=`, `*=`), nth-child.
- **Dynamic XPath tips.** Use `contains()`, `starts-with()`, normalize-space, relative axes.
- **Avoid brittle locators.** No auto-generated IDs or deep absolute XPaths.
- **Shadow DOM strategy.** Use JS execution or frameworks (Selenium 4 supports `getShadowRoot()`).
- **ARIA/accessibility hooks.** Prefer semantic roles/labels when stable attributes exist.
- **Dealing with iframes.** Switch to correct frame before locating; wait for frame readiness.
- **Custom data-test attributes.** Collaborate with devs to add `data-testid` or `data-qa`.
- **Locator debugging.** Use browser DevTools, $x/$, copy selectors, and highlight scripts.
- **Internationalization locators.** Avoid text-dependent selectors for multi-locale apps.

4. Waits & Synchronization — 9 Qs

- **Implicit vs Explicit vs Fluent waits.** Prefer explicit/fluent for targeted conditions; avoid mixing implicit with explicit.
- **Common ExpectedConditions.** Visibility, clickability, presence, urlContains, textToBe, frameToBeAvailable.
- **Avoid fixed sleeps.** Use condition-based waits; consider polling frequency.
- **AJAX/spinner handling.** Wait for invisibility of loader + a stable UI state.
- **Navigation waits.** Wait for document ready or specific element signaling load.
- **Stability over speed.** Tune timeouts per suite; prioritize deterministic waits.
- **Animation issues.** Wait for element to be stable/not moving before clicking.
- **Network idleness caveat.** SPAs may keep requests open—prefer UI readiness signals.
- **Flake triage using logs.** Collect browser logs/network traces to confirm timing issues.

5. Page Object Model (POM) — 9 Qs

- **Why POM?** Separates UI mapping from test logic; improves maintainability and reuse.
- **PageFactory pros/cons.** Convenient annotations, but lazy init and stability vary; consider plain objects.
- **Composition over inheritance.** Prefer small components/composables over deep class hierarchies.
- **Fluent APIs.** Return `this` or next page for readable chains; avoid anti-patterns.
- **Single Responsibility.** Each page encapsulates locators + actions for that page only.
- **Data vs action separation.** Keep test data external; pages expose behavior.
- **Handling async components.** Expose wait-until-ready helpers inside page objects.
- **DRY locators.** Extract common widgets (headers, modals) into components.
- **Test readability signals.** Tests should read like user workflows using page methods.

6. Test Automation Framework Design — 10 Qs

- **Architecture styles.** Hybrid: POM + data-driven + keyword for flexibility.
- **Test runner choice.** TestNG vs JUnit 5; parallelism, data providers, tagging.
- **Project structure.** /tests, /pages, /components, /data, /utils, /drivers, /reports.
- **Config management.** Externalize envs (YAML/JSON), use owners/picocli, avoid hard-coding.
- **Logging & observability.** Use Log4j/SLF4J; correlate test IDs with backend logs.
- **Retry & flake control.** Retry analyzers, quarantine lists, fail-fast switches.
- **Test data strategy.** Factories, builders, synthetic data; reset/cleanup hooks.
- **Parallel-friendly design.** No shared mutable state; thread-local drivers; isolated artifacts.
- **Versioning & releases.** Semantic versioning for framework libs; changelog with migration notes.
- **Security & secrets.** No secrets in code; use vault/CI secrets; scrub logs.

7. Handling Alerts, Frames, and Windows — 9 Qs

- **Alerts API usage.** `switchTo().alert()` → accept/dismiss/sendKeys/getText.
- **iframe handling.** Switch via index/name/webelement; ensure correct context before actions.
- **Multiple windows/tabs.** Use window handles; iterate/switch back after action.
- **Wait for new window.** Poll until handle count increases; then switch.
- **Modal vs alert.** Modal is DOM-based; alert is browser-level.
- **Nested frames.** Switch stepwise; provide reliable anchors at each level.
- **Auth popups.** Use URL creds, capabilities, or robot libraries depending on browser.
- **Download dialogs.** Prefer headless auto-download dirs or browser prefs to suppress prompts.
- **Focus issues.** Ensure correct window/frame focused before interact.

8. Actions and JavaScriptExecutor — 9 Qs

- **Actions class basics.** Mouse move, click, doubleClick, contextClick, dragAndDrop, key chords.
- **Advanced interactions.** Build/perform chains; handle HTML5 drag with custom JS if needed.
- **When to use JS executor?** Scroll into view, set value, trigger events when native click fails.
- **Scrolling strategies.** JS `scrollIntoView` vs Actions wheel; prefer natural when possible.
- **Reading performance metrics.** Use JS to access `window.performance` entries.
- **Handling stale via JS.** Re-query element in script scope; verify attached/visible.
- **Risks of JS actions.** Bypasses user semantics; use sparingly with clear comments.
- **Keyboard interactions.** SendKeys with modifiers; ensure active element is correct.
- **Custom events.** Dispatch synthetic events to simulate complex gestures.

9. Data-Driven Testing — 9 Qs

- **External data sources.** Excel (Apache POI), CSV, JSON (Jackson/Gson), DBs.
- **TestNG @DataProvider.** Supply 2D arrays/iterators; pair with factories for flexibility.
- **Parameterization in JUnit 5.** Use `@ParameterizedTest` with sources; custom converters.
- **Schema validation.** Validate input/output contracts; sanitize before UI input.
- **Env-driven data.** Switch datasets per environment using config flags.
- **Sensitive data handling.** Mask PII; use secrets managers; rotate credentials.
- **Data reset & cleanup.** Idempotent seeding; teardown to avoid polluted state.
- **Faker libraries.** Generate realistic but deterministic data with seeds.
- **Reporting data coverage.** Track which datasets/scenarios were executed.

10. Cross-Browser & Parallel Execution — 9 Qs

- **Why cross-browser?** Rendering/JS engines differ; catch vendor-specific issues.
- **Selenium Grid topology.** Hub + nodes (or Grid 4 distributed); session routing via events bus.
- **Parallel in TestNG.** Use suite XML with `parallel=methods/tests` and thread-count.
- **Vendor clouds.** BrowserStack/LambdaTest/Sauce: use remote URLs and capability maps.
- **Capability management.** Build matrices per browser/version/OS; tag critical flows.
- **Dockerized runs.** Use Selenium Grid with Docker Compose for local parity.
- **Flake in parallel.** Avoid shared state; unique test data; isolate downloads.
- **Headless differences.** Headless may differ in focus/timing; validate critical UI flows headed.
- **CI resource tuning.** Right-size executors; limit max parallelism to avoid contention.

11. Headless and Mobile Testing — 9 Qs

- **Chrome/Firefox headless flags.** Use options like `--headless=new`; configure window size.
- **Mobile emulation in Chrome.** Device metrics/emulation via DevTools options; limited vs real devices.
- **Selenium + Appium.** Appium implements WebDriver for mobile; reuse concepts with mobile-specific locators.
- **Hybrid apps & contexts.** Switch between NATIVE_APP and WEBVIEW in Appium.
- **Touch actions.** Use Appium TouchAction/W3C actions for gestures.
- **Responsive checks.** Vary viewport sizes; validate breakpoints & critical journeys.
- **Performance on mobile.** Network throttling, CPU limits; monitor console logs.
- **Real device vs emulator.** Use emulators for breadth, real devices for fidelity/edge cases.
- **Visual differences.** Font/rendering differences; avoid pixel-perfect assumptions.

12. CI/CD Integration & Reporting — 9 Qs

- **CI choices.** Jenkins, GitHub Actions, GitLab CI, Azure DevOps—use containers for consistency.
- **Triggering test stages.** PR checks for smoke; nightly full/regression; env-specific runs.
- **HTML reports.** ExtentReports, Allure, ReportNG; attach screenshots and logs.
- **JUnit/TestNG outputs.** Publish JUnit XML for CI test summaries.
- **Artifacts & retention.** Store reports, videos, traces per build with expiration.
- **Parallel on CI.** Shard by suite/tags; tune worker concurrency to infra capacity.
- **Secrets in CI.** Use encrypted variables; mask in logs; rotate periodically.
- **Quality gates.** Block merge on critical suite failures; trend flake over time.
- **Notifications.** Post results to Slack/Email; include links to artifacts and failing traces.

Built with ❤️ for fast Selenium interview prep. Total: 111 questions across 12 cards.