Top 40 Automation Testing Interview Questions & Answers

1. What is your automation strategy in your current project?

I follow a structured approach: analyze test cases → identify stable scenarios → design framework → develop scripts → integrate with CI/CD → maintain scripts after every release.

2. Which automation framework have you built? Explain architecture.

A Hybrid POM Framework containing:

* Page Object Model

* TestNG

* Data-driven (Excel/JSON)

* Utilities (waits, logs, screenshots)

* Reporting (Allure/Extent)

* CI integration

3. How do you decide which test cases to automate?

Automate:

* High regression scenarios

* Repetitive flows

* Stable modules

* Data-heavy tests

* High-risk business flows

Avoid:

* UI-changing modules

* One-time tests

* Unstable requirements

4. Explain your Selenium automation workflow.

Driver setup → browser launch → page methods → validations (assertions) → reporting → teardown.

5. How do you handle dynamic web elements?

Using:

* Dynamic XPath (starts-with, contains)
* CSS patterns
* Explicit waits
* JavaScript executor

6. What are the different waits used in Selenium?

* Implicit wait
* Explicit wait
* Fluent wait

7. What is Synchronization?

Handling timing issues between script and web page loading using waits.

8. What is POM (Page Object Model)? Why use it?

A design pattern that stores locators + methods in separate classes.

Advantages: reusability, maintainability, readability.

9. Difference between POM vs PageFactory.

* POM → manual element initialization
* PageFactory → uses @FindBy + initElements()

10. What are TestNG annotations used in your framework?

@BeforeMethod, @AfterMethod, @Test, @DataProvider, @Parameters, @Listeners.

11. How do you run tests in parallel?

Using TestNG XML:

parallel="tests" | "classes" | "methods"

or Selenium Grid.

12. What is Selenium Grid?

A tool to run tests on different machines & browsers in parallel.

13. How do you perform cross-browser testing?

Using WebDriverManager or Grid to run scripts on Chrome, Edge, Firefox.

14. How do you capture screenshots for failed tests?

Using ITestListener + TakesScreenshot interface.

15. What is the difference between Assert and SoftAssert?

* Assert → stops execution

* SoftAssert → continues & logs failures

16. What challenges have you faced in automation?

* Flaky locators

* Timeout issues

* Frequent UI changes

* Script maintenance during agile cycles

* Test environment instability

17. What is your approach when a script fails in CI pipeline?

Check logs → validate environment → re-execute selectively → analyze root cause → fix script or raise defect.

18. How do you handle popups in Selenium?

Using:

* Alert interface

* Window handles

* ExpectedConditions.alertIsPresent

19. What is API Automation? Which tools do you use?

Automating APIs using:

*Postman (Collections + Newman)

* RestAssured for Java projects

20. How do you validate an API response?

Check:

* HTTP status

* Header

* JSON values

* Response time

* Schema validation

21. What is JSONPath?

Used to extract specific values from JSON responses.

22. What is CI/CD? How did you integrate automation?

CI/CD automates build → test → deploy.

Automation integrated using Jenkins + Git → triggers on commit → runs scripts → publishes report.

23. What is your Jenkins pipeline structure?

Checkout → Install dependencies → Run test suite → Generate report → Mail/Slack notification.

24. What is a Hybrid Framework?

Combination of POM + Data-driven + Keyword-driven + Utilities + Reporting.

25. How do you manage test data in automation?

Using:

* Excel

* JSON

* Properties file

* Database

26. Explain DataProvider in TestNG.

Used for parameterized testing — provides multiple sets of input data to a single test.

27. What is Cucumber? Why use BDD?

BDD tool using Gherkin syntax.

Used for:

* Better collaboration

* Human-readable scenarios

* Faster understanding for BA/PO

28. How do you maintain your automation scripts?

Version control (Git) → branching strategy → pull request review → refactoring → removing dead code.

29. If your build fails 10 minutes before a release, what will you do?

Stop full-suite run → execute only critical smoke tests → analyze failing script → verify manually → update stakeholders → fix in next sprint if non-critical.

30. What's the difference between driver.close() and driver.quit() commands?

The *driver.close()* command is used to close the current tab/window of the browser, which is controlled by the Selenium WebDriver.

The *driver.quit()* command closes all browser windows and ends the Selenium

WebDriver session.

31. Explain the differences between ***driver.findElement ()*** **and** ***driver.findElements ()*** **in**

Selenium WebDriver

driver.findElement () returns the first web element matching the provided locator strategy, while *driver.findElements ()* returns a list of all elements that match the locator.

The former is useful when you expect a single element, like clicking a button, and the latter is used when dealing with multiple elements, like a list of search results.

32. What is parameterization in Selenium?

It's the process of parameterizing test scripts (or executing the same test multiple times with different parameters) to automate the passing of data to a specific application during runtime.

Qualified candidates should explain what parameterization is *and* how it functions in Selenium specifically. They should also mention that this process can be conducted several times with various values.

33.How do you get all the options in a dropdown in Selenium?

You can retrieve all options in a dropdown in Selenium by using the *getOptions()* method of the Select class.

Candidates should be able to explain how this method retrieves all options on a Select tag and provides a list of web elements without accepting any arguments.

34. Explain the difference between *getText()* and *getAttribute()* in Selenium

The *getAttribute()* method provides the value of a particular HTML attribute, and the *getText()* method provides the visible text of a web element.

35. What are assertions in Selenium? What are the types?

Assertions are validations (checkpoints) to help the tester keep track of commands for an application.

They help determine whether a test case is behaving as expected by validating the automated test cases so testers can understand if the tests have passed or failed. Look for answers that touch on this definition and explain the two major assertion types:

1. **Soft asserts** help verify certain test conditions in the case even after the assert condition fails. Testers use this assert when passing one test condition is not necessary to complete subsequent tests.

2. **Hard asserts** abort test execution when the test doesn't meet the asserted condition. Candidates should also mention that when there is an assertion error, it displays the java.lang.AssertionError exception.

When the candidate mentions hard asserts, look for answers that list the different types of Hard Assertions in Selenium:

- *assertNull()*
- *assertNotNull()*
- *assertEquals()*
- *assertNotEquals()*
- *assertTrue()*
- *assertFalse()*

36. How would you handle dynamic elements that change attributes or IDs on each page load?

For dynamic elements, you must use robust locators like XPath or CSS selectors based on stable attributes or parent elements. This ensures a reliable way to locate elements even if IDs or attributes change.

Additionally, you must implement explicit waits to ensure the element is present before interacting, using *ExpectedConditions* to wait for specific conditions.

37. Explain how to handle browser cookies in Selenium WebDriver and some different scenarios it applies to.

Browser cookies are small data blocks stored by websites and placed on a user's computer. They are typically placed on a device used to access websites and handle tasks like session management, tracking, and personalization.

Look for answers that provide in-depth knowledge of the key ways to manage browser cookies:

1. Retrieving cookies: Candidates should start by explaining the *get_cookies()* method, which returns a set of dictionaries that each represent a cookie with attributes like name, value, domain, path, expiry, and secure flag.

2. Adding cookies: The *add_cookie()* method takes a cookie dictionary as its parameter, focusing on the most important attributes – name and value.

3. Deleting cookies: To delete cookies, candidates should mention the *delete_cookie()* method, which takes the cookie's name as a parameter.

4. Handling individual cookies: Candidates should know and explain how Selenium WebDriver allows the tester to manage cookies individually. For example, the tester can grab a specific cookie using its name and change its attributes with the first two

methods mentioned above.

5. Handling expiry time: Ideal answers include an understanding of expiry time and Selenium's ability to set a cookie's expiry time with the attribute in the cookie dictionary.

Managing browser cookies is important for these scenarios:

- Logging in
- Maintaining a session's status
- Performing cookie-based function tests
- Executing various automation tasks

38.Explain the concept of the FluentWait class in Selenium and how you would implement it.

FluentWait is an advanced version of WebDriverWait, allowing custom polling intervals and ignoring specific exceptions. It's useful for handling scenarios where elements might appear/disappear over time.

To implement it, you must create a FluentWait instance, set up its conditions and polling intervals, and then use it to wait for an expected condition to be satisfied.

39.What is the difference between *getWindowHandle()* and *getWindowHandles()*?

getWindowHandle() returns the unique handle of the currently focused window or tab.

getWindowHandles() returns a set of all open window handles. This is especially handy when dealing with multiple windows or tabs, allowing easy switching between them.

40. How do you perform drag-and-drop operations in Selenium WebDriver?

To perform drag-and-drop, you must use the Actions class. You call *clickAndHold()* on the source element, then *moveToElement()* to the target element, and finally release the mouse using *release()*. This simulates the drag-and-drop action effectively.