**Company: Amadeus Labs – 3.5+ yr Exp. Automation**

**A candidate with 3.5+ years of experience in automation testing would have a solid understanding on**

***Testing methodologies: -***

**Waterfall Model:**

**The Waterfall model follows a sequential and linear approach.**

**It consists of distinct phases such as requirements gathering, design, implementation, testing, and maintenance.**

**Each phase is completed before moving on to the next one**

**Spiral Model:**

**The Spiral model combines elements of both Waterfall and iterative development.**

**It consists of multiple iterations, each involving requirements gathering, design, development, and testing.**

**The model allows for feedback and refinement at each iteration.**

**V-Model:**

**The V-Model is an extension of the Waterfall model.**

**It emphasizes the relationship between each development phase and its corresponding testing phase.**

**Testing activities are planned and executed in parallel with development stages to ensure proper verification and validation**

**Agile Testing:**

**Agile methodologies, such as Scrum or Kanban,**

**emphasize iterative and incremental development.**

**Testing is integrated throughout the development lifecycle, with frequent feedback and collaboration between developers, testers, and stakeholders.**

**Rapid Application Development (RAD):**

**RAD focuses on quick prototyping and iterative development.**

**Testing in RAD involves frequent interaction with end-users and rapid feedback cycles to refine the software.**

**Exploratory Testing:**

**This methodology relies on the tester's domain knowledge and intuition to explore the application,**

**uncover defects, and learn about the system under test in real-time.Test design and execution happen simultaneously.**

**Test-Driven Development (TDD):**

**TDD is an agile approach where tests are written before the development of the actual software.**

**The software is then developed incrementally to pass the tests, ensuring that the code meets the specified requirements.**

**DevOps Testing:**

**DevOps promotes collaboration between development and operations teams,**

**aiming for continuous integration and delivery. Testing is an integral part of the DevOps pipeline, with automated testing frameworks and tools.**

***STLC Stages in Automation:-***

**Test Script Development:**

**write and maintain automation scripts using the Selenium WebDriver API in a programming language like python**

**Test Case Design:**

**analyze test requirements and design test cases that cover various functionalities and scenarios of the web application under test. Test cases would include specific steps, expected results, and assertions to validate the application's behavior.**

**Test Environment Setup:**

**configure the test environment, including setting up browsers, drivers, and any additional dependencies required for running Selenium tests. This would involve ensuring proper installations, configurations, and compatibility of the testing environment.**

**Test Execution:**

**execute automated test scripts using Selenium WebDriver, running them against different browsers and platforms to validate the web application's functionality. This involves interacting with web elements, performing actions, and verifying expected results.**

**Test Data Management:**

**manage test data used during test execution, including input data sets, expected results, and test data repositories. This may involve data-driven testing techniques, where different data inputs are used to drive the same test script.**

**Test Result Analysis and Reporting:**

**analyze test results, identify defects, and report them in a clear and concise manner. This includes logging defects, capturing screenshots or videos, and providing detailed information to assist developers in bug fixing.**

**Test Maintenance and Refactoring:**

**update and maintain automation scripts to ensure their relevance and accuracy. This may involve modifying existing scripts, adding new test cases, or refactoring code to improve script maintainability and efficiency.**

**Continuous Integration/Continuous Delivery (CI/CD) Integration:**

**integrate Selenium automation tests into CI/CD pipelines, enabling automated execution of tests as part of the software delivery process. This ensures quick feedback on the application's quality and helps catch issues early.**

***Tools:-***

**Selenium: Selenium is a widely used open-source automation testing framework for web applications.**

**Selenium provides a suite of tools, including Selenium WebDriver for browser automation,**

**Selenium Grid for distributed testing,**

**Selenium IDE for record and playback.**

**Jenkins:**

**Jenkins is a widely used open-source continuous integration and continuous delivery (CI/CD) tool.**

**Automation testers utilize Jenkins to automate the build, test, and deployment processes.**

**It helps in scheduling and orchestrating test executions, integrating with various testing frameworks and tools.**

**SoapUI:**

**SoapUI is a popular tool for testing web services and APIs. It allows automation testers to create, execute, and validate SOAP and RESTful web service requests and responses.**

**SoapUI provides features like data-driven testing, assertion libraries, and reporting capabilities.**

**Postman: Postman is an API testing tool that allows automation testers to create and execute API requests, validate responses, and perform end-to-end API testing. It supports various authentication methods, data-driven testing, and collaboration features.**

**Git:**

**Git is a version control system used by automation testers to manage and track changes to their automation code. It enables collaboration, code versioning, and rollback capabilities**

**Other test automation tools available for Python:**

* **pytest, unit test, Robot Framework, Behave, Appium, PyTest-BDD**

***Programming language for automation testing - Python:-***

**Basic Syntax:**

**Understand Python's syntax, including proper indentation, variable declaration, and basic data types (integers, floats, strings, booleans, etc.).**

**Data Structures:**

**Familiarize yourself with built-in data structures such as lists, tuples, dictionaries, and sets. Know how to manipulate and access elements within these data structures.**

**Control Flow:**

**Understand control flow statements like if-else conditions, loops (for and while), and how to use them in test automation scenarios.**

**Functions and Modules:**

**Know how to define and use functions to encapsulate reusable code blocks. Understand how to import and use modules to access pre-defined functionalities.**

**Exception Handling:**

**Be familiar with handling exceptions using try-except blocks to catch and handle potential errors during test execution.**

**File Handling:**

**Understand how to read from and write to files, as this may be necessary for reading test data or logging test results.**

**Object-Oriented Programming (OOP):**

**Have a basic understanding of OOP concepts such as classes, objects, inheritance, and polymorphism. This knowledge is beneficial for creating test frameworks or building reusable test libraries.**

**Assertions and Unit Testing:**

**Understand how to use assertions in Python to validate expected results and perform unit testing. Familiarity with testing frameworks like unit test or pytest is valuable.**

**String Manipulation:**

**Be comfortable with string manipulation techniques such as concatenation, slicing, formatting, and regular expressions. This knowledge is often used in test data manipulation and validation.**

**Working with APIs:**

**Understand how to make HTTP requests, handle responses, and parse JSON/XML data using Python libraries like Requests or built-in modules like urllib.**

**Libraries and Frameworks:**

**Familiarize yourself with popular Python libraries and frameworks used in test automation, such as Selenium WebDriver.**

**Debugging:**

**Be proficient in debugging techniques using tools like breakpoints, print statements, and logging to identify and troubleshoot issues in your test scripts.**

**Python Packaging and Virtual Environments:**

**Understand how to create Python packages, install external dependencies using package managers like pip, and set up virtual environments to manage project dependencies effectively.**

***Selenium Framework:-***

**What is Selenium?**

**Selenium is an open-source framework used for automating web browsers. It provides a set of tools and libraries that enable testers to automate browser actions and test web applications.**

**Selenium consists of three main components:**

* **Selenium WebDriver:**

**It provides a programming interface to interact with web browsers and automate user actions.**

* **Selenium IDE (Integrated Development Environment):**

**It is a record and playback tool used for creating simple automation scripts.**

* **Selenium Grid:**

**It allows for distributed test execution on multiple machines or browsers simultaneously.**

**programming languages are supported:**

**Selenium supports multiple programming languages, including Java, C#, Python, Ruby, and JavaScript.**

**Testers can choose the language they are most comfortable with for writing Selenium scripts.**

**Selenium WebDriver and Selenium IDE:**

* **Selenium WebDriver is a powerful API that provides a programming interface to interact with web browsers programmatically. It allows for more advanced automation scenarios and supports various programming languages.**
* **Selenium IDE is a simpler tool that provides a record and playback feature, but it has limited functionality compared to WebDriver.**

**concept of locators in Selenium.**

**Locators are used to identify elements on a web page. Selenium provides various types of locators, such as ID, name, class name, CSS selector, and XPath. Testers use these locators to locate and interact with web elements during test automation.**

**absolute XPath and relative XPath.**

**Absolute XPath specifies the complete path of an element from the root node, starting with a forward slash (/).**

**Relative XPath, specifies the path of an element relative to the current node or any parent/ancestor node.**

**Relative XPath is generally preferred as it is more flexible and robust to changes in the structure of the web page.**

**dynamic elements in Selenium:**

**Dynamic elements are those that have attributes or values changing dynamically on a web page.**

**testers can use techniques like using relative XPath, CSS selectors, or dynamic element identification strategies (e.g., using regular expressions or partial attribute values).**

**synchronization in Selenium:**

* **Synchronization refers to the technique of ensuring that the automation script waits for the web page to load completely or for a specific element to appear before performing any actions.**
* **synchronization methods, such as implicit waits, explicit waits, and fluent waits, to handle synchronization issues and avoid timing-related errors.**

**pop-up windows or alerts in Selenium:**

**Selenium provides methods to handle pop-up windows and alerts. Testers can switch to the new window or frame using window handles and interact with elements on the pop-up window. They can also accept or dismiss alerts using methods like switchTo().alert().accept() or switchTo().alert().dismiss().**

**advantages of using Selenium for test automation:**

* **Open-source and free to use.**
* **Supports multiple programming languages.**
* **Provides cross-browser compatibility.**
* **Supports parallel and distributed testing.**
* **Large community support and active development.**
* **Integration with various test management tools and frameworks**

***Test planning:-***

**Defining the approach and strategies for testing a software application.**

**comprehensive test plan that outlines the scope, objectives, test deliverables, test environment, test schedule, and resource allocation for the testing activities.**

**Here are the key steps involved in test planning:**

**Understand Requirements:**

* **Review the software requirements specification (SRS) document or any other relevant documents to understand the project's scope, features, and functionality.**
* **Identify any gaps in the requirements that need clarification from stakeholders.**

**Define Test Objectives:**

**Determine the specific goals and objectives of the testing effort. This could include verifying functional requirements, testing specific user scenarios, validating performance and security aspects, or ensuring compatibility across different platforms.**

**Identify Test Deliverables:**

**Identify the documents and artifacts that need to be created as part of the testing process. This may include test plans, test cases, test data, test scripts, test environments, defect reports, and test summary reports.**

**Define Test Scope:**

**Clearly define the boundaries of the testing effort by identifying what is to be tested and what is not. Consider factors such as the modules or functionalities to be tested, supported browsers or platforms, and any third-party integrations that need to be considered.**

**Determine Test Strategy:**

**Develop a high-level approach to testing by determining the test levels (unit testing, integration testing, system testing, etc.) and test types (functional, performance, security, etc.) that will be employed.**

**Consider factors such as test automation, risk assessment, and prioritization of test cases.**

**Plan Test Environments:**

**Identify the necessary hardware, software, and network configurations required for testing. Ensure that the test environment closely resembles the production environment to accurately simulate real-world conditions.**

**Estimate Resources:**

**Estimate the human and infrastructure resources required for testing.**

**This includes identifying the number of testers, test environment setup, tools, and any additional resources such as training or external support.**

**Create Test Schedule:**

**Develop a timeline or schedule that outlines the start and end dates of various testing activities.**

**Allocate time for test design, test execution, defect tracking, and retesting.**

**Consider dependencies with other project activities and factor in any risks or constraints that may impact the schedule.**

**Identify Test Risks:**

**Assess the potential risks associated with the testing process and the impact they may have on the project.**

**Identify mitigation strategies to address these risks and develop contingency plans to minimize their impact.**

**Define Test Metrics:**

**Determine the metrics that will be used to measure the progress and effectiveness of the testing effort.**

**This may include metrics such as test coverage, defect density, test execution status, and defect closure rate.**

**Obtain Stakeholder Approval:**

**Review the test plan with relevant stakeholders, including project managers, developers, and business analysts.**

**Incorporate their feedback and obtain their approval before proceeding with the execution phase.**

***Test design Techniques:-***

**Test design techniques in software testing are systematic approaches used to determine what test cases should be executed in order to verify the correctness, completeness, and quality of a software application.**

**These techniques help testers in identifying and selecting the most effective set of test cases to achieve maximum test coverage.**

**Here are some commonly used test design techniques:**

**Equivalence Partitioning:**

**This technique divides the input data into partitions or groups based on their equivalence classes, where all members of each class are expected to exhibit similar behavior.**

**Test cases are then selected from each partition to ensure that each class is represented and tested.**

**Boundary Value Analysis:**

**This technique focuses on testing the boundaries or extremes of input values. Test cases are designed to exercise the minimum and maximum values, as well as values just below and above the specified limits, as these tend to be error-prone areas.**

**Decision Table Testing:**

**Decision tables are used to model complex business logic or rules. This technique involves creating a table with combinations of inputs and their corresponding expected outputs.**

**Test cases are derived from different combinations to cover all possible scenarios.**

**State Transition Testing:**

**This technique is used when a system or component has different states and transitions between them. Test cases are designed to exercise the transitions between states, ensuring that the system behaves correctly in each state and during state changes.**

**Pairwise Testing:**

**Also known as all-pairs testing, this technique focuses on covering all possible combinations of input parameters with a minimal number of test cases. It utilizes the observation that most defects are caused by interactions between pairs of input parameters.**

**Decision Coverage/Condition Coverage:**

**These techniques aim to ensure that every decision point or condition in the code has been exercised by at least one test case. This helps identify potential gaps in the test coverage and ensures that all branches of the code are executed.**

**Error Guessing:**

**This technique relies on the tester's experience, intuition, and knowledge of common errors or problem areas in software. Test cases are designed based on assumptions about where defects are likely to occur.**

**Use Case Testing:**

**This technique focuses on testing the system's functionality from end-to-end based on the defined use cases or user scenarios. It helps ensure that the software meets the users' requirements and performs as expected in real-life situations.**

***Test Execution Phase:-***

**During the test execution stage in software testing, These concepts contribute to the effective and efficient execution of test cases and the overall assessment of software quality.**

**Here are some key concepts in the test execution stage:**

**Test Case Execution:**

**Test cases are executed by running the test procedures and steps outlined in the test plan or test scripts.**

**The primary objective is to observe the software's behavior and compare it with the expected results to identify any deviations or defects.**

**Test Environment:**

**The test environment refers to the hardware, software, and network configuration in which the tests are conducted. It should closely resemble the production environment to ensure accurate and realistic results. The test environment setup should be well-documented and maintained properly.**

**Test Data Management:**

**Test data is the input provided to the software during testing. Managing test data involves creating, selecting, and configuring appropriate data sets for each test case. It includes techniques such as data creation, data retrieval, data masking, and data cleanup to ensure test data integrity and privacy.**

**Test Execution Logs:**

**Test execution logs capture detailed information about the execution process, including test case status, actual results, timestamps, test environment details, and any issues encountered. These logs are essential for tracking progress, troubleshooting, and reporting.**

**Defect Reporting:**

**Defect reporting involves documenting and communicating any discrepancies or issues found during test execution. Defect reports typically include information such as the defect description, steps to reproduce, severity, priority, and supporting artifacts. It enables developers and stakeholders to understand and address the identified defects.**

**Test Execution Metrics:**

**Test execution metrics provide quantitative insights into the progress and effectiveness of the testing effort. Metrics can include test case execution status, defect counts, test coverage, test execution time, and other relevant measurements. These metrics help evaluate the quality and efficiency of the testing process and support decision-making.**

**Test Progress Monitoring:**

**Test progress monitoring involves tracking and analyzing the test execution progress against predefined timelines and objectives. It ensures that the testing effort remains on track, identifies any deviations or risks, and enables timely corrective actions.**

**Regression Testing:**

* **during the test execution stage to validate that existing functionality has not been affected by newly introduced changes or fixes.**
* **ensures that the software maintains its expected behavior after modifications**
* **prevents the reintroduction of previously resolved defects.**

**Test Execution Automation:**

**Test execution automation involves using specialized tools or frameworks to automate the execution of test cases. Automation can speed up the testing process, improve test coverage, and increase efficiency.**

**Test scripts or test cases are automated, reducing the need for manual intervention.**

***Defect tracking:-***

**In the defect tracking stage of software testing, various concepts and activities are involved to effectively manage and resolve software defects.**

**Here are some key concepts related to defect tracking:**

**Defect:**

**A defect refers to a flaw or deviation in the software application that causes it to behave incorrectly or produce incorrect results. Defects can occur due to coding errors, design flaws, or other issues.**

**Defect Tracking System:**

**A defect tracking system is a software tool or system used to track and manage defects throughout the software development lifecycle. It allows testers and developers to record, prioritize, assign, and track the status of defects.**

**Defect Report:**

**A defect report is a document that provides detailed information about a specific defect, including its description, steps to reproduce, severity, priority, and any other relevant details. It serves as a communication tool between testers and developers.**

**Severity:**

**Severity indicates the impact or seriousness of a defect on the system. It categorizes defects based on their potential to disrupt the normal functioning of the software. Common severity levels include critical, high, medium, and low.**

**Priority:**

**Priority determines the order in which defects should be addressed and fixed. It considers factors such as business impact, customer requirements, and project timelines. Defects with higher priority are typically addressed and fixed first.**

**Defect Life Cycle:**

**The defect life cycle defines the various stages through which a defect progresses, from identification to resolution. The typical defect life cycle includes stages like new, open, assigned, fixed, retested, verified, closed, and reopened.**

**Defect Triage:**

**Defect triage is a process of reviewing and prioritizing defects to determine their severity and priority levels. It involves cross-functional collaboration between testers, developers, project managers, and other stakeholders to make informed decisions about defect resolution.**

**Root Cause Analysis:**

**Root cause analysis is a technique used to identify the underlying cause or source of a defect. It involves investigating the defect, analyzing the system or code, and determining the factors that led to its occurrence. Identifying the root cause helps in implementing effective corrective actions.**

**Regression Testing:**

* **during the test execution stage to validate that existing functionality has not been affected by newly introduced changes or fixes.**
* **ensures that the software maintains its expected behavior after modifications**
* **prevents the reintroduction of previously resolved defects.**

**Defect Metrics and Reporting:**

**Defect tracking also involves capturing and analyzing defect-related metrics, such as defect density, defect age, open and closed defect counts, defect trends, etc.**

**These metrics help in assessing the quality of the software, identifying bottlenecks, and making data-driven decisions.**

***Test Automation Tools:***

**hands-on experience with**

* **Selenium WebDriver - automating web browsers.**
* **Pytest - writing and executing tests**
* **designing, developing, and maintaining automated test scripts – python**

***Scripting and Programming:***

**scripting and programming skills.**

**object-oriented programming (OOP) principles**

***Web and Mobile Testing:***

**worked extensively on web and mobile testing projects.**

**testing web applications across different browsers and platforms,**

**testing mobile applications on various devices and operating systems.**

***Test Management and Reporting:***

**test management tools like Jira, TestRail, or Zephyr to organize test cases, track test execution, and report defects.**

***Continuous Integration and Delivery (CI/CD):***

**have integrated automated tests with CI/CD pipelines.**

**tools like Jenkins, GitLab CI/CD, to automate test**

***1) Tell me about yourself:***

* **"Sure! My name is [Your Name], and I have [X] years of experience as an Automation Testing Engineer. I have a strong background in designing and implementing test automation frameworks and scripts.**
* **Throughout my career, I have worked extensively with tools like Selenium WebDriver, JUnit, and I'm proficient in programming languages such as Python. I have hands-on experience in web and mobile testing**
* **I have a solid understanding of testing methodologies, including test planning, test case design, and test execution. I am well-versed in agile development practices, and I have successfully integrated test automation into CI/CD pipelines using tools like Jenkins.**

***2) Tell me about your daily day to day activities:***

**I start my day by reviewing the project requirements and understanding the features or functionality to be tested. I collaborate with the development teams to discuss the test scope, test strategy, and prioritize test cases. - -----** Test Planning:

**Once the test cases are finalized, I focus on creating automation scripts using suitable tools and frameworks like Selenium.**

**I write scripts in programming languages Python------** Test Script Development:

**After the test scripts are ready, I execute them to validate the application under test. I run automated tests on different environments, platforms, and browsers to ensure the software functions correctly and meets the required quality standards.**

**I analyze the test results, report any defects found, and work with the development team to troubleshoot and resolve issues.-----** Test Execution:

**creating reusable functions, implementing data-driven or keyword-driven testing, or integrating tools for test management, reporting, and continuous integration. ------** Test Framework Development:

**developers, product managers, and other testers, to discuss testing progress, share updates, and address any issues. -----** Collaboration and Communication: Throughout

**I document my test plans, test cases, and test results to maintain a clear record of the testing activities. This documentation helps in knowledge transfer, future reference, and compliance with quality assurance processes.-----** Documentation

**3) Tell me about your Roles and Responsibilities**

**4) Explain OOPs concepts used in your framework in detail**

**5) Why constructor was used in your framework**

**6) Explain differences between Method vs constructor**

**7) Constructor can be static isn't it?**

**8) Why constructor cannot be static?**

**9) Explain Handling of different drop-downs and write code**

**10) Explain the Handling of Child browser Window pop ups**

**11) Explain differences between FindElement vs FindElements**

**12) Explain all the Different Webdriver APIs available**

**13) What is Webelement**

**14) Explain all the Different Webelement APIs available**

**15) Write Code for fetching all links and click on link with name ClickHere, then navigate to new tab, and then click on link inside this tab, link named as ClickHere2, later close this tab and switch to present browser tab**

**16) Explain differences between Set vs Map**

**17) Explain differences between Interface vs abstract class**

**18) Is Webdriver an interface, are you sure?**

**19) For any given String, write code for fetching and printing the duplicates and also print duplicates count**

**20) Explain Exceptions faced in your project**

**21) How to handle staleelementreference exception, explain with code**

**22) What is @Findby**

**23) Explain differences between / Vs //**

**24) Types of locators available**

**25) Explain Locator Strategy in your Project**

**26) Why ID Locator is best**

**27) How have you handled dynamic web elements**

**28) What is Action API in Selenium and its usage**

**29) Explain Sprint process in your project and what are the ceremonies sprint had in your project**

**30) Explain differences between Assert vs Verify**

**31) How to fetch attribute of Webelement**

**32) How to handle multiple inheritance in java**

**33) How do you achieve multiple inheritance with Interface (explain the approach)**

**34) How to submit a form in Selenium Webdriver**

**35) Have you worked on Jeb and Spock?**

**36) Do you have any questions to ask to me?**

**37) Why TestNG Listeners are used in your Framework**

**38) Which Listeners are used in your Project and Why?**

**39) Why Maven and Log4J?**