**Vijay Kumar Reddy**

* DOB – 10 AUG 1991
* GC ON 09 JUNE 2022
* 2012 June B. Tech
* 2012 July – 2015 Jan (CGI) – 2.5 Years
* Got Married in 2015 Feb
* Came to USA on 2015 March on H4 Visa
* 2015 June to 2016 Feb (H4 EAD Process)
* 2016 March to 2019 April (Progressive\_H4 EAD) – 3 Years
* 2019 May to 2020 Nov (Johnson & Johnso\_H4 EAD) – 1.5 Years
* 2020 Dec to 2022 Jan (Target\_GCEAD) – 1 year
* 2022 Feb to Till Now (Huntington\_GC) – 2 years.

**Sr. Functional Test Lead with (Must have experience with LoanIQ)**

Responsibilities:

1. Experience in Commercial Banking domain, possessing comprehensive knowledge of industry best practices, regulations, and trends.
2. Experience working on End-to-End Product lifecycle in all **Commercial banking** products.
3. Good experience in automation of **consumer banking test cases**
4. Collaborate with cross-functional teams to help understand business and functional requirements, test scenarios, execution.
5. Good experience in Framework development/enhancements
6. Good experience in complete **CICD**
7. Collaborate with development teams to define testing requirements, identify defects, and track resolution.
8. Establish QA metrics and KPIs to measure the effectiveness of testing processes and drive continuous improvement.
9. Ensure effective communication between cross-functional teams, keeping stakeholders informed about progress and challenges.
10. Identify opportunities to enhance processes related to QA, domain knowledge management, and business analysis within the consumer banking domain.

Required Skills/Qualification:

1. 10+ years of in-depth experience in Manual Testing of commercial **Banking domain** either inbuilding a banking product or implementing banking application or providing banking domain-based IT services to Banks and Financial Institutions
2. **5+ years’ Experience in Tools – LoanIQ**.
3. Strong knowledge **of QA methodologies, testing tools, and best practices**.
4. Experience in leading and managing cross-functional teams.
5. Graduates/ Postgraduates in Engineering/ Finance/ Science / Business Administration

**INTRO:**

Having 10 years of working experience as Senior QA Automation and Manual Engineer. I will be involved in creating and enhancing automated testing solutions across various sectors, including **Banking**, **E**-**commerce**, **Insurance**, and **Healthcare.**

**Banking: (Huntington Bank)** In the Banking Project mainly I will be doing **security Testing,** **Performance Testing**, **Functional Testing**, **Database Testing**.

* I played a key role in the development and quality assurance of a robust mobile banking application, I've successfully automated test cases for **native iOS and Android applications** **using Appium and cloud device farms**. This includes ensuring compatibility, performance, and reliability of mobile applications across different devices and platforms.
* Develop and execute test cases to validate **Loan IQ** functionalities. Collaborate with business analysts and users to ensure accurate testing of loan processes.
* In my role, I've extensively worked with **Appium**, demonstrating a deep understanding of the **WC3** protocol, and leveraging tools like **WebDriverAgent**, **XCUITest,** and **UIAutomator2**. This allows for seamless automation of **mobile UI components**, ensuring **robust** and reliable test scripts.
* **Understand and analyze** banking application requirements thoroughly.
* Design test cases covering critical **banking** functionalities, including **account management**, **transactions**, **fund transfers**, and **security features**.
* Implement an automation framework suitable for the banking application's technology stack.
* Automate regression tests to ensure quick and **accurate verification** of software updates.
* Work closely with **developers**, **product** **managers**, and **business** **analysts** to understand requirements and resolve issues promptly.
* Designed and implemented end-to-end automated test scripts using **XCode’s testing framework and Swift scripting.**
* Perform **security testing** to identify vulnerabilities and ensure compliance with banking industry security standards.
* Utilize tools and techniques for penetration testing, **encryption verification**, and **secure** **data handling.**
* Collaborated closely with security experts to conduct regular security assessments and address emerging threats.
* Conducted rigorous testing to guarantee the accuracy and integrity of financial transactions and sensitive user data.
* Integrated automated tests into the **CI/CD pipeline**, ensuring swift and reliable validation of code changes.
* Implemented automated deployment verification to ensure consistency between testing and production environments.
* Worked closely with **developers, product managers, and business analysts** to understand banking **requirements** and **translate** them into effective **test cases.**
* Conducted thorough regression testing to identify and address potential issues in new releases. Validated each release to guarantee the stability and functionality of the **mobile** banking app, collaborating with development teams to resolve any identified issues.

**E commerce Project: (Target)** In this I will be doing **Performance Testing’s like Load Testing, Stress Testing, Scalability Testing, Response Time Testing.**

* The eCommerce project involves the development and maintenance of an online platform for buying and selling products or services.
* The platform includes features such as **product catalog**, shopping cart, user accounts, checkout, payment processing, and order management.
* Design and implement automated test scripts using tools like **Selenium**, or similar frameworks.
* Apply mobile testing expertise **to automate test cases** for E-commerce **mobile** applications using tools like Appium and a cloud device farm.
* Collaborate with developers, product managers, and business analysts to understand requirements and translate them into effective test cases.
* Integrate automated tests into the Continuous Integration/Continuous Deployment (CI/CD) pipeline for quick validation of code changes.
* Conduct regression testing to identify and address potential issues in new releases.
* Validate the **functionality**, **performance**, and compatibility of E-commerce mobile apps, catering to the growing trend of **mobile shopping** and providing a seamless **mobile** experience for customers.
* Perform cross-browser and cross-device testing to ensure a consistent user experience.
* Validate user authentication, authorization, and role-based access control.
* Execute **performance** testing to ensure the platform can handle expected and peak loads.

**Health Care Project: (Johnson & Johnson)** In the Health Care Project mainly I will be doing **Usability and Accessibility Testing, Integration Testing.**

**Insurance Project: (Progressive)** In this I will be doing the **Security** Testing, **Integration** Testing, **Claims** **Processing** Testing, **Policy** **Administration** **System** Testing.

**API Testing**

I worked on Postman for manual API testing. We used to create and execute test cases, validate API responses, and API Integrations. Also, I have experience in building and managing test environments, and collections.

I have worked in **Automating the APIs using Rest Assured libraries in Java**, With Rest Assured, sending HTTP requests to a RESTful web service and validate the response and authentication mechanisms, custom headers, and request/response logging. Using rest assured we can reuse the code and save execution time. I used TestNG to execute the test cases and generate reports, and for assertions and grouping and parallel execution.

**UI testing**

Also, I have experience on UI testing, for this I have used Browser Stack and Sauce Labs for executing on different browsers like Chrome, Safari, Edge, and Firefox to test manually and I also used Selenium to work on a web automation framework across different browsers and platforms. I have experience in **BDD** frameworks like **cucumber** which is built on top of selenium for acceptance testing.

I did tests on UI elements such as textbox, Radio buttons, images, check boxes, file upload, link text, scroll up and down, mouse hover. I have worked on Selenide which is a framework built on top of Selenium that simplifies the usage of Selenium. It provides a more concise API and better error messages.

I also had experience on JAVA concepts such as OOPs Concepts, Overloading & Overriding, Inheritance with Interface and Abstract Class, Exception Handling, Packages, Collections.

**DB testing**

I have experience working in Relational databases like SQL SERVER, MY SQL and in No SQL databases. Expertise in writing simple and complex queries to check the consistency and integrity of the database. I used **JDBC** Template to connect to the database and validate the data, to make sure the data is in the format.

I also have best practices for database testing, such as creating test data, using test cases and scenarios, comparing expected and actual results, reporting defects and errors, and documenting the test results.

**Mobile Testing**

I have experience in working mobile application testing both manual and automation. For testing IOS applications I used test flight in manual testing. I worked on mobile automation testing using Appium. I have tested applications across various devices, screen sizes, and operating systems, ensuring compatibility and consistency.

**Appium** desktop with the drivers

and to connect the phone we pass the desired capability.

and we use **xcuitest** for apple testing.

and we use ui **automator2** for android testing.

I have also used **XCUITest and Expresso frameworks** as well. I have used **Fastlane** for beta deployments and releases for our iOS and Android apps included **CICD pipelines.**

**Security Testing:**

Ensure the robustness of security measures in online banking platforms to protect sensitive financial information.

**Performance Testing:**

Evaluate the responsiveness and scalability of banking applications, especially during peak transaction periods.

**Automation roles in the project**

* **Develop** and **execute** the overall test strategy for the project.
* Create comprehensive test plans, considering functional, **performance**, security, and compliance aspects.
* Creating and maintaining **automated** test scripts for **functional**, regression, and performance testing, ensuring their reusability and maintainability to support ongoing development cycles.
* I've successfully conducted API automation testing using Java with **Springboot2**, Rest Assured, **GraphQL**, and **Postman**. This involves validating API endpoints, ensuring data accuracy, and verifying proper communication between components, contributing to the overall reliability of the software.
* I maintain open communication with developers throughout the development lifecycle. Regular meetings, joint **sprint planning**, and a shared understanding of testing goals ensure that all areas of the application are thoroughly tested.
* I have a solid background in web performance testing, incorporating principles to identify and address performance bottlenecks. This involves **load testing**, **stress testing**, and optimizing for responses.
* I Used to Integrate automated test suites into **CI/CD** pipelines for seamless and continuous testing. I will be executing the automated tests in the **CI/CD** pipeline to quickly check how well the application is working.
* Collaborate closely with **developers** to ensure thorough testing coverage for all areas of the application.
* Identify test cases suitable for automation and **prioritize** them based on business impact.
* Select appropriate tools and technologies for automation based on project requirements.
* Integrate automated tests into **CI/CD** pipelines for continuous testing.
* Executing automated tests, analyze results for failures and **defects**, and collaborate closely with the development team to identify root causes and implement resolutions.
* Organizing test cases into groups and create comprehensive test reports using **TestNG.**
* Collaborate with developers and product managers to **prioritize** **defects** based on their **impact** and **urgency**.

**Test lead roles responsibilities in the project**

* Create, review, and maintain the Test Plan outlining the scope, strategy, resources, schedule, and deliverables for testing.
* Allocate tasks among the testing team members based on their skill sets and strengths.
* Mentor and provide guidance to the testing team.
* Conduct regular team meetings to discuss progress, risks, and challenges.
* Oversee the creation and execution of detailed test cases, scripts, and scenarios.
* Oversee the defect tracking process.
* Prioritize defects in coordination with developers and product managers.
* Regularly update stakeholders, including developers, product managers, and senior management, about the testing progress, risks, and results.
* Participate in meetings with cross-functional teams to provide insights from the testing perspective.
* Review the test artifacts, including test cases, scripts, and strategies, to ensure they meet quality standards.
* Create end-of-cycle reports summarizing testing efforts, results, and metrics.

**Sprint Planning:**

* I am part of a **14-member** team that includes Project Manager, Business Analyst, Product Owner, developers, and testers.
* Our sprint cycle is **2 weeks** long and we follow the agile methodology.
* We use **JIRA** for Task Management tool. In the sprint planning meeting, I reviewed the user stories and acceptance criteria with the BA and the rest of the team. We estimate the size of each task and assign them to the appropriate team members.
* I write **test cases** based on the acceptance criteria and the expected behavior of the software. During the **sprint**, I collaborate with the developers and other testers through GitHub, where we push and review our code. I also attend daily **stand-up meetings** to report my progress and challenges.
* I report and track defects found during testing using Jira platform. I participate in defect triage meetings (**Defect review board**) three times a week to prioritize and assign defects to developers.
* Before the demo session, I perform regression testing to ensure that all the features are working as expected.
* I use **Jenkins** to run my automated test cases and generate reports. I attended the demo session where we presented our work to the **PO** in the **UAT** environment. I provide feedback and suggestions for improvement.
* After the demo session, I attended the retrospective meeting where we discussed what went well and what can be improved in the next sprint. I also prepare for the production release by ensuring that all the defects are resolved, and all the test cases are passed.
* Prior to sprint planning, lead a team meeting to review upcoming user stories and acceptance criteria.
* Actively lead the discussion during the user **story review**.
* Facilitate the estimation process and capacity planning discussions.
* Coordinate with the development team during **sprint planning.**
* Assign specific **testing tasks** to team members.
* Discuss the **automation** strategy for the sprint.
* Collaborate with the team to estimate testing efforts for **each user story**.
* Engage in discussions with the development team.
* **Prioritize** test cases based on feature criticality.
* Collaborate with the **Scrum Master** on capacity planning.
* Communicate effectively during sprint planning and throughout the sprint.
* Lead a **post-sprint planning review meeting**.

**Test case best practices:**

We use Zephyr for test case management, and I had good practices in writing test cases.

* Creating test cases simple and Transparent.
* Create test case with End User in Mind.
* Avoid writing test case Repetition.
* Do not assume, Ensure 100% coverage. (Requirement Traceability Matrix)
* Creating Unique ID for each test case (Easy tracking)
* Implementing testing techniques – Boundary Value analysis, Equivalence Partition
* And peer Reviewing the test cases.

**How does a typical day look like for QA engineer?**

* First thing in the morning is to review emails for any critical issues reported overnight or messages from offshore teams.
* update Jira board to reflect the status.
* participate in daily stand-up meetings (if following Agile) to discuss what was done the previous day, what's planned for today, and any blockers.
* Look for any issues in automation builds especially if there were code integrations or merges the previous day.
* Test new features, changes, or bug fixes manually based on test cases.
* If any issues are found during testing, log them in the defect tracking system with detailed steps, expected vs. actual results, and any relevant screenshots or logs.
* Check resolved issues to confirm they're fixed and close them in the tracking system.
* send a daily status report highlighting progress, issues found, and other critical metrics.
* Make a to-do list for tomorrow.

**CI/CD Pipeline:**

* When Developers push the code to the **GITHUB** Repository. We use **Jenkins** as a Maven Build tool which will build the jar files.
* Jenkins polls with the GITHUB and invokes for any new code changes. where it does Smoke testing, Unit testing, Integration testing, as a tester I do regression testing.
* Jenkins will automate and execute test cases.
* We deploy code to the QA and UAT environment.

**Create an effective bug ticket?**

* Description: A summary of the bug. It should provide a clear picture.
* Priority: Indicates the urgency to fix the bug (e.g., low, medium, high, critical).
* Severity: Indicates the impact of the bug on the system (e.g., minor, major, critical).
* A step-by-step guide to reproduce the bug. This should be detailed enough that someone unfamiliar with the issue can replicate the problem.
* Expected Result
* Actual Result
* Screenshots or screen recordings can be very helpful.
* Logs or error messages that might help in diagnosing the issue.
* Specify the environment in which the bug was observed.
* Mention details like browser version, OS, device type (for mobile issues)

**Bug Life Cycle:**

New: When a new defect is logged and posted for the first time. It is assigned a status as NEW.

Assigned: Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team

Open: The developer starts analyzing and works on the defect fix

Fixed: When a developer makes a necessary code change and verifies the change, he or she can make bug status as “Fixed.”

Pending retest: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is “pending retest.”

Retest: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to “Re-test.”

Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is “verified.”

Reopen: If the bug persists even after the developer has fixed the bug, the tester changes the status to “reopened”. Once again the bug goes through the life cycle.

Closed: If the bug is no longer exists then tester assigns the status “Closed.”

Duplicate: If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to “duplicate.”

Rejected: If the developer feels the defect is not a genuine defect, then it changes the defect to “rejected.”

Deferred: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status “Deferred” is assigned to such bugs

Not a bug: If it does not affect the functionality of the application then the status assigned to a bug is “Not a bug”.

**Healthcare**

In my previous role, I was responsible for the quality assurance of the company's healthcare claims processing system. I worked closely with developers and other QA analysts to identify and fix defects in the system. I also developed and executed test plans to ensure that the system met all the requirements. I do have experience in the healthcare domain, mainly in insurance claims, benefits, and enrolment. I have a strong understanding of EDI 837 transactions as per HIPAA based transactional 5010 formats and in analyzing test EDI 837I and 837P files and modifying them appropriately to create different test data scenarios to validate various business rules of the application. I am also experienced in working with 999 (Acknowledgement), 277CA (Accepted with errors), 278 (Authorization Request and Response), 835 (Claim approval with EOB) type EDI transactions. I have a good understanding of eligibility enquiry (270) and response (271) type EDI transactions. I have worked with edit codes to validate HIPAA compliance of the application. I am also experienced in working with ICD10 and HIPAA (5010).

HIPAA-based Transactional 5010 Formats:

HIPAA mandates the use of specific EDI transaction standards, and the 5010 format is the current version that organizations must adhere to. It defines the structure and data elements for various healthcare transactions, ensuring consistent and accurate data exchange.

In the Health care domain, I have been responsible for creating comprehensive test plans, test cases, and test scripts to thoroughly validate the functionality of various applications, such as electronic health record (EHR) systems, patient management systems, and medical billing software. I have experience in performing functional, integration, regression, and user acceptance testing, often working closely with domain experts to ensure the software meets the specific needs of health care professionals.

In one project, we were dealing with a complex patient data synchronization process across multiple systems. To address this challenge, I developed a data-driven testing approach using Selenium and Excel spreadsheets.

Ensuring the security and privacy of patient information is of paramount importance in the health care domain.

Regularly reviewing and updating our testing procedures and ensuring compliance with industry regulations like HIPAA are also key aspects of maintaining data security.

I have good Experience in writing complex SQL queries to extract, transform, and analyze data from large and intricate health care databases. For instance, I have developed queries to retrieve patient records, medical history, treatment plans, and billing information. These queries often involve multiple joins, subqueries, and aggregate functions to provide comprehensive insights to medical practitioners and administrators.

**Automation framework from scratch using selenium, cucumber, TestNG.**

* In our project we are using Data driven framework by using page object model design pattern and page factory. Initializing the web driver and used different types of locators like Id, name, Xpath, link text, partial link text, CSS selectors and used page object model.
* In page object model, we have maintained class for every web page which holds the functionality of web page.
* We have separate packages for pages and tests.
* Let’s say like for home page and login page we have separate classes to store element locators. For login test there would be a separate class which calls the methods from home page and login page.
* We used **maven** as build tool all the tests are kept in the **'src/test/java'** and remaining files such as config. properties, element locators, test data, etc., kept under **'src/main/java'**.
* **Test Base Class** is responsible for loading the configurations from properties files, Initializing the WebDriver, Extent Reports.
* **Utility class** stores and handles the functions which can be used across the entire framework. The main purpose of creating the utility class is to achieve reusability.
* **The Property files** are used to store the information that remains static throughout the framework such as browser-specific information, application URL, screenshots path, etc.
* **Screenshots** will be captured and stored in a separate folder and the screenshots of failed test cases will be added to the extent reports.
* All the historical test data will be kept in an excel sheet. By using test data xlsx', we pass test data and handle data-driven testing. We use Apache POI to handle excel sheets.
* Used TestNG for Parallel execution. Assertions, grouping.
* We use **cucumber** as BDD framework.
* Create a **feature file** in src/test/resources.
* Create the **Step Definition** class or Glue Code in src/test/java.
* Create a **TestNG Cucumber Runner class** in src/test/java.
* Run the tests from **TestNG Runner.**
* Run the tests from **TestNG.xmI**
* Used Maven for build, execution, and dependency purpose. Integrating the Test dependency in the POM.XmI file and running this POM.xml file using Jenkins.
* When the developer commits the code in git remote repository, the Jenkins will detect it and trigger the CI/CD pipeline. in the CI/CD pipeline first step is build the code, then run unit test and integration test and deployed in the targeted environment and smoke test will be performed there and run the automated tests and allure reports.

**What is cucumber key words ?**

**Feature**: Each feature file begins with 'Feature' keyword. Every feature file should have a feature extension. It gives a summary of what you will be testing.

**Scenario**: Each feature file contains one or more multiple scenarios. Each test is called a scenario followed by three parts.

**GIVEN**: Defines precondition for the test. For example, if you want to verify that a company logo is displaying, the precondition to verify the logo is displaying would be that the user is at the Home Page.

**WHEN**: Defines that an action is performed by the user.

**THEN**: Defines the outcome of the previous steps.

**AND**: If you have multiple 'WHEN,' you can use 'AND.'

**Background**: you'll find yourself repeating the same Given steps in all of the scenarios in a Feature. A Background allows you to add some context to the scenarios that follow it. It can contain one or more Given steps, which are run before each scenario, but after any Before hooks.

Hooks: Hooks are blocks of code that can run at various points in the Cucumber execution cycle. They are typically used for setup and teardown of the environment before and after each scenario.

Scenario Outline:-The Scenario Outline keyword can be used to run the same Scenario multiple times, with different combinations of values. A Scenario Outline must contain One or more Examples. the scenario Outline is run once for each row in the Examples section beneath it (not counting the first header row).

The steps can use < > delimited parameters that reference headers in the examples table. Cucumber will replace these parameters with values from the table before it tries to match the step against a step definition.

Scenario Outline: eating

**Cucumber with selenium Connection**

First, we dependencies

We create Junit/ TestNG runner class.

Use @Runwith annotation. I specify (**Cucumber. Class)** is a parameter.

@cucumber. options I specify (feature is a parameter that provides path of feature file).

**Glue** is another parameter is a path of the step definition class. Plugin generates HTML reports on the provided locations.

**Best Practices writing cucumber Scripts?**

* Each Scenario Should Execute Separately.
* Every feature should be able to be executed along.
* Steps information should be shown independently.
* Connect your scenarios with requirements.
* Keep a complete track of what scenarios should be included in a required document.
* Create modular and easy to understand steps.
* Try to combine all your common scenarios.

***Common UI test cases?***

*Ideally, we can group UI elements into 3 major categories.*

***Input elements*** *– Input elements are responsible for handling different user inputs.*

* *Checkboxes*
* *Dropdowns*
* *Combo boxes*
* *Buttons*
* *Toggles*
* *Text/password fields*
* *Date pickers*
* *Checkboxes*
* *Radio buttons*
* *Confirmation dialogues*

***Output elements*** *– Output elements are responsible for showing results against various user inputs. They also show alerts, warnings, success, and error messages to the users.*

***Helper elements*** *– further divided into navigational, informational, and containers, they help to move through the digital product, get information, and point user’s attention to some element.*

* *Notifications*
* *Breadcrumbs*
* *Icons*
* *Sliders*
* *Notifications*
* *Progress bars*
* *Tooltips*

**Required Fields** - If the screen requires data entry on a specific field, it is good practice to identify the required fields with a red asterisk and to give a friendly warning if the data is left blank.

**Data Type Errors** - If the screen contains dates, numeric, currency or other specific data types, ensure that only valid data can be entered.

**Field Widths** - If the screen contains text boxes that allow data entry, ensure that the width of data entered does not exceed the width of the table field (e.g., a title that allows 100 characters in the database should not allow more than 100 characters to be entered from the user interface).

**Navigational elements** – Verify all navigational buttons on the page are working correctly, and that they redirect users to the right page or screen.

**Progress Bars** - If the screen takes more than 5 seconds to render results, it should contain a progress bar so that the user understands the processing is continuing.

Cosmetic Inconsistencies - The screen look, feel and design should match the other screens in your application. Creating and using a style guide is a great way to ensure consistency throughout your application.

**Save Confirmations** - If your screen allows changing of data without saving, it should prompt you to save if you move to another record or screen.

**Delete Confirmations** - If a user deletes an item, it is a good idea to confirm the delete. However, if your user interface allows deleting several records in a row, in some cases you might consider allowing them to ignore the confirmation as it might get frustrating to click the confirmation repeatedly.

**Error Messages** - Ensure that error messages are informative, grammatically correct, and not condescending.

**Common Cross browser testing UI test cases?**

**Cross Browser Testing** plays a crucial role in website development by verifying that your web application functions optimally across different browsers, operating systems, devices,

**Testing Browser-OS Combinations**:

* Verify compatibility on popular browsers: Firefox, Chrome, Edge, Safari, etc.
* Assess performance on different operating systems: Windows, macOS, iOS, Android, etc.
* Ensure consistent functionality and visual display across various browser-OS combinations.

**Testing on Different Devices:**

* Assess website responsiveness and usability on smartphones, tablets, desktops, laptops, etc.
* Verify that the layout adapts seamlessly to different screen sizes and resolutions.
* Test touch interactions, gestures, and device-specific functionalities.

**Challenges in UI Testing?**

* **Continuous upgrades**: Upgrading constantly to accommodate new features and functionalities means performing comprehensive UI tests is challenging.
* **Increasing complexity**: Modern applications have highly complex features such as embedded frames etc.
* **Cross Browser Adaptability:** This helps ensure that their web apps are easily accessible by users across multiple browsers.
* **Inconsistencies in the Page Layout across Devices**: Web applications tend to suffer from rendering inconsistencies when accessed through devices with different screen sizes. This ensures that the web Validate the API keys for minimum and maximum range/length.
* Verify correct HTTP status code.
* Identify and verify the handling of API error codes.
* Verify response payload Check the JSON Schema validation, the Field Type, and the Mandatory Fields
* Validate the Response headers key and values.

Test request chaining pp is responsive across multiple devices like tablets, smartphones, desktops, etc.

***Common API test cases?***

**Functional Test Cases:**

* Validate the API keys for minimum and maximum range/length.
* Verify correct HTTP status code.
* Identify and verify the handling of API error codes.
* Validate the response headers Key and Values.
* to identify the working of multiple APIs together
* Validate end-to-end CRUD (create, read, update, and delete) flow for application API.
* Check database integrity test cases.
* Verify file upload and download test cases.

**Non-Functional Test Cases:**

* Check that the API is designed according to correct security principles using different authentication mechanisms.
* Role Permissions: ensure that specific endpoints are exposed to the user based on role.
* Check API performance with response time, latency, TTFB/TTLB in various scenarios (in isolation and under load)

Elapsed: TTLB - time to last byte

Latency: TTFB - time to first byte

**Challenges in API Testing?**

* **API Parameter Combinations:**

APIs handle communication between systems by assigning data values to parameters and passing those parameters through data requests. It’s necessary to test all possible parameter request combinations in the API to test for problems pertaining to specific configurations. Adding an additional parameter exponentially increases the number of possible combinations.

* **API Call Sequences:**

When calling an API, a client application sends multiple requests, which must be called in the correct order. If the requests are handled in the wrong order, the program will return an error. An example of this would be the error that comes up when an API call to delete an object is made before the call to create it.

* **Validating Parameters:**

Validating to make sure all parameter data uses the correct string or numerical data type, fits within length restrictions, fits within a designated value range, and passes other validation criteria. For example, U.S. phone numbers should appear in a 10-digit format and returning a 5-digit zip code should trigger an invalidation error.

* **Updating the Schema of API Testing:**

The schema–in other words, the data formatting that handles requests and responses for the API–needs to be maintained throughout the testing process. Any updates to the program that create additional parameters for the API calls need to be reflected in the schema configuration.

**Types of Bugs that API testing detects**

* Fails to handle error conditions gracefully
* Timeout Issues
* Unused flags
* Missing or duplicate functionality
* Reliability Issues. Difficulty in connecting and getting a response from API.
* Security Issues.
* Multi-threading issues.
* Performance Issues. API response time is very high.
* Improper errors/warning to a caller.
* Incorrect handling of valid argument values.
* Response Data is not structured correctly (JSON or XML).

**How to decide what test cases to be automated?**

* If the feature operates on a critical path or is frequently used, it must perform the best. And automating their tests will help in achieving that.
* Regression tests these tests are consuming in terms of time and resources as they’re the backbone of each release’s testing process.
* Data-driven tests or Tests with several combinations: Test cases that involve different combinations or configurations are automated to save time and to avoid human errors.
* Performance tests (load test, stress test, etc.): they are repetitive and time-consuming to reach the desired coverage.

what is CAPA process in QA Testing?

We follow **Corrective Action and Preventive action Process**.

Even though we might do through regression. bugs can happen. CAPA is the process to eliminate those from happening.

**Corrective Action** starts with locating the root cause of the problem, and then taking necessary actions to eliminate the root cause of the problem. It is performed after the occurrence of a defect. We deploy a hotfix on production to fix a production incident caused by the most recent production deployment.

**Preventive Action** is an activity designed to prevent a future (or potential) defect from occurring. Intention here is to stop this problem from occurring in the future.

* Looking for how we missed this in the acceptance criteria. Look at RTM and see how it got missed.
* Root cause analysis process
* Retrospective process
* Reviewing organizational process assets like historical lessons learned, etc.

How to parse Json in Java

1)using Json object and JsoNArray (Java Standard library)

2)using Jackson Library

3)It provides objectmapper class that can map Json to Java objects.

4)using Json library using Json-processing) API Json Path and POJOS.

Created a TEST file under src/test/java to write the test code.

and did Test Execution through TestNG-Runner class and right-click Run as TestNG Test. The tests will run as TestNG.

tests

Run the tests from TestNG.xml-Create a TestNG.ml and run the tests as TestNG.

After the test execution, refresh the project, and a new folder with the name test-output will be generated. This folder contains the reports generated by TestNG.

**selenium web driver Architecture**

* Import the WebDriver library for your programming language.
* Choose a browser and its corresponding WebDriver implementation.
* Write test scripts using WebDriver API methods to automate browser interactions.
* Run the test scripts using a testing framework or a test runner.
* WebDriver sends commands to the browser driver, which in turn controls the browser to perform the desired actions.
* The browser driver communicates with the browser using browser-specific automation mechanisms.
* The browser performs the actions and sends back the results to the WebDriver driver.
* Your test script receives the results and can make assertions or validations based on them.

Selenium Error Debugging skill.

I use breakpoints, screenshots, and session logging to debug my code and identify the root cause of the bug.

**Breakpoints**: Breakpoints are points in the code where you want to pause the execution and inspect the variables, expressions, or outputs. You can use breakpoints to run your code step by step and see how it behaves at each line.

**Screenshots:** Screenshots are images that capture the state of the web page or the browser at a certain moment. You can use screenshots to verify the visual appearance of your web application, compare the expected and actual results, or troubleshoot any UI issues.

**Session Logging**: Session logging is the process of recording the events and actions that occur during a test session. You can use session logging to track the progress and status of your test execution, analyze the performance and errors of your web application, or reproduce any issues or failures.

4.Here's a simplified explanation of testing Android and iOS using Appium:

**1.Install Appium:**

Install Appium on your computer. It's a tool that helps automate testing for mobile apps.

**2. Setup Dependencies:**

Make sure you have the necessary software installed, like Node.js, Java (for Android), and Xcode (for iOS).

**3.Start Appium Server:**

Run the Appium server. It's like a traffic cop that helps communication between your computer and the mobile devices.

4. **Define Device Settings:**

Tell Appium about the mobile device you want to test (name, version, etc.). These are called "desired capabilities."

**5.Write Test Scripts:**

Use a programming language (like Java or Python) to write simple scripts that describe what actions the app should take during testing.

**6.Identify App Elements:**

Specify how to find and interact with elements in your app, like buttons or text fields.

**7.Run Your Scripts:**

Run your scripts. Appium translates your commands into actions on the mobile device, like tapping buttons or entering text.

**8.Check Results:**

See if your app behaves as expected. Appium provides logs and reports to help you understand what happened during the tests.

**9.Repeat for iOS:**

If you want to test on an iOS device, you follow a similar process, but you might need a Mac and Xcode installed.

**10.Make Testing Easier:**

Use tools like Appium Inspector to visually identify app elements, making it easier to write your scripts.

**11.Automate Everything (Optional):**

If you like, you can set up things, so your tests run automatically whenever you make changes to your app.

In short, Appium helps you automate testing for both Android and iOS by allowing you to write scripts that simulate user interactions on mobile devices, helping ensure your app works as expected.

**Playwright:**

* Playwright is a framework for Web Testing and Automation. It allows testing Chromium, Firefox and WebKit with a single API. Playwright is built to enable cross-browser web automation that is ever-green, capable, reliable, and fast. Headless execution is supported for all browsers on all platforms.
* **The primary goal of Playwright is improving automated UI testing.**
* Playwright comes with e integration. For example, the Playwright has Docker images, allowing you to run tests quickly in an isolated and controlled environment.
* They also support your existing JavaScript test runners, like Jest/Jasmine, AVA, and Mocha, which is helpful if you are porting from an existing code base.
* The framework supports cross-browser development on Chromium, Web Kit, and Firefox – including Chrome, Edge, Firefox, Opera, and Safari.
* Cross-platform execution is supported on Windows, Linux, and macOS.
* Testing cross-language, including JavaScript, TypeScript, Python, Java, and .NET – choose the environment that suits you while still covering all areas and formats.
* Auto-wait, smart assertions that retry until an element is found, and test data tracing – keep track of logs and videos easily.
* Built with modern architecture and no restrictions, the application allows you to interact with multi-page, multi-tab websites like a real user and easily handles frames and browser events.
* As the Playwright framework is aligned with the modern browser's architecture, it doesn't have the in-process test runner limitations.
* Playwright delivers full test isolation with no overhead. It creates a browser context for each test which only takes a handful of milliseconds.

**Cypress:**

* I would choose Cypress for its simplicity and fast execution, especially for non-Angular applications.
* Data-driven testing in Cypress involves iterating through test data sets.
* It's beneficial for testing different scenarios without duplicating test code. Data-driven tests enhance test coverage and make scripts more maintainable.
* In my previous role, I extensively used Cypress for end-to-end testing due to its ease of use and robust capabilities. I also have experience with Protractor for Angular applications, Zest for API testing, Enzyme for React component testing, and Mocha for JavaScript unit testing.

**Applying oops concept in Selenium Automation?**

**Page object Design Pattern**

 In Selenium, we call objects as locators (such as ID, Name, Class Name, Tag Name, Link Text, Partial Link Text, XPath, and CSS**). Object repository is a collection of objects**. One of the ways to create Object Repository is to place all the locators in a separate file (i.e., properties or a Page Object file). But the best way is to use Page Object Model. **In the Page Object Model Design Pattern, each web page is represented as a class. All the objects related to a particular page of a web application are stored in a class.**

**Why is the Main method static in Java?**

The main method is always static because static members are those methods that belong to the classes, not to an individual object. So, if the main method will not be static then for every object, it is available. And that is not acceptable by JVM.JVM calls the main method based on the class name itself. Not by creating the object. Because there must be only 1 main method in the java program as the Execution starts from the main Method. So, for this reason the main method is static.

1. **ABSTRACTION**

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.

In Page Object Model design pattern, we write locators (such as id, name, Xpath etc.,) and the methods in a Page Class. We utilize these locators in tests, but we can’t see the implementation of the methods. Literally we hide the implementations of the locators from the tests.

Example: **Login function of LMS (Login Page Module) portal**

A page Class “Login Page” was created to store all the objects or locators of Login Page Module. All functions performed on the UI of Login page are stored as methods in the same Page.

**2. INTERFACE**

WebDriver is an Interface.

WebDriver driver = new ChromeDriver();

WebDriver driver = new FireFoxDriver();

Here, we are initializing Chrome browser using Selenium WebDriver. It means we are creating a reference variable (driver) of the interface (WebDriver) and creating an Object. Here WebDriver is an Interface as mentioned earlier and Chrome driver is a class.

**3. INHERITANCE**

The mechanism in Java by which one class acquires the properties (instance variables) and functionalities of another class is known as **Inheritance.**

We create a Base Class in the Automation Framework to initialize WebDriver interface, WebDriver waits, Property files, Excels, etc., in the Base Class. We extend the Base Class in other classes such Tests and Utility Class.

4. **POLYMORPHISM**

Polymorphism allows us to perform a task in multiple ways.

**METHOD OVERLOADING (Compile time Polymorphism)**

A class having multiple methods with same name, but different parameters is called Method Overloading

We use Implicit wait in Selenium. Implicit wait is an example of overloading. In Implicit wait we use different time stamps such as SECONDS, MINUTES, HOURS etc.

driver. manage (). timeouts (). implicitlyWait (30, TimeUnit.SECONDS).

driver. manage (). timeouts (). implicitlyWait (30, TimeUnit.MINUTES).

driver. manage (). timeouts (). implicitlyWait (30, TimeUnit.HOURS).

**Assert class** in TestNG is also an **example** of **overloading.**

**we have utility methods in our framework we used method overloading for setting up test data.**

**we can pass parameters set up with diferent parameter.**

**METHOD OVERRIDING (Run time polymorphism)**

We use a method which was already implemented in another class by changing its parameters. To understand this, you need to understand Overriding in Java.

webdriver we pass chrome driver or fireforx based on build we change the parameter, browsert type

Declaring a method in child class which is already present in the parent class is called Method Overriding. Examples are **get** and **navigate** methods of different drivers in Selenium.

**5. ENCAPSULATION**

All the classes in a framework are an example of Encapsulation.

In POM classes, we declare the data members using @FindBy and initialization of data members will be done using Constructor to utilize those in methods. Encapsulation is a mechanism of binding code and data (variables) together in a single unit.

settting the password we dont give getter methods

**Rest Assured** from Scratch: -

Add REST Assured and TestNG dependencies to the project.

**Given** ( ) 'Given' keyword, lets you set a background, here, you pass the request headers, query, and path param, body, cookies. This is optional if these items are not needed in the request.

**When** ( ) when' keyword marks the premise of your scenario. For example, 'when' you get/post/put something, do something else.

**Method** ( ) Substitute this with any of the CRUD operations(get/post/put/delete)

**Then** ( ) Your assert and matcher conditions go here.

**RequestSpecification** in Rest Assured can be used to group together common request specs and turn them into a single object. This is interface has methods for defining the base URL, base path, headers, and other parameters. To obtain a reference for **RequestSpecification**, we must use the given ( ) function of the Rest Assured class. We can't make an object out of RequestSpecification because it's an interface.

Its implemented class is RequestSpecificationImp.

Request Specification reqSpec = RestAssured.given ( ).

reqSpec.baseUri ("http: / /localhost: 8080")

reqSpec.basePath (" /employees");

**What are the best practices for developing a maintainable Rest Assured**

**Framework?**

1) Separate test data, test logic, and assertions using a modular approach.

2) Reduce code duplication by implementing reusable helper methods or classes.

3) Variable and method names should be meaningful and descriptive.

4) To make troubleshooting easier, implement effective error handling and logging.

|  |  |
| --- | --- |
| **List** | **Set** |
| An ordered collection of elements | An unordered collection of elements |
| Preserves the insertion order | Doesn’t preserve the insertion order |
| Duplicate values are allowed | Duplicate values are not allowed |
| Any number of null values can be stored | Only one null value can be stored |
| ListIterator can be used to traverse the List in any direction | ListIterator cannot be used to traverse a Set |
| Contains a legacy class called vector | Doesn’t contain any legacy class |

**How to install the Git plugin in Jenkins**

***Step 1***: Launch *Jenkins* and redirected to the ***Jenkins Dashboard***. Now click on the "***Manage Jenkins.*** "

***Step 2***: As soon as we will click on the ***Manage Jenkins*** link, we will be redirected to the *Manage Jenkins page*. Now click on the "***Manage Plugins.*** "

***Step 3***: As soon as we click on the "***Manage Plugins***" link, we will be redirected to the "***Plugin Manager***" page.

* ***If the git plugin is not already installed,****then click on the "****Available****" tab.*
* *Type "****Git Plugin****" in the Filter box and select the****Git Plugin****checkbox after its appearance.*
* *Choose either the "****Install without restart****" or "****Download now and install after restart****" button. If we will choose the "****Install without restart****" option then it will install the plugin without restarting Jenkins and if we choose "****Download now and install after restart****" then the Git plugin will be installed after restarting Jenkins.*

**Step 4**: Now after installing *Git Plugin*, we need to verify it also.

* *Click on the "****Installed****" tab.*
* *Type "****Git Plugin****" in the Filter box. (Once we verify it Appear Installed)*

**How to pull a GitHub repository in Jenkins?**

***Step 1***: Go to ***Jenkins Dashboard*** and click on the "***New Item*** " link to create a new job

***Step2***

* *Enter the Project Name.*
* *Select the Project type as we selected "Freestyle Project".*
* *Click on the OK button.*

**Step 3**: As soon as we click on the "***OK*** " button then we will be redirected towards the ***Configuration*** page. Here we can put the description of our project.

***Step 4:*** Now just scroll down and go to the ***Source Code Management*** section. Now, select the "***Git*** " option.

***Step 5***: Now enter the ***repository URL***

***6th Step***: Now, go to the ***Build triggers section*** and select the option "***GitHub hook trigger for GITScm polling***".

***Step 7***: Go to the ***Build*** section. Now click on "***Execute Windows batch Command*** " after clicking on the ***Add build*** step dropdown. We are using the ***Execute Windows Batch Command***

***Step 8***: Now here we will write batch commands to execute this *Java* Program.

cd <locate the package for source code>

javac Basic/Hello\_ToolsQA.java

java Basic/Hello\_ToolsQA

Now click on the ***Save*** button

***How do I trigger a build automatically in Jenkins?***

***Step 1*: Go to the *GitHub* repository and click on the *settings.***

***Step 2*: Click on the *Webhooks* option listed.**

***Step 3*: As soon as we click on *Webhooks*, we will redirect to the *Webhooks* page. Now, click on the "*Add webhook*” button (give information regarding webhooks.)**

***Step 4*: need to fill in is "*Payload URL.* need to give the *Jenkins URL*.**

***Step 5***: Now perform the below steps to setup webhooks in GitHub.

* *Put the Payload URL in the textbox. Kindly note that doesn't forget to append text****GitHub-webhook/****at the last.*
* *Click on the "****Just the push event****" option.*
* *Please make sure that you check the "****Active****" checkbox.*
* *Click on the "****Add webhook****” button.*
* After clicking on the ***Add webhook*** button, we will see a successful message.

***Step 6***: Now go to code, make some changes, and commit our code again. As soon as we commit out changes, webhook sent the notification to *Jenkins*, and automatically build will be triggered.

***Step 7***: After successfully running the build, we can see results in the console output.

***Step 8:*** After a successful run, we can see the blue sign under the *Build History* section highlighted inside the red rectangle in the below image. After that, click on "***Console Output*** " to see the actual output.

**To configure AWS credentials in Jenkins:**

* On the Jenkins dashboard, go to *Manage Jenkins > Manage Plugins* in the *Available* tab. Search for the Pipeline: AWS Steps plugin and choose Install without restart.
* Navigate to *Manage Jenkins > Manage Credentials > Jenkins (global) > Global Credentials > Add Credentials*.
* Select Kind as AWS credentials and use the ID & Credentials.
* Enter the access key ID and secret access key and choose OK.
* Create Amazon S3 buckets for each Region in the pipeline. S3 bucket names must be unique within a partition:
* Create a file named Jenkins file at the root of the project and add.
* Commit and push the code to the GitHub repository by running following commands:

git commit -am “Adding Jenkins pipeline config.”  
git push origin -u main.

Next, create a Jenkins Pipeline project:

1.From the Jenkins dashboard, choose New Item, select Pipeline, and enter the project name sam-jenkins-demo-pipeline.

2.Under Build Triggers, select *Poll SCM* and enter \* \* \* \* \*. This polls the repository for changes every minute

3.Under the Pipeline section, select Definition as Pipeline script from SCM.

* Select GIT under SCM and enter the repository URL.
* Set *Branches to build* to \*/main.
* Set the Script Path to Jenkins file

4.Save the project.

After the build finishes, we can see the pipeline.

How To Send Email Notification In Jenkins?

**Using Email Extension Plugin**

**Using Default Email Notifier**

**Email Extension Plugin**

*Step 1: Log in to the Jenkins Homepage*

*Step 2: Install Email Extension Plugin*

**Manage Jenkins-> Manage Plugins**

In the available tab search for Email Extension Plugin

*Step 3: Configure System*

Now go to **Manage Jenkins-> Configure System**.

*Step 4: Create Jenkins Pipeline Job*

Now go to Jenkins homepage and create a new job. Name the job with whatever name that you want and select pipeline. Click on OK.

*Step 5: View Console Output*

 Click on Build Number “#1” and click on “Console Output” on the build menu

*Step 6: Check Email.*

**Quality Assurance Vs Quality Control**

**QA** generally focuses on the processes and procedures that increase quality, such as training, documentation, monitoring, and audits. **QC** focuses on the product to identify faults that remain after development. QC specialists discover these vulnerabilities through a variety of methods, including software testing and beta or canary testing.

Spring boot test application framework with selenium testing cucumber

In our last project we built entire framework spring boot.

it saved lot of problems with dependencies etc.

We have WebDriver Beans **for chome,firefox and edge**. We inject the environment variable using @**value** annotation. this is where we use spring profiles for environment specific things.

Our webdriver bean cannot be singleton we need to use spring boot simplethread scope. As this will be an issue with parallel execution.

Since we need webdriver instance for every execution of the test.

and we use **ConditionalOnProperty** annotation with **name** and **havingValue** to set correct webdriver based on **name=browser** **and havingValue=Chrome**

we use the spring dependency for injecting the dependent page. Lets say we have a page which also needs two other pages. we create bean for those.

we use the page object model we define the elements **@FindBy** annotation and Pass how keyword and using key word. we use **PageFactory.initElements** method is used to initialize web elements.

we used the class **AbstractTestNGSpringContextTests** includes the spring **ApplicationContext**. To make it available when executing TestNG test, **AbstractTestNGSpringContextTests** has methods annotated with TestNG annotations like @BeforeClass and @BeforeMethod.

we annotated that with @**SpringBootTest.**

we have common logic for setting up the webdriver etc in that class we invoke the **springTestContextPrepareTestInstance.**

we have testng.xml and we can specify **parallel=true** and set **threadcount** if we need run the classes or methods parallely.

we use cucumber as our BDD framework.

we write feature file which has given when then keyworks with multiple scenarios and create respective **Step defintions** methods with **@Given @when and @And** annotation. This is where again dependency injection will be super easy where we can inject different pages.

for cucumber test runner we extend **AbStractTestNgCucumberTests** class and specify feature and glue parameters. Glue where we give the path to steps.

We create the **CucumberTestContextConfig** and annotate with @**CucumberContextConfiguration** and @**SpringBootTest**

We use hooks for our teardown logic at scenario level and this where we perform if scenario is failed take screenshot do our retry logic if we need to retry.

As I said before we can run through with command line we can specify our testrunner class.

We pass the data between steps using @**ScenarioScope**

we use maven surefile plugin to run in command line and thats the same command we configure in jenkins.

mvn clean test and pass the properties -Dbrowser=chrome

**Evolutyz Interview Questions.**

* Tell me what you will be doing for the last 2 years in your project.
* How to you automate the payments in your nonproduction.
* Do you see any challenges for like purdge Strategy.