**1. If you chose to use a tool or language other than the recommended, briefly explain why?**

**Ans:** I choose to useRest Assured for API Automation, a java Domain Specific Library, which is not your recommendation.

The reason is that I am already using REST Assured currently. I have two days to complete this assessment, within this short period of two days, it is good to use the tool I am already comfortable with rather than trying to learn and not implement a framework with the recommended tool. If this assessment would have demanded only recommended, then I should have definitely implement using the recommended tools only.

Rest Assured has been a very effective API test automation tool, in below aspects. It is based on Behavior Driven Development framework,

* It follows Given/When/Then test notation, makes it more readable and easy to understand.
* REST id Java DSL, so integration into CICD pipeline is easy. Because inorder to implement test reporting on Jenkins using a Java test framework like JUnit or testing. So integration is easy.
* No need to write boilerplate code required to set up an HTTP connection, sending payload and receiving response.
* We can perform schema level validations also using Rest Assured. We can perform data driven testing or use page object model, which are the famous test patterns in industry.
* Most of the API’s that are exposed needs authentication before interacting with them, so Rest Assured supports a number of commonly used API authentication mechanisms.

**2. How do you debug a failed test in your test framework?**

**Ans:** Test cases run in two environment: as part of end to end work flow in Virtual Environment like some automation server (Jenkins pointing to some enterprise server) or in the automation engineer’s computer. So, first analyze the test report or build summary on what all the test cases failed. Lot of times, the error message in test logs is good enough to figure out the real cause of the failure. Are there any assertion failures or any exceptions like Timeout exception, stale element exception, WebDriver exception, null pointer exception, NoSuch element exception etc., If there are any assertion failures, check the expected output vs actual output manually. If there are any exceptions, then do the following

* Set a breakpoint in one or more test methods that are failed. Go each step after the other to verify what’s happening in the test case. If the test case passes in debug but failing in the normal execution mode, then most likely problem would be timing issue due network overhead. So add appropriate timing to the test case. Add Println statements in the test cases while debugging for better understanding.
* Implement screenshot functionality for failure investigation in case of UI Automation, where ever necessary.
* Use custom annotations.
* Check the input data is correct.
* Check the expected results are correct.
* Check the actual outcomes are set correctly.

**3. What do you believe are the most common causes for instability in UI automation?**

**Ans:** For UI Automation:

* **Timing Issue** – this is the most common issue for UI test cases failure because of the **network overhead (asynchronous waits)** between server that is hosting the application and the server or slave that is hosting test automation.
* Too many **Ajax calls** in web application.
* There might be the reason when the previous test run didn’t close correctly.
* Server that is hosting the test automation has **less configuration than required**. Because most of the, this server might be hosting test automation for many projects in the organization. Too much causes network overhead due to **longer loading time.**
* Is input data available to the test or Sometimes test data used in the test cases might be wiped out by someone else because of some database changes? So establishing a database connection from the test case and querying the database for **dynamic data might** be the good option. So that data is available to the test.
* Least chances might be the, **lack of communication** between Automation Engineers and the Developers about the changes or enhancements made in the application.
* Lack of expertise – In case of any application having more **dynamic web elements** to interact with, in that case, resources who are developing automation cases are having lack of knowledge on identify and interact with the dynamic web element.
* In case of, when application is hosted on the public clouds, because of too much of load on the servers, causes so much of **latency**.
* In case of projects using CICI build pipelines, pipeline sometimes doesn’t catch up with integration or regression test suite.
* While locating web elements, use id’s at most, in case of id’s are not available, then use relative paths not absolute paths.
* Bugs keep identifying (This is a good thing).
* If an automated test passes in one run and fails in the next run, without any changes on the software under test, we cannot be certain if the failure is due to the application or due to other factors, such as test environment issues or problems in the test code itself.

For API Automation:

* Network overhead between the Server, that is hosting the application.
* Test data issue.

**4. How do you make your tests consistent and easy to debug?**

**Ans:** Best practice to spend some time to design the test cases before jumping into automation, so that the regression suite can serve in long run – Test Planning.

* Key point is to automate testing in order to get the **consistent results**, so that we can be certain that if something has actually gone wrong when a test fails.
* Use ISTQB test techniques in automation.
* Don’t automate unstable functionality
* Implement event handlers in your framework to deal with unexpected events, popups, and alerts.
* Having a standardized quality assurance process for test development, is good way to catch a bug before going live.
* Using job specific tests - Follow design patterns like **TEST DOUBLES** to achieve this.
* A well designed framework (using any tool, make it a hybrid framework), serves the automation enhance testcases in the future to fit the business needs down the line.

So ability to think of a bigger picture is necessary while developing a framework.

* Understand the app and know what to automate.
* Develop a framework that tells you the exact reason, when a tets case failed.
* Never use Thread.sleep() in test cases.
* Consider Behavior Driven Development (BDD) framework.
* Separate test cases from test framework.
* Using soft assertions in test cases is a good practive, in order to make a list of related checks on the same web page.
* Add comments for every piece of code.
* Follow ‘pesticide paradox’, ‘error seeding’ policy to ensure test code is working correctly.
* All test cases should be independent.
* Creating groups with bug ID and execute or include groups while execution after validating the bug status, in case of TestNG.

**“Measure twice, cut once” is the best thing to keep in mind, while working on any production applications to avoid risks.**