1. What strategies do you use to prevent and diagnose UI automation flakiness? Provide one real example of a time when you identified the root cause of flaky tests and implemented a long-term fix.

   ANSWER:

   First, I prevent flakiness by using explicit waits, stable locators, and ensuring tests validate. For diagnosis, I analyze failure patterns in CI, review screenshots and logs, and rerun failing tests to determine whether the issue is related to timing, environment behavior, or legitimate multiple UI Outcomes.

   A real-world example I encountered involved a toast notification that Selenium intermittently failed to locate. The toast was short-lived and animation-based, so depending on execution speed and environment load, it would either appear briefly or disappear before the locator check ran.

   Initially, the test failed sporadically even though the feature was working correctly.

   To address this, I analyzed the behavior and implemented a conditional approach using if–else logic:

   - The script first attempts to locate the toast using an explicit wait with a short timeout.
   - If the toast is not detected within that window, the script switches to an alternative validation path by extending the wait slightly and validating the resulting state that the toast represents (for example, a successful action reflected in the UI).

2. Describe a CI/CD automation setup you've built or improved. What tool was used, and what measurable impact did it have on the development/release process?

   ANSWER: In my previous role, I worked with a CI/CD pipeline already implemented for our Selenium automation tests. My role involved maintaining the automated test scripts, ensuring they ran reliably in the pipeline, and reviewing the test reports generated after each build.

   Using this setup, I was able to:

   - Quickly identify defects after each commit, reducing the feedback loop for developers.
   - Ensure consistent test coverage for critical functionality without manual intervention.