1. **What are the environment variations you would face when automating mobile tests. What strategies or tools would you use to ensure your tests are robust across various environments in mobile automation?**

   Currently the challenges I can think of are
   - Running test cases on different OS like IOS or android which we could avoid by running our test cases on a cloud service which provides virtual emulators. We can also optimize our scripts to suit both OS and control the type of test suit from a single entry file like the runner.properties in my framework.
   - Device screen size we can solve by optimizing our locators and use dynamic and relational ones.
   - Device language and local settings. To solve this, we can add a hook before each test case to reset the settings to default or define our preferred settings in the device.

2. **What are the best practices for handling dynamic web elements (e.g., elements whose IDs or classes change frequently) in test automation? How can you ensure your automated tests remain stable despite these changes?**

   The best way is to utilize the use of relative Xpaths. It will help us maintain the stability of our automated tests. Relations between elements like parent, child, sibling, preceding-sibling, following-sibling, ancestor, will make our test script adjust with any changes that could happen to the elements IDs or classes.
   Also the use of stable attributes will decrease the hassle. Using attributes like name, value, placecholder, aria-lable, et.

3. **You have a login Page, and when clicking on the login button after inserting the information nothing happened. You reported the defect and the Frontend and Backend team told you that there is no issue from their side. What are the steps needed to identify the root cause and communicate the issue? and list all possible scenarios**

   - Possible invalid credentials with no warning message being displayed by the frontend. Check API response to check the status code payload.
   - Try to eliminate dependencies and try the same scenario in different browsers.
   - Check the request sent by the front end and validate the correctness of the request against your inputs.
   - Check if there any prerequisites to login causes the button click being blocked like captcha or checkbox that is not being shown by the front end.
   - Meet with concerned teams (backend and frontend) and retest and check the logs for any trace of error messages.

4. **Explain how you would test the rate limiting feature of an API. How would you design an automated test to verify that the API responds with the correct status code when the rate limit is exceeded?**

   Rate limiting is used to restrict the number of API calls a client can make in a given time-window.

   - First, I will check if this rate limiting is designed per user using an Auth token or is it a general API behavior.
   - Check the documentation for the API to check the request limit, time-window, and the response behavior when the limit is exceeded. Also, check the response headers if there is a Retry-After header when the limit is exceeded.
   - If not documented, the behavior could be explored manually.
   - I will manually test the API call using postman to confirm the documented behavior to design a stable automated test.
   - I will design multiple test cases
     - One for regular API calls not exceeding the limit.
     - Test case for near-exceeding limits calls.
     - Test case for API calls exceeding the limits.
     - Test case to test API recovery after you exceed the limits (Retry-After)
     - Test case with different time-window and the same number of API calls.
     - Test case for a huge number of calls in a less time-window than the one which triggers the rate limiting exception.
   - In my test cases design, I will use RestAssured with threads to simulate multiple user calls.