ANHTESTER

Automation Testing

BỘ CÂU HỎI PHỎNG VẤN

TEST AUTOMATION

20 câu tình huống — phân theo Role: Junior & Middle

JUNIOR 10 câu tình huống cơ bản hằng ngày

MIDDLE 10 câu tình huống thiết kế & xử lý hóc búa

◆ MỤC LỤC

PHẦN A — JUNIOR

PHẦN B — MIDDLE

PHẦN A — JUNIOR TEST AUTOMATION

JUNIOR

J1. Element có id nhưng id thay đổi mỗi lần load trang (dynamic id) thì bắt thế nào?

★ Tìm phần bất biến, đừng bắt theo id đầy đủ.

► Bám vào phần cố định của id: **contains(@id,'phần_cố_định')** hoặc **starts-with(@id,'...')**.

► Hoặc bắt theo attribute ổn định khác (name, type, placeholder) hoặc theo **text**.

► Hoặc neo vào element cha/label cố định rồi điều hướng tới element cần.

◆ *Không phụ thuộc giá trị sinh động; luôn tìm phần không đổi của element.*

JUNIOR

J2. Click vào button mà báo lỗi 'element not clickable / not interactable' thì xử lý sao?

★ Hiểu vì sao chưa click được rồi mới xử lý.

► Thường do element **chưa sẵn sàng**, bị element khác **che**, hoặc nằm **ngoài vùng nhìn**.

► Dùng explicit wait **elementToBeClickable**; scroll element vào view trước khi click.

► Nếu vướng overlay/animation thì chờ nó biến mất; phương án cuối mới dùng JS click.

◆ *Tìm nguyên nhân thật (bị che / chưa load) thay vì JS click ngay — JS click chỉ che giấu vấn đề.*

JUNIOR

J3. Test pass trên máy bạn nhưng fail trên máy khác / chạy headless thì vì sao và xử lý thế nào?

★ Môi trường chạy không nhất quán là nguyên nhân chính.

► Khác kích thước màn hình → element ngoài viewport: **set window size cố định** (maximize / --window-size).

► Khác version browser / driver → đồng bộ phiên bản.

► Headless render khác → test với headless ngay từ đầu và thêm wait phù hợp.

◆ *Đừng giả định máy nào cũng giống máy dev; chuẩn hóa môi trường chạy.*

JUNIOR

J4. Cần nhập cùng một form với 50 bộ dữ liệu khác nhau mà không muốn copy test 50 lần thì làm gì?

★ Một logic test, nhiều bộ data.

► Tách data ra file ngoài (**Excel / CSV / JSON**) rồi đọc vào.

► Dùng **@DataProvider** (TestNG) truyền từng bộ data vào cùng một test.

► Thêm / bớt data chỉ cần sửa file, không sửa code test.

◆ *Không bao giờ copy-paste test case cho từng bộ data — đó là data-driven.*

JUNIOR

J5. Có bước phải đợi loading spinner biến mất mới thao tác tiếp được thì xử lý thế nào?

★ Chờ đúng tín hiệu, không chờ theo thời gian.

- ► Dùng explicit wait **invisibilityOfElementLocated** cho spinner.
- ► Không dùng **Thread.sleep** vì thời gian load thay đổi từng lần.

◆ *Chờ "spinner biến mất" chắc chắn hơn nhiều so với chờ một số giây cố định.*

JUNIOR

J6. Sau khi submit, trang hiện popup confirm rồi mới qua trang mới — bắt sao cho ổn định?

★ Mỗi lần chuyển trạng thái phải có wait tương ứng.

- ► Chờ popup / modal xuất hiện rồi mới click confirm (wait elementToBeClickable).
- ► Nếu là JS alert: **switchTo().alert().accept()**.
- ► Sau đó chờ tín hiệu trang mới (URL đổi / element đặc trưng) trước khi assert.

◆ *Đừng thao tác "mù" — luôn xác nhận trạng thái mới đã sẵn sàng.*

JUNIOR

J7. Đoạn login bị lặp lại ở rất nhiều test, làm sao cho gọn?

★ Lặp 3 lần trở lên là dấu hiệu cần tách dùng chung.

- ► Tách thành method / Page Object dùng chung (vd **LoginPage.login(user, pass)**).
- ► Đưa vào **@BeforeMethod** nếu mọi test đều cần đăng nhập.
- ► Sau này muốn nhanh hơn thì login qua API thay vì UI.

◆ *Code lặp = khó bảo trì; gom về một chỗ để sửa một lần là xong.*

JUNIOR

J8. Report bị ghi đè mỗi lần chạy, muốn giữ lịch sử để so sánh thì làm gì?

★ Lưu theo từng lần chạy.

- ► Đặt tên file / thư mục report theo **timestamp** hoặc build number.
- ► Dùng report tool có lưu history & trend (vd Allure history).

◆ *Có lịch sử mới so sánh được kết quả qua các lần chạy và phát hiện xu hướng flaky.*

JUNIOR

J9. Bắt được element nhưng getText() trả về rỗng dù trên UI rõ ràng có chữ — vì sao và xử lý?

★ Element tồn tại không có nghĩa nội dung đã render xong.

- ► Text load động, lúc lấy chưa kịp render → **wait cho text xuất hiện**.
- ► Một số ô input giữ giá trị ở attribute value → dùng **getAttribute('value')**.
- ► Nội dung nằm sâu → thử lấy **textContent** qua JS.

◆ *Chờ đúng nội dung cần đọc, không chỉ chờ element hiện ra.*

JUNIOR

J10. Muốn chạy test trỏ từ môi trường DEV sang UAT mà không phải sửa code thì làm gì?

★ Tách cấu hình ra khỏi code.

- ► Để URL / account theo từng file config riêng cho mỗi môi trường.
- ► Chọn môi trường bằng tham số khi run: **mvn test -Denv=uat** hoặc biến môi trường.

◆ *Đổi môi trường chỉ bằng một tham số, tuyệt đối không hard-code rồi build lại.*

PHẦN B — MIDDLE TEST AUTOMATION

MIDDLE

M1. Tài khoản chỉ có 30k nhưng muốn run chuyển khoản 100 lần x 30k mà không phải nạp tiền lại mỗi lần?

★ Test tự đưa data về trạng thái cần thiết.

- ► Trong setup mỗi vòng, **reset số dư về 30k** qua API hoặc SQL update thẳng vào DB.
- ► Hoặc dùng **transaction + rollback**, hoặc restore từ DB snapshot.
- ► Nếu chỉ test luồng phía client thì có thể **mock** response của service chuyển khoản.

◆ *Không phụ thuộc thao tác tay; mỗi lần run là một trạng thái data sạch, đoán trước được.*

MIDDLE

M2. Khách hàng muốn dùng credit card để nhờ auto trên UAT nhưng không muốn lộ thông tin thì bạn làm gì?

★ Ưu tiên thẻ test, bảo mật ở tầng lưu trữ.

- ► **Không dùng số thẻ thật** — dùng **test card / sandbox card** (vd Visa test 4111 1111 1111 1111).
- ► Nếu buộc dùng data nhạy cảm: **không hard-code, không commit Git**; lưu trong secrets manager / biến môi trường.
- ► Áp dụng **masking / tokenization** và **tắt log** field nhạy cảm.

◆ *Bảo mật nằm ở chỗ lưu trữ và logging, không nằm trong source code.*

MIDDLE

M3. Muốn bắt thẻ div nhưng không có attribute thì bắt thế nào? Dùng index lỡ dev thêm vài div nữa thì sao?

★ Dùng XPath theo quan hệ thay vì attribute.

► Bám theo **text**: **//div[contains(text(),'abc')]**.

► Neo vào element ổn định gần đó rồi điều hướng: **following-sibling, parent, ancestor, child**.

► Index **(//div)[3]** rất dễ vỡ → neo vào cha/anh em có attribute ổn định rồi mới đếm index trong phạm vi nhỏ.

► Tốt nhất: đề nghị dev gắn **data-testid**.

◆ *Locator theo text / cấu trúc tương đối bền hơn khi DOM thay đổi.*

MIDDLE

M4. Khi muốn run auto mà cần tới 100 triệu user thì làm thế nào?

★ Tuyệt đối không tạo data qua UI.

► Tạo hàng loạt qua **API** hoặc **bulk insert vào DB**; sinh data giả bằng Faker / DataFaker.

► Nếu mục tiêu là **tải đồng thời** (performance) thì không tạo account thật, dùng **JMeter / k6 / Gatling** với virtual user + ramp-up, chạy distributed.

◆ *Tách rõ: "cần data tồn tại sẵn" (seed DB) khác với "cần tải đồng thời" (load test tool).*

MIDDLE

M5. Bộ regression đang chạy mất 3 tiếng, sếp muốn rút xuống còn 30 phút thì bạn làm gì?

★ Tối ưu bằng kiến trúc, không cắt bừa test.

- ► Chạy **song song (parallel)** trên Grid / Docker / cloud nhiều node.
- ► Đẩy bước **setup & verify xuống API** thay vì thao tác qua UI; loại bỏ test trùng lặp.
- ► Chia tầng: **smoke** nhanh cho mỗi build, **full regression** chạy nightly.

◆ *Rút thời gian bằng parallel + API, không phải bằng cách bỏ bớt test quan trọng.*

MIDDLE

M6. Bộ test thỉnh thoảng fail random khoảng 5% nhưng kiểm tra lại thì app không có bug — xử lý thế nào?

★ Đây là flaky test, phải soi nguyên nhân gốc.

- ► Nguyên nhân hay gặp: **thiếu wait đúng**, **data phụ thuộc**, **thứ tự chạy**, **môi trường không ổn**.
- ► Làm test **độc lập**, thay sleep bằng explicit wait, ổn định lại locator.
- ► Bật **retry** chỉ để giảm nhiễu báo cáo tạm thời, không che giấu lỗi gốc.

◆ *Flaky test làm mất niềm tin vào cả bộ automation — phải xử lý dứt điểm.*

MIDDLE

M7. Màn login yêu cầu OTP gửi qua SMS / email mới đăng nhập được thì automation chạy thế nào?

★ Lấy OTP tự động, không nhập tay.

- ► Lấy OTP qua **API / đọc DB / mailbox API** thay vì chờ điện thoại.
- ► Hoặc môi trường test cấu hình **OTP cố định** / tắt OTP.
- ► Hoặc **mock** service gửi OTP.

◆ *Phối hợp với dev để có cách lấy OTP tự động trên môi trường test.*

MIDDLE

M8. App có captcha ở màn login chặn automation thì làm sao chạy được auto?

★ Không phá captcha — xử lý bằng cấu hình môi trường test.

- ► **KHÔNG bypass / giải captcha** vì như vậy là phá vỡ mục đích bảo mật.
- ► Trên môi trường test: đề nghị dev **tắt / whitelist captcha**, hoặc dùng **reCAPTCHA test key**.
- ► Hoặc **login qua API** để bỏ qua màn UI có captcha.

◆ *Đây là câu trả lời đúng chuẩn: không vượt captcha, mà cấu hình môi trường test cho hợp lệ.*

MIDDLE

M9. Chạy song song mà test khác sửa data làm case của bạn fail thì xử lý thế nào?

★ Test song song chỉ an toàn khi data độc lập.

- ► Mỗi test **tự tạo data riêng**, không dùng chung tài khoản / record với test khác.
- ► Gắn data theo **thread / run id** để các luồng không đụng nhau.
- ► Tránh dùng biến / state static chia sẻ giữa các test.

◆ *Cô lập data là điều kiện bắt buộc để parallel không gây kết quả sai ngẫu nhiên.*

MIDDLE

M10. Cần verify một giá trị nằm trong DB sau khi thao tác trên UI thì kiểm tra thế nào?

★ Verify ở nguồn dữ liệu, không chỉ tin UI.

- ► Kết nối DB **query trực tiếp** (JDBC) rồi so với giá trị mong đợi.
- ► Hoặc verify qua **API** nếu có endpoint tương ứng.
- ► UI có thể hiển thị đúng nhưng lưu sai (hoặc ngược lại) → kiểm tra tận backend cho chắc.

◆ *Kiểm tra ở DB / API cho kết quả tin cậy hơn chỉ nhìn màn hình.*