❓ 2. How do you know a failure is real bug or automation issue?

☑ Smart Answer:

Steps:

1. Check logs
2. Check screenshot
3. Re-run locally
4. Check API response
5. Validate test data

If it fails consistently → real bug
If random → synchronization issue

❓ How do you ensure your automation code is correct?

From code quality practices to

☑ 1 Code Quality Practices

✔ Follow Clean Code Principles

- Meaningful variable names
- No hardcoded values
- Proper folder structure
- Reusable methods

Bad:

```
await $('#btn1').click();
```

Good:

```
await loginPage.loginButton.click();
```

☑ 2 Proper Wait Strategy (Very Important)

Most automation failures come from bad waits.

Always:

```
await element.waitForClickable({ timeout: 5000 });
await element.click();
```

Never depend on:

```
browser.pause(3000);
```

☑ 3 Assertions Everywhere

Without assertion → test is useless.

Example:

```
const title = await browser.getTitle();
expect(title).toBe('Dashboard');
```

Good test must:

- Validate state
- Validate data
- Validate UI

☑ 4 Error Handling

Use try-catch only where needed:

```
try {
   await element.click();
} catch (error) {
   console.error("Click failed:", error);
   await browser.saveScreenshot('./error.png');
}
```

But don’t hide errors unnecessarily.

☑ 5 Reusable Utilities

Instead of repeating:

```
await element.waitForDisplayed();
await element.click();
```

Create utility:

```
async function safeClick(element) {
   await element.waitForClickable();
   await element.scrollIntoView();
   await element.click();
}
```

☑ 6 Proper Logging

Logs help in CI debugging.

Example in wdio.conf.js:

```
logLevel: 'info'
```

Add logs inside tests:

```
console.log("User logged in successfully");
```

☑ 7 Screenshot on Failure

In Cucumber hooks:

```
After(async function (scenario) {
   if (scenario.result.status === 'FAILED') {
      await browser.saveScreenshot(`./error.png`);
   }
});
```

This ensures traceability.

☑ 8 Code Review

Before pushing:

- Check unused imports
- Check proper awaits
- Run locally
- Linting check
- Remove console logs (if not needed)

☑ 9 Proper test data

Don’t use shared test data.

Bad:
Same username in all tests.

Good:
Generate unique user via API.

☑ 🔟 CI Validation

Run your code in:

- Headless mode
- Different browser
- Parallel execution

If it passes everywhere → stable.

❓ 1. What is the biggest mistake automation engineers make?

☑ Strong Answer:

- Over-automation
- Automating unstable features
- Using hard waits
- Not maintaining framework
- Writing scripts instead of designing automation

❓ 2. Can 100% automation be achieved?

☑ Smart Answer:

No.

Because:

- Exploratory testing
- Usability testing
- Visual validation
- Frequently changing features

Automation should target high-value, repetitive tests.

❓ 3. What is flaky test? How do you reduce it?

☑ Answer:

A flaky test is one that passes sometimes and fails sometimes without code change.

Causes:

- Poor wait strategy
- Test data conflict
- Environment instability
- Parallel execution conflicts

Fix:

- Explicit waits
- Stable locators
- Isolated test data
- Retry logic carefully

❓ 4. Why is `browser.pause()` bad?

☑ Answer:

- Slows execution
- Makes test unreliable
- Not environment-aware
- Fails if system slower

Use `waitForDisplayed()` instead.

❓ 5. Why does your script work locally but fail in CI?

Possible Reasons:

- Different browser version
- Headless execution
- Slow network
- OS differences
- Missing waits

❓ 6. What is difference between implicit and explicit wait?

Implicit Wait	Explicit Wait
Global	Condition-based
Applies to all elements	Specific element
Not recommended in modern frameworks	Recommended

WebdriverIO mainly uses explicit waits.

❓ 7. What happens if DOM refreshes after locating element?

Answer:
You get **StaleElementReferenceException**.
Solution: Re-locate element.

❓ 8. Why do we need Page Object Model?

Answer:

- Separate test logic from locators
- Reusability
- Maintainability
- Clean code structure

❓ 9. What is hybrid framework?

Combination of:

- POM
- Data-driven
- Keyword-driven
- BDD

❓ 10. How do you design a scalable automation framework?

Answer:

- Modular structure
- Reusable utilities
- Config-based execution
- Environment support
- Logging & reporting
- CI/CD integration

❓ 11. How do you manage test data?

Options:

- JSON
- Excel
- DB
- API-generated data
- Environment variables

Best practice: Generate data via API.

❓ 12. Why validate API if UI already tested?

Answer:
UI shows formatted data.
API gives raw data.
Backend validation is more reliable.

❓ 13. Login API works but UI login fails. Why?

Possible:

- Token not stored
- Cookie issue
- UI validation error
- Frontend bug

❓ 14. How do you handle authentication token expiry?

Answer:

- Refresh token mechanism
- Generate token before suite
- Store token globally

❓ 15. How do you debug “element not clickable at point”?

Answer:

- Check overlay
- Scroll into view
- Wait for clickable
- Inspect z-index
- Use JS click as last option

❓ 16. How do you reduce 5-hour regression suite?

Answer:

- Parallel execution
- Smoke subset
- Remove redundant tests
- API for setup
- Optimize waits

❓ 17. What is your strategy if build fails at midnight?

Answer:

- Check CI logs
- Identify flaky test
- Rerun failed tests
- Analyze screenshots

- Inform team if real defect

❓ 18. What is difference between SDET and QA automation engineer?

SDET:

- Writes framework
- Designs architecture
- Strong coding
- Involved in CI/CD

Automation QA:

- Writes test scripts
- Executes tests

❓ 19. How do you measure automation success?

Metrics:

- Defect detection rate
- Execution time reduction
- Flaky rate
- Maintenance cost
- Coverage %

❓ 20. When should automation be stopped?

Answer:
If:

- Maintenance cost > manual cost
- Feature unstable
- Low ROI

❓ 21. API returns 200 but data incorrect. Is it success?

Answer:
No.

Status code validation alone is not enough.
Always validate response body.

❓ 22. How do you handle microservices testing?

Answer:

- Service-level API tests
- Contract testing
- Integration tests
- End-to-end UI tests

❓ 23. Why automation sometimes increases cost?

Answer:

- Poor framework
- High maintenance
- Flaky tests
- Bad design

❓ 24. What is over-engineering in automation?

Answer:
Adding unnecessary abstraction layers for small projects.

❓ 25. Why do companies reject automation candidates?

Common reasons:

- Only tool knowledge
- No debugging skills
- No framework knowledge

- Weak coding fundamentals
- Cannot explain architecture

❓ What is an Automation Framework?

☑ Smart Answer:

An automation framework is a structured set of guidelines, reusable components, utilities, reporting, and configurations that help in building scalable, maintainable, and reusable automation tests.

It is not just test scripts — it includes:

- Folder structure
- Page Object Model
- Reusable utilities
- Logging
- Reporting
- CI integration
- Retry logic

❓ Why do we need a framework?

☑ Answer:

Without framework:

- Code duplication
- Hard to maintain
- No proper reporting
- No scalability
- No reusability

Framework gives:

- Reusability
- Maintainability
- Scalability
- Clean structure
- Team collaboration

❓ What type of framework are you using?

You can say:

I am using Hybrid Framework combining Page Object Model + Cucumber BDD + Utility-based reusable components.

❓ What are key components in your framework?

You can answer based on your setup:

- `pages/` → Page Objects
- `stepdefinitions/` → Cucumber steps
- `hooks/` → Before/After hooks
- `utils/` → Reusable functions
- `config/` → WDIO configuration
- `reports/` → Allure/HTML reports

❓ How do you handle failures in your framework?

Smart answer:

- Screenshot on failure
- Logs using logger
- Retry mechanism
- Capture failed test file
- Rerun failed cases

❓ Why Git is important in automation?

☑ Answer:

- Version control
- Collaboration
- Track changes
- Code rollback
- CI/CD integration

❓ Difference between git fetch and git pull?

git fetch	git pull
Downloads changes	Downloads + merges
Safe	Directly affects branch

❓ What is branching strategy in automation?

You can say:

- main → production-ready
- develop → integration
- feature branches → new automation scripts
- hotfix branch → urgent fix

❓ What is rebase?

Rebase rewrites commit history to maintain clean commit structure.

❓ How do you resolve merge conflicts?

Steps:

1. Identify conflict file
2. Fix manually
3. git add
4. git commit

🔥 3 Smoke vs Regression (Automation View)

❓ What is Smoke Testing?

☑ Answer:

Smoke testing validates critical functionalities to ensure build is stable.

Example:

- Login
- Dashboard load
- Logout

Runs:

- After every build
- Short duration
- High priority test cases

❓ What is Regression Testing?

☑ Answer:

Regression testing ensures that new changes didn’t break existing features.

- Covers full application
- Large test suite
- Runs before release

❓ Which tests should be automated — smoke or regression?

Best answer:

Smoke tests must be automated.
Regression tests are highly recommended to automate due to repetitive execution.

🔥 4️⃣ When Do We Do Automation?

❓ When should we automate?

☑ Good Scenarios:

- Stable application
- Repetitive test cases
- Regression testing
- High risk areas
- Large data testing

❌ When NOT to automate?

- Frequently changing UI
- One-time test cases
- Exploratory testing
- Small project

🔥 5️⃣ Why Should We Do Automation?

☑ Business Answer:

- Saves time
- Faster feedback
- Reduces human error
- Improves coverage
- CI/CD integration
- Cost effective long term

🎯 Strong Interview Statement:

Automation increases release confidence and reduces manual effort in regression cycles.

❓ How do you make your framework scalable?

Answer:

- Modular design
- Reusable utilities
- Centralized locators

- Config-based execution
- Environment handling

? How do you handle test data?

Options:

- JSON files
- Excel
- Environment variables
- Database
- API-driven test data

? How do you run specific tags in Cucumber?

```
npx wdio run wdio.conf.js --cucumberOpts.tagExpression="@smoke"
```

? How do you run tests in parallel?

In wdio.conf.js:

```
maxInstances: 5
```

? How do you integrate automation in CI/CD?

Answer:

- Push code to Git
- Pipeline triggers
- Install dependencies
- Run tests
- Generate report
- Publish results

? What if your regression suite takes 6 hours?

Answer:

- Parallel execution
- Smoke subset for quick validation
- Optimize waits
- Remove redundant test cases

❓ How do you decide which test cases to automate?

Answer:

- High priority
- Stable functionality
- Repetitive
- Business critical
- Data driven

❓ How do you measure automation success?

Metrics:

- Execution time reduction
- Defect detection rate
- Coverage %
- Flaky test rate
- Maintenance effort

❓ Why should we hire you as automation engineer?

Strong answer:

I don’t just write scripts — I design scalable frameworks, handle CI/CD integration, manage version control, debug flaky tests, and ensure automation adds real business value.

❓ 1. Why combine API and UI automation?

☑ Smart Answer:

UI tests are slower and flaky.
API tests are fast and reliable.
We combine both to:

- Prepare test data via API

- Validate backend response
- Reduce UI execution time
- Improve reliability

❓ 2. How do you create test data before UI test?

☑ Example (WebdriverIO + API request)

```
const response = await fetch('https://api.example.com/createUser', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        username: 'testuser',
        password: 'password'
    })
});

const data = await response.json();
console.log(data.userId);
```

Then use that user in UI login.

❓ 3. How do you validate UI data with API response?

Scenario:

1. Create order via API
2. Login UI
3. Verify order appears

Example:

```
const apiOrder = await getOrderFromAPI(orderId);
await $('#searchOrder').setValue(orderId);
await $('#searchBtn').click();

const uiOrder = await $('#orderAmount').getText();
expect(uiOrder).toBe(apiOrder.amount);
```

❓ 4. How do you reduce UI test flakiness using API?

Answer:

- Create test data via API
- Delete data after test

- Validate backend via API instead of UI text
- Avoid heavy UI flows

❓ 5. How do you validate DB using API + UI?

Answer:

- Trigger UI action
- Call API
- Validate DB response

❓ 6. Why is your UI test flaky?

Possible Reasons:

- Poor wait strategy
- Dynamic elements
- Hardcoded pause
- Environment issue
- Test data conflict

❓ 7. How do you debug a failing test in CI?

Answer:

1. Check logs
2. Check screenshots
3. Check video recording
4. Run locally in headless mode
5. Increase log level

```
logLevel: 'info'
```

❓ 8. What happens internally when you click an element?

Smart answer:

Webdriver sends HTTP request to browser driver,
driver interacts with browser DOM,
browser executes click event.

❓ 9. How do you handle token-based authentication in API automation?

```
const login = await fetch('/login', {
    method: 'POST',
    body: JSON.stringify({ username, password })
});

const token = (await login.json()).token;

await fetch('/orders', {
    headers: {
        Authorization: `Bearer ${token}`
    }
});
```

❓ 10. What is difference between UI validation and API validation?

UI Validation	API Validation
Slow	Fast
Visual	Data-level
Prone to flakiness	Stable

❓ 11. Login works via API but fails in UI. Why?

Possible answers:

- Frontend bug
- CORS issue
- Token not stored
- Session cookie issue
- UI validation issue

❓ 12. Order created via UI but not visible in DB. Why?

Possible:

- Async processing delay
- Cache issue
- DB replication lag
- Environment mismatch

❓ 13. API returns 200 but UI shows error.

Why?

- Backend response format changed
- UI parsing issue
- Field mapping issue

❓ 14. Where do you keep API utilities in framework?

Answer:

`/services` folder

Example:

```
class UserService {
   async createUser(data) {
      return await fetch('/users', {
         method: 'POST',
         body: JSON.stringify(data)
      });
   }
}
```

❓ 15. How do you separate API and UI tests?

Answer:

- Use tags

```
@api
@ui
@regression
```

Run separately:

```
--cucumberOpts.tagExpression="@api"
```

❓ 16. How do you handle environment configs?

Answer:

Use environment variables:

```
baseUrl: process.env.BASE_URL
```

❓ 17. How do you validate performance via automation?

Answer:

- Measure API response time
- Validate response under 2 sec

```
const start = Date.now();
await fetch('/orders');
const time = Date.now() - start;
console.log(time);
```

❓ 18. If UI and API validations conflict, which one is correct?

Answer:

API is source of truth.
UI depends on API data.

❓ 19. How do you design automation for microservices architecture?

Answer:

- Test services independently
- Contract testing
- API-level validations
- End-to-end integration tests

❓ 20. How do you reduce execution time of 1000 test cases?

Answer:

- Parallel execution
- Tag-based filtering
- Remove redundant tests
- Data-driven approach
- API for setup

❓ Explain complete login automation flow (API + UI + Framework + CI)

Strong Answer:

1. Create user via API
2. Login via UI
3. Validate dashboard
4. Validate token via API
5. Capture screenshot on failure
6. Push results to CI pipeline
7. Generate report

1. Browser Commands

These commands control the browser itself.

☑ Q1: How do you launch a URL?

```
await browser.url('https://example.com');
```

Explanation:

- `browser.url()` → Opens the specified URL.
- Always use `await` (because WebdriverIO is async).

☑ Q2: How do you get the page title?

```
const title = await browser.getTitle();
console.log(title);
```

☑ Q3: How do you get the current URL?

```
const currentUrl = await browser.getUrl();
```

☑ Q4: How to maximize the browser window?

```
await browser.maximizeWindow();
```

☑ Q5: How to refresh the page?

```
await browser.refresh();
```

☑ Q6: How to go back and forward?

```
await browser.back();
await browser.forward();
```

☑ Q7: How to close browser?

```
await browser.closeWindow();   // closes current tab
await browser.deleteSession(); // ends entire session
```

🔹 2. Navigation Commands

☑ Q1: What is navigation in automation?

Navigation means moving between pages using browser controls.

☑ Example:

```
await browser.url('https://google.com');
await browser.url('https://github.com');
await browser.back();
await browser.forward();
```

🔹 Interview Question:

Difference between `browser.url()` and `browser.navigateTo()`?

👉 In WebdriverIO both work similarly.
`navigateTo()` is an alias of `url()`.

🔹 3. Scroll Commands

Scrolling is required when elements are not visible on screen.

☑ Q1: How do you scroll to an element?

```
const element = await $('#footer');
await element.scrollIntoView();
```

☑ Q2: Scroll by pixel?

```
await browser.execute(() => {
    window.scrollBy(0, 500);
});
```

☑ Q3: Scroll to bottom of page?

```
await browser.execute(() => {
    window.scrollTo(0, document.body.scrollHeight);
});
```

🔹 Interview Question:

Why do we need scroll in automation?

👉 Because Selenium/WebdriverIO can interact only with visible elements.

🔹 4. Click Commands

☑ Q1: How do you click an element?

```
await $('#loginBtn').click();
```

☑ Q2: How do you handle element not clickable?

```
const btn = await $('#loginBtn');
await btn.waitForClickable({ timeout: 5000 });
await btn.click();
```

☑ Q3: How to use JavaScript click?

```
const btn = await $('#loginBtn');
await browser.execute((el) => el.click(), btn);
```

5. Handling Frames (iFrames)

An iframe is a webpage inside another webpage.

🖼️ Example of iframe HTML

```
<iframe id="paymentFrame"></iframe>
```

☑ Q1: How to switch to frame?

```
const frame = await $('#paymentFrame');
await browser.switchToFrame(frame);
```

☑ Q2: Switch back to main page?

```
await browser.switchToParentFrame();
```

🔹 Interview Question:

What happens if you don't switch to iframe?

👉 You get "no such element" error.

🔹 6. Window Handles (Multiple Tabs)

Used when clicking opens new tab.

☑ Q1: How to get window handles?

```
const handles = await browser.getWindowHandles();
console.log(handles);
```

☑ Q2: Switch to new window?

```
const handles = await browser.getWindowHandles();
await browser.switchToWindow(handles[1]);
```

☑ Q3: Get current window handle?

```
const current = await browser.getWindowHandle();
```

🔹 Interview Question:

What is difference between window handle and window handles?

- `getWindowHandle()` → Current window
- `getWindowHandles()` → All open windows

🔹 7. Shadow DOM (Shadow Roots)

Shadow DOM is used in modern web applications.

Normal locator won’t work.

🖼️ Example HTML

```
<custom-element>
   #shadow-root
      <button id="submitBtn"></button>
</custom-element>
```

☑ Q1: How to handle shadow DOM?

```
const shadowHost = await $('custom-element');
const shadowRoot = await shadowHost.shadow$('button#submitBtn');
await shadowRoot.click();
```

☑ Nested Shadow DOM Example

```
const parent = await $('parent-element');
const child = await parent.shadow$('child-element');
const button = await child.shadow$('button');
await button.click();
```

🔹 Interview Question:

Why normal XPath doesn’t work in Shadow DOM?

👉 Because shadow DOM is isolated from main DOM.

🔥 Real Interview Rapid Fire Questions

1️⃣ What is Explicit Wait?

```
await element.waitForDisplayed({ timeout: 5000 });
```

Waits until condition is true.

2️⃣ What is difference between waitForDisplayed and waitForExist?

waitForExist	waitForDisplayed
Element in DOM	Element visible

3️⃣ What is Stale Element?

Element removed or refreshed from DOM.

4️⃣ How to handle dynamic elements?

Use:

```
$('button*=Login')
```

Partial matching.

🔥 Real-Time Scenario Example

```
it('Complete Login Flow', async () => {

    await browser.url('https://example.com');

    await $('#username').setValue('testuser');
    await $('#password').setValue('password');

    await $('#loginBtn').waitForClickable();
    await $('#loginBtn').click();

    await $('#dashboard').waitForDisplayed();

    const title = await browser.getTitle();
    console.log(title);

});
```

1️⃣ Difference between `browser.execute()` and `browser.executeAsync()`

☑ `browser.execute()`

- Runs JavaScript inside the browser
- Returns result immediately
- Used for synchronous JS

🔹 Syntax

```
const title = await browser.execute(() => {
    return document.title;
});
```

☑ `browser.executeAsync()`

- Used when JavaScript is asynchronous
- Requires a callback to signal completion
- Useful for API calls, timers, promises

🔹 Syntax

```
const result = await browser.executeAsync((done) => {
    setTimeout(() => {
        done("Finished");
    }, 2000);
});
```

🎯 Interview Difference

execute()	executeAsync()
Synchronous	Asynchronous
Returns immediately	Waits for callback
No `done()` needed	Must call `done()`

👉 If you forget `done()`, test will hang.

🔥 2 What is Shadow DOM?

Shadow DOM is a **hidden DOM tree** attached to an element.
Normal locators (XPath/CSS) won't work directly.

It is used in modern apps (Angular, Salesforce, Polymer).

Example Structure

```
<custom-element>
   #shadow-root (open)
       <button id="submit">
</custom-element>
```

☑ Handling in WebdriverIO

```
const shadowHost = await $('custom-element');
const shadowButton = await shadowHost.shadow$('#submit');
await shadowButton.click();
```

3 How to Handle Multiple Windows?

When clicking link opens new tab.

☑ Steps

1. Get all window handles
2. Switch to required window

🔹 Syntax

```
const handles = await browser.getWindowHandles();
await browser.switchToWindow(handles[1]);
```

Switch back

```
await browser.switchToWindow(handles[0]);
```

🎯 Interview Tip

`getWindowHandle()` → Current
`getWindowHandles()` → All windows

🔥 4️⃣ How to Handle iFrames inside iFrame?

Nested frames require switching step by step.

Example

```
Main Page
   ↳ Frame1
       ↳ Frame2
```

☑️ Syntax

```
const frame1 = await $('#frame1');
await browser.switchToFrame(frame1);

const frame2 = await $('#frame2');
await browser.switchToFrame(frame2);

// Perform action
await $('#insideElement').click();
```

Switch back

```
await browser.switchToParentFrame();
await browser.switchToParentFrame();
```

🎯 Interview Answer

We must switch to parent frame first before switching to another frame.

🔥 5️⃣ How to Scroll Inside Specific Element?

Sometimes scrolling page is not enough.

☑ Using scrollIntoView()

```
const element = await $('#target');
await element.scrollIntoView();
```

☑ Scroll inside specific container

```
const container = await $('#scrollDiv');

await browser.execute((el) => {
    el.scrollTop = el.scrollHeight;
}, container);
```

🎯 Interview Answer

We use JavaScript execution to scroll inside specific div container.

🔥 6️⃣ How to Debug Click Interception Error?

❌ Error:

ElementClickInterceptedError

Means:

- Element overlapped
- Popup covering it
- Not visible
- Not clickable yet

☑ Debug Steps

1️⃣ Wait for clickable

```
await element.waitForClickable({ timeout: 5000 });
await element.click();
```

2️⃣ Scroll into view

```
await element.scrollIntoView();
await element.click();
```

3️⃣ Use JS Click (Last Option)

```
await browser.execute(el => el.click(), element);
```

4️⃣ Check overlapping element

Use DevTools → Inspect → Check z-index

🎯 Interview Answer

First I wait for clickable, then scroll, and finally use JS click as fallback.

🔥 7️⃣ How to Handle Dynamic IDs?

Example:

```
id="login_12345"
id="login_67890"
```

☑ Use Partial Match

```
$('button[id^="login_"]')   // starts with
$('button[id*="login"]')    // contains
```

☑ Using XPath contains

```
$('//button[contains(@id,"login")]')
```

🎯 Interview Answer

I avoid exact ID and use starts-with or contains strategy.

🔥 8️⃣ What is Page Object Model (POM)?

POM is design pattern where:

- Page locators
- Page methods
- Test logic

are separated.

🎯 Interview Answer

POM improves reusability, maintainability, and reduces code duplication.

🔥 9️⃣ How to Capture Screenshot?

☑️ Capture full page

```
await browser.saveScreenshot('./screenshots/error.png');
```

☑️ Capture element screenshot

```
const element = await $('#logo');
await element.saveScreenshot('./logo.png');
```

What happens if you don’t use `await` in WebdriverIO?

☑️ Smart Answer:

WebdriverIO commands are asynchronous.
If we don’t use `await`, execution continues before the command finishes, causing flaky tests.

❌ Wrong Code

```
$('#loginBtn').click();
browser.getTitle();
```

☑ Correct Code

```
await $('#loginBtn').click();
const title = await browser.getTitle();
```

🎯 Why interviewer asks?

To check if you understand async behavior.

🔥 2 Why is your test passing locally but failing in CI?

☑ Smart Answer:

Possible reasons:

- Timing issue (missing wait)
- Different browser versions
- Headless mode behavior
- Network latency
- Environment differences

☑ Fix Example

```
await element.waitForDisplayed({ timeout: 5000 });
```

🎯 They check debugging ability.

🔥 3 Difference between `waitForExist()` and `waitForDisplayed()`?

waitForExist	waitForDisplayed
Element in DOM	Element visible
May be hidden	Must be visible

Example

```
await element.waitForExist();
await element.waitForDisplayed();
```

👉 Hidden elements pass `waitForExist()`.

🔥 4️⃣ Why does click sometimes fail even if element is visible?

☑ Reasons:

- Overlay present
- Animation not finished
- Element disabled
- z-index issue
- Not clickable yet

☑ Smart Fix Order:

1. waitForClickable()
2. scrollIntoView()
3. Pause (last option)
4. JS click (last fallback)

🔥 5️⃣ How do you handle stale element reference?

☑ What is it?

Element refreshed or re-rendered.

☑ Solution:

Re-locate element again.

```
await $('#submit').click();
const button = await $('#submit');
await button.click();
```

🎯 Trick:

Never store element for long time in dynamic apps.

🔥 6 How do you handle dynamic dropdown loaded after API call?

☑ Smart Answer:

Wait until options count increases.

```
await browser.waitUntil(async () => {
    const options = await $$('#dropdown option');
    return options.length > 1;
});
```

🔥 7 Why should you avoid `browser.pause()`?

☑ Smart Answer:

Pause makes test slow and flaky.
We should use smart waits like waitForDisplayed or waitUntil.

🔥 8 How do you verify element is NOT present?

❌ Wrong Way:

```
await element.waitForExist({ reverse: true });
```

☑ Correct Way:

```
const isExisting = await element.isExisting();
expect(isExisting).toBe(false);
```

🔥 9️⃣ What is difference between `$` and `$$`?

`$`	`$$`
Returns single element	Returns array
Used for unique locator	Used for multiple elements

```
const button = await $('#login');
const buttons = await $$('button');
```

🔥 🔟 Why sometimes `getText()` returns empty string?

☑️ Reasons:

- Element hidden
- Text inside child element
- Using wrong locator

☑️ Alternative:

```
await element.getHTML();
```

🔥 1️⃣1️⃣ How to handle authentication popup?

☑️ Smart Answer:

Use URL format:

```
await browser.url('https://username:password@example.com');
```

🔥 1️⃣2️⃣ How to handle file upload?

☑️ Trick Question

```
const filePath = path.join(__dirname, '../file.txt');
const remotePath = await browser.uploadFile(filePath);
await $('#fileUpload').setValue(remotePath);
```

🔥 13 How do you handle retry for flaky test?

☑ In WDIO config:

```
mochaOpts: {
   retries: 2
}
```

Or Cucumber:

```
retry: 2
```

🔥 14 What is difference between `isDisplayed()` and `waitForDisplayed()`?

isDisplayed	waitForDisplayed
Immediate check	Waits until condition
No retry	Retries until timeout

🔥 15 How do you debug element not found?

Steps:

1. Check locator in DevTools
2. Confirm iframe
3. Confirm shadow DOM
4. Add wait
5. Print page source

```
console.log(await browser.getPageSource());
```

🔥 16 What is difference between hard assertion and soft assertion?

Hard:

```
expect(title).toBe("Home");
```

Soft:
Continue execution even if fails.

(WebdriverIO mostly uses hard assertion unless custom soft logic added.)

🔥 17 How do you handle infinite scroll?

```
await browser.execute(() => {
    window.scrollTo(0, document.body.scrollHeight);
});
```

Loop until no new content.

🔥 18 Why is your test flaky?

Smart Answer:

Flaky tests are caused by:

- Poor wait strategy
- Dynamic elements
- Timing issues
- External dependency

🔥 19 What is best locator strategy?

Priority Order:

1. ID
2. Data-testid

3. CSS
4. XPath (last option)

🔥 2️⃣0️⃣ How to handle element inside Shadow DOM inside iframe?

Advanced Answer:

1. Switch to iframe
2. Access shadow root

```
await browser.switchToFrame(await $('#frame'));

const shadowHost = await $('custom-element');
const shadowBtn = await shadowHost.shadow$('#btn');

await shadowBtn.click();
```

🚀 SUPER TRICKY QUESTION

❓ Why does this fail?

```
const btn = $('#login');
await btn.click();
```

❗ Answer:

Because element wasn’t awaited during selection.

Correct:

```
const btn = await $('#login');
await btn.click();
```

Common WebdriverIO / Selenium Exceptions

1️⃣ NoSuchElementError

☑ When do we get it?

- Locator is wrong
- Element not in DOM
- Page not loaded
- Forgot iframe switch

❌ Example

```
await $('#wrongId').click();
```

☑ Fix

```
const element = await $('#correctId');
await element.waitForExist({ timeout: 5000 });
await element.click();
```

---

2 ElementNotInteractableError

☑ When?

- Element hidden
- Element disabled
- CSS display: none
- Covered by another element

❌ Example

```
await $('#hiddenButton').click();
```

☑ Fix

```
await element.waitForDisplayed();
await element.scrollIntoView();
await element.click();
```

---

3 ElementClickInterceptedError

☑ When?

- Popup covering element
- Overlay present
- Animation running

☑ Fix Strategy

```
await element.waitForClickable();
await element.scrollIntoView();
await element.click();
```

If still failing:

```
await browser.execute(el => el.click(), element);
```

---

4 StaleElementReferenceError

☑ When?

- Page refreshed
- DOM re-rendered (React/Angular)
- Stored element for long time

❌ Example

```
const btn = await $('#submit');
await browser.refresh();
await btn.click();   // ❌ stale
```

☑ Fix

Re-locate element:

```
await browser.refresh();
const btn = await $('#submit');
await btn.click();
```

---

5 TimeoutError

☑ When?

- Wait exceeded timeout
- Condition never met

Example:

```
await element.waitForDisplayed({ timeout: 2000 });
```

If element loads in 3 sec → timeout error.

🔥 2 Explicit Waits in WebdriverIO

WebdriverIO does NOT use implicit wait like Selenium.

We use explicit waits.

1️⃣ waitForExist()

Element must exist in DOM.

```
await element.waitForExist({ timeout: 5000 });
```

2️⃣ waitForDisplayed()

Element must be visible.

```
await element.waitForDisplayed({ timeout: 5000 });
```

3️⃣ waitForClickable()

Element must be:

- Visible
- Enabled
- Not covered

```
await element.waitForClickable({ timeout: 5000 });
```

4️⃣ waitUntil()

Custom condition.

```
await browser.waitUntil(async () => {
    return (await $('#success').isDisplayed());
}, {
    timeout: 5000,
    timeoutMsg: 'Success message not displayed'
});
```

🔥 3️⃣ isDisplayed(), isExisting(), isClickable()

These are validation methods — NOT waits.

---

1 isDisplayed()

☑ When to use?

To verify element visible.

```
const status = await element.isDisplayed();
expect(status).toBe(true);
```

⚠ When does it fail?

If element not in DOM → throws error.

---

2 isExisting()

Checks only DOM presence.

```
const exists = await element.isExisting();
```

Use when:

You want to check element NOT present.

---

3 isClickable()

Checks:

- Visible
- Enabled
- No overlay

```
const clickable = await element.isClickable();
```

⚠ It does NOT wait automatically.

🔥 4 isCheckable / Checkbox Handling

WebdriverIO does not have direct `isCheckable()` method.
Instead, we use:

- isSelected()
- getAttribute("checked")

Example Checkbox HTML

```
<input type="checkbox" id="agree" />
```

☑ Check if selected

```
const checkbox = await $('#agree');
const isChecked = await checkbox.isSelected();
```

☑ Select checkbox

```
if (!(await checkbox.isSelected())) {
    await checkbox.click();
}
```

When do we get errors?

- Clicking disabled checkbox → ElementNotInteractable
- Checkbox inside iframe → NoSuchElement

🔥 5️⃣ verifyIsDisplayed – When do we use it?

Usually used in assertion.

Scenario:

After login, verify dashboard visible.

```
await $('#dashboard').waitForDisplayed();
expect(await $('#dashboard').isDisplayed()).toBe(true);
```

When do we get failure?

1. Element exists but hidden
2. Element inside iframe
3. Page not loaded
4. Wrong locator