# Dealls Jobs - QA Software Engineer / SDET Challenge

Al Khawarismi Atma Pratama | [alkhawarismi989@gmail.com](mailto:alkhawarismi989@gmail.com) | +6282182790655

## Challenge 1: Automation Testing (Technical Skills)

1. **Test Strategy**

Understanding a website's feature or product can be challenging without access to the business requirements, acceptance criteria, user scenarios, or predefined test cases.

Here are my test strategy and assumptions to break it down into clear and manageable steps:

* **The type of testing** that I choose would be “**Exploratory Testing**” because I was not given any hint and have minimum knowledge about this feature or product.
* **The key areas of focus** right now would be **Core Business Flows** such as **High Impact, High Frequency** flow.
* **The environments** should be **development/staging** because the link that given was Google Cloud Development link (--a.run.app).
* **Gather basic information** of the feature such as what does it do, explore the flow, and make a test scope and test plan.
* **Exploring** the website through **exploratory testin**g to define and take note about the features, flows, and even collecting test data for future automation test.
* **Exploring the DOM elements** of the feature, documenting the locators that I need for automation test.
* **Setting up** automation environment and begin developing automation.
* **Develop** automation (create, review, execute, integrate, and maintain).
* I assume this automation test would collaborate with many people and must scalable, so I need to integrate to CI/CD (GitHub Actions) for continuous validation.

1. **High-Level Test Scenarios**

Here is the list of high-level test scenarios that I found on the product:

* Register.
* Login.
* Search for a mentor as mentee.
* Request schedule to a mentor as mentee.
* Make a schedule as mentor.
* Confirm mentee attendance when the session is finished as mentor.
* Give comments or note as mentor.
* Give comments or note as mentee.
* Give review to mentor as mentee.

## Challenge 2: UI Automation – Web Scenarios

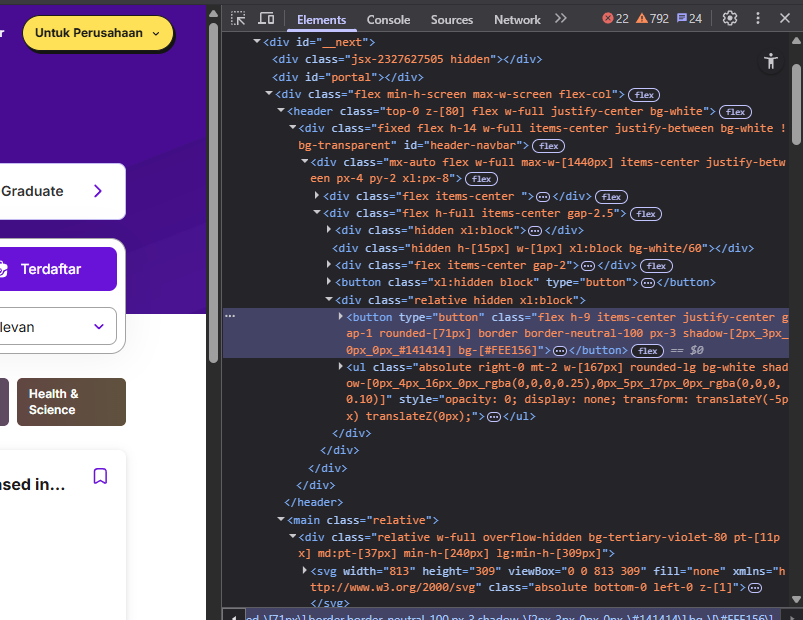
I chose **Cypress** for this task. The full automation scripts are structured using **describe/it blocks** with POM-DDT and are available in the **GitHub repository.** Here is the list of scenarios that I automate:

* Register.
* Login.
* Search for a mentor as mentee.
* Requesting Schedule to a mentor as a mentee.

## Challenge 3: Exploration & Bug

Here is the list of failure points that I found in Dealls.com:

1. Many elements do not have id or data-testid.



* **Investigation Steps**:

1. Inspected various UI components rendered in the application.
2. Identified key interactive elements (e.g., inputs, buttons, dropdowns) lacking unique id or data-testid attributes
3. Attempted to locate these elements using standard automation tools (e.g., Cypress).
4. Found that selectors had to rely on unstable properties, such as class names, DOM structure, or inner text
5. Verified that these selectors often broke when the UI was updated or refactored, indicating fragility.

* **Information Gathering:**

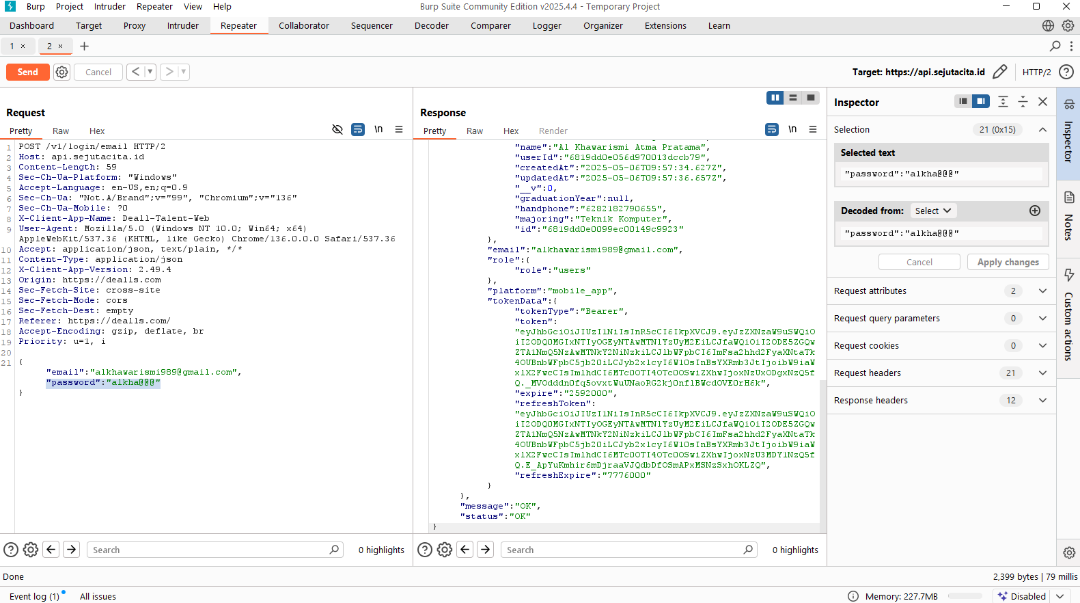
1. Framework Used**:** Ant Design (AntD).
2. Issue Observed: Multiple interactive elements (e.g., dropdowns, input fields, modals) do not have id or data-testid attributes.
3. Selector Options: Only class names or complex XPath selectors are available.
4. Test Stability Impact: Automation scripts fail intermittently due to DOM changes.
5. Accessibility Impact: Missing id may also affect assistive technologies.

* **Potential RC:**

1. Library defaults.
2. Developers not aware of importance of this.
3. No guidelines.

* **Reproduction Strategy:**

1. Open the application in the browser
2. Right-click and inspect any input field, dropdown, or button
3. Observe that there is no id attribute and there is no data-testid, data-cy, or similar test hook.
4. Attempt to write an automation selector you will be forced to use unstable locators.
5. Password sent in plaintext in login API request body.



* **Investigation Steps:**

1. Monitored the login API using Burp Suite.
2. Performed a login attempt with valid credentials.
3. Captured the request payload sent to the login endpoint.
4. Observed that the password field was transmitted in plaintext (unencrypted).

* **Information Gathering:**

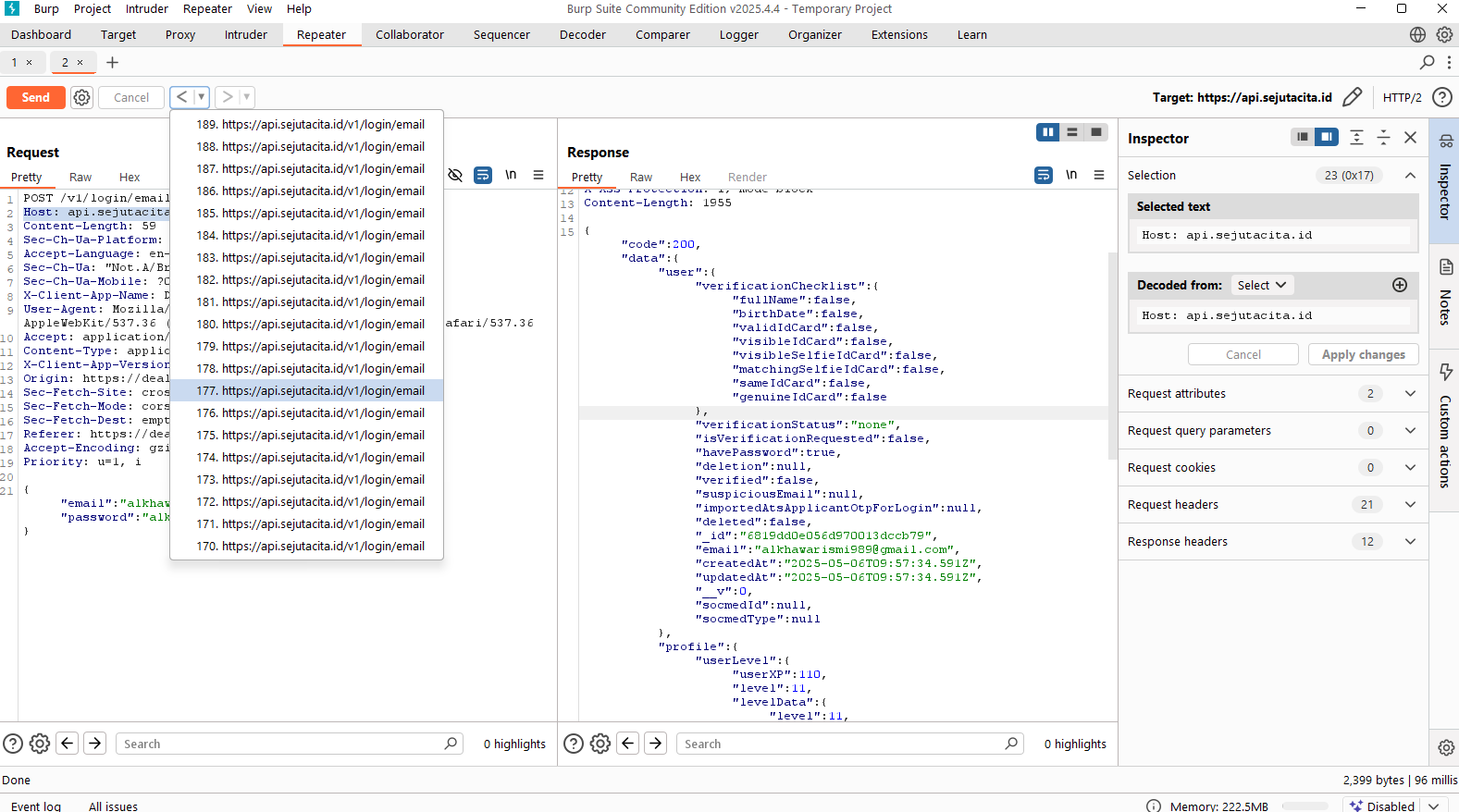
1. Tool Used: Burp Suite (intruder or repeater).
2. POST /v1/login/email HTTP/2
3. Host: api.sejutacita.id

* **Potential RC:**

1. Just relying on HTTPS for protection.
2. Lack of security awareness.

* **Reproduction Strategy:**

1. Launch Burp Suite and configure the browser to route traffic through Burp's proxy.
2. Log into the target application using valid credentials.
3. In Burp Suite, go to the HTTP history tab under the Proxy tab.
4. Locate the POST request to the login API endpoint.
5. Inspect the raw request body.
6. Observe that the password field appears in plaintext
7. Missing rate limiting on login API endpoint potential brute force attack.



* **Investigation Steps:**

1. Conducted a security test using Burp Suite Intruder to simulate multiple login attempts.
2. Targeted the login endpoint with a set of common or guessed passwords.
3. Observed that the API responded to each request without delay, blocking, or error after multiple attempts.
4. No CAPTCHA, cooldown period, or HTTP 429 (Too Many Requests) was observed during or after the attack.

* **Information Gathering:**

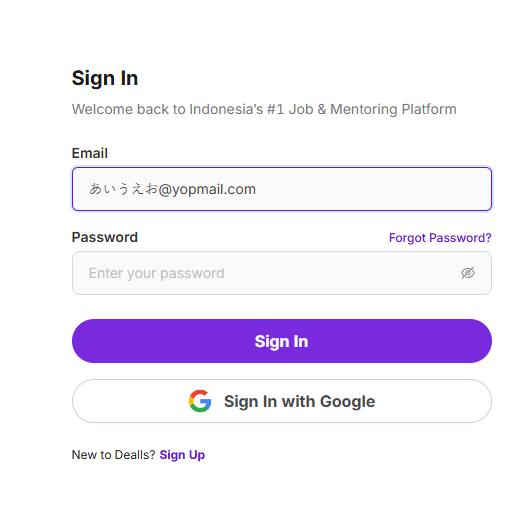
1. Tool Used: Burp Suite (intruder or repeater)
2. POST /v1/login/email HTTP/2
3. Host: api.sejutacita.id
4. Observed Behaviour: Login endpoint accepts unlimited requests without restriction or rate-based throttling.
5. Expected Behaviour: Endpoint should enforce limits (e.g., 5 attempts per IP per minute) and return 429 or similar.

* **Potential RC**:

1. Missing security control.
2. No bot mechanism.
3. Lack of security awareness.

* **Reproduction Strategy:**

1. Open Burp Suite and configure browser proxy.
2. Log in once to capture a valid login request.
3. Send the request to Repeater.
4. Keep send the request about 50 times or more.
5. Observe that the API responds with consistent status codes (e.g., 200, 401) and does not return 429 or trigger any blocking.
6. Email fields accept non-ascii char without any error.



* **Investigation Steps:**

1. Navigated to the user input form containing the email field.
2. Entered a non-ASCII email address: あいうえお@yopmail.com.
3. Submitted the form without receiving any validation error or warning.
4. Checked backend and client-side validation rules—no apparent restrictions against non-ASCII characters.
5. Verified that the system accepted and processed the input without complaints.

* **Information Gathering:**

1. Input Field: Email address input on login/signup or contact form.
2. Input Provided: Non-ASCII characters (Japanese Hiragana) in local-part of email (あいうえお@yopmail.com).
3. Observed Behaviour: No client-side or server-side validation error was shown.
4. Expected Behaviour: Validation should enforce compliance with standard email format rules.
5. Impact on System: Possible issues downstream (email delivery failure, data integrity issues).

* **Potential RC**:

1. Weak input validation.
2. Does not use RFC 5322 or RFC 6530 standard for email validation.

* **Reproduction Strategy:**

1. Open the form with the email input field.
2. Type or paste an email address containing non-ASCII characters in the local part (e.g., あいうえお@yopmail.com).
3. Submit the form.
4. Observe that no validation error appears and the form accepts and proceeds with submission.