Request Validation (Often Missed by Testers)

What to Validate in Requests

- Proper Method Usage → GET, POST, PUT, DELETE must be used appropriately.
- Mandatory Headers in Request → Authorization tokens, Content-Type.
- Request Payload Structure → Mandatory fields, correct formats, valid data types before sending.
- Correct Query Parameters → Validate correct params and optional vs required fields.
- Authentication Flow Validations → Is token refreshed correctly? Is expired token handled gracefully?

Validate Request with Mandatory Headers

```
given()
    .header("Authorization", "Bearer " + token)
    .header("Content-Type", "application/json")
    .body(validPayload) .when() .post("/users").then().statusCode(201);
```

Validate Proper Query Parameter Handling

```
given()
    .queryParam("filter", "active")
    .header("Authorization", "Bearer " + token)
.when().get("/users").then().statusCode(200);
```

Request Validation Continue.......

Negative Testing for Request Validation (Missing Required Field)

```
given()
    .header("Authorization", "Bearer " + token)
    .contentType(ContentType.JSON)
    .body(invalidPayloadMissingMandatoryField)
.when().post("/users").then() .statusCode(400);
```

Where Testers Commonly Find Bugs

- API works with invalid payload because backend doesn’t validate it properly.
- Optional parameters are handled poorly, causing hidden failures.
- Missing authentication headers leading to unauthorized access errors.
- Wrong HTTP method usage (e.g., using GET instead of POST) passing unexpectedly.

.

Interview Questions to Practice

✓ What are key request-side validations you always perform?
✓How do you test negative scenarios with invalid payloads?
✓How do you test APIs with dynamic query parameters?

.

Essential Response Data Validations

Status Code Validation

- Most Common Status Codes to Validate
- 200 OK – Successful GET request.
- 201 Created – Successful POST request (resource created).
- 204 No Content – Successful DELETE request.
- 400 Bad Request – Invalid input (missing mandatory fields).
- 401 Unauthorized – Invalid or missing auth token.
- 403 Forbidden – Authenticated but no access rights.
- 404 Not Found – Invalid endpoint or missing resource.
- 500 Internal Server Error – Backend crash.
- 503 Service Unavailable – Service downtime, often missed in non-prod.

Where Testers Commonly Find Bugs

- APIs silently returning 200 OK despite errors.
- Improper 401/403 handling — security loopholes.
- Lack of handling for missing resources (404).
- Unhandled server errors showing 500 instead of graceful messages.

Interview Questions to Practice

✓ Which status code would you expect when authentication fails?
✓ What's the difference between 401 and 403?
✓ How do you validate proper error handling via status codes?
✓ Explain a time you found a bug by checking the wrong status code.

Header Validation

Key Headers to Validate

- Content-Type → Always check if the response returns the correct data format like application/json.
- Authorization → Validate presence and correctness of authentication tokens.
- Cache-Control → Ensure sensitive APIs do not get cached unintentionally.
- Custom Headers → Check for tracing headers like X-Correlation-ID or localization headers like Accept-Language.

Header Validation

```
given()
    .when().get("/users/1")
    .then()
    .header("Content-Type", "application/json; charset=utf-8")
    .header("Cache-Control", "no-cache");
```

Where Testers Commonly Find Bugs

- Wrong or missing Content-Type breaks frontend parsing.
- Security misconfigurations from absent or expired Authorization headers.
- Cache issues returning stale data.
- Debugging blockers when trace IDs are missing in production.

Interview Questions to Practice

✓ Why is header validation important in secured APIs?
✓Name 3 headers you commonly validate and why.
✓How would you debug a production issue using headers?

Response Time Validation

Performance Indicators to Validate

- Validate API response time is within agreed SLAs (e.g., < 2 seconds).
- Check for increased latency during load tests or after deployments.

Response Time Validation

```
given()
    .when().get("/users/1")
    .then().time(lessThan(2000L));
```

Where Testers Commonly Find Bugs

- Slow backend services passing functional tests but failing performance SLAs.
- Missing timeout handling leads to user-facing failures.
- Performance regressions after code or infra updates.

Interview Questions to Practice

✓ How do you incorporate response time checks in automation?
✓What is the importance of response time monitoring?
✓How do you catch performance issues in your pipeline?

Payload Validation (The Real Value Zone)

Critical Fields to Validate

- Field presence and data accuracy (e.g., user ID, status fields).
- Correct formats for dates, numbers, and enums.
- Nested JSON validation and value assertions.

Code Snippet

```
given()
    .when().get("/users/1")
    .then()
    .body("name", equalTo("John Doe"))
    .body("status", equalTo("active"));
```

Where Testers Commonly Find Bugs

- APIs return null or empty fields.
- Wrong status transitions causing downstream failures.
- Mismatch in expected vs. actual formats.

Interview Questions to Practice

✓How do you validate dynamic response fields?
✓Explain a scenario where payload validation saved you in production.
✓Which library or method do you use for deep payload validation?

Schema Validation

Schema Checks to Include

- Validate response structure using JSON Schema.
- Assert mandatory fields, correct types, and value formats.
- Version-specific schema validations.

Schema Validation (with JSON Schema Validator)

```
given()
    .when().get("/users/1")
    .then()
    .assertThat()
    .body(matchesJsonSchemaInClasspath("user_schema.json"));
```

Where Testers Commonly Find Bugs

- Schema mismatches after backend updates.
- Contract-breaking changes that were unannounced.
- Integration failures from unversioned schema updates.

Interview Questions to Practice

✓What tools do you use for schema validation?

✓Why is schema validation crucial in microservices architecture?

✓Have you experienced a contract break? How did you catch it?

Request Validation – Advanced Ideas

- Boundary Value Testing → Test min/max length, number ranges in request payload fields.
- SQL/Script Injection Testing → Validate that the API properly rejects malicious inputs.
- Content-Type Negative Testing → Send incorrect Content-Type headers (e.g., text/plain instead of application/json) and ensure proper rejection (415 Unsupported Media Type).
- Empty Request Validation → Send requests without body where required and validate 400 Bad Request.
- Incorrect Method Testing → Use wrong HTTP methods (e.g., DELETE instead of GET) and expect proper rejection (405 Method Not Allowed).
- Duplicate Requests → Test idempotency where applicable (especially in PUT and DELETE).

Response Validation – Additional Ideas

- Empty Payload Handling → Ensure APIs return 204 No Content when appropriate.
- Optional Field Validation → Validate presence/absence of optional fields based on request type or user role.
- Correct Default Values → Check if APIs return correct defaults when optional inputs are skipped.
- Pagination Validations → For paginated APIs, validate page numbers, limits, and hasNext flags.
- Data Sorting → Verify response data is sorted correctly when requested (e.g., /users?sort=asc).
- Error Message Clarity → Validate not just status codes but also meaningful, user-friendly error messages in the response.
- Chained API Flow Validation → Validate data flow across multi-step API chains