Assertions That Matter – What to Validate Beyond Status Code

Audience: QA / Automation Engineers
Mindset: Senior QA reviewing real production tests

---

0. Let's kill the most dangerous QA habit

> ❌ *"API returned 200, test passed."*

This is **lazy testing**.

A status code only tells you:

- Request was syntactically accepted

It tells you **nothing** about:

- Data correctness
- Business success
- Side effects
- Corruption

Senior QAs never stop at status codes.

---

1. Levels of assertions (this is how pros think)

Think in **layers**, not single checks:

1. Transport-level (status, headers)
2. Contract-level (schema)
3. Business-level (meaning)
4. Persistence-level (DB / downstream)

If you only validate level 1 → your tests are weak.

---

2. Status ≠ Success (real examples)

Example 1: Fake success

HTTP 200

```
{
  "message": "Order created successfully"
}
```

Reality:

- Order not inserted into DB
- Event failed after response

➡️ Status passed, business failed.

Example 2: Async systems

HTTP 202 Accepted

This means:

- Processing NOT done
- Failure may still happen

If you assert only status here — you're asserting **nothing useful**.

---

3. Schema validation (non-negotiable baseline)

What schema validation catches

- Missing mandatory fields
- Wrong data types
- Contract breaks

What it does NOT catch

- Wrong values
- Wrong business state
- Partial failures

Schema validation is **necessary but never sufficient**.

Example: Basic schema assertion

Validate:

- id exists
- id is not null
- id is correct type

But don’t stop here.

---

4. Business validations (this is where quality lives)

Ask these questions every time

- Does the response make **business sense**?
- Is the state transition valid?
- Is the data usable downstream?

Example: Create Order API

Response:

{

"orderId": "ORD-123",
"status": "CREATED"
}

Weak assertion:

- status code = 201

Strong assertions:

- orderId exists and follows format
- status = CREATED (not hardcoded text)
- order is retrievable via GET API

---

5. Validate DB ID presence (this is critical)

Why message-only validation is useless

This is common and wrong:

"message": "User created successfully"

Messages:

- Are cosmetic
- Change frequently
- Lie often

What actually matters

You must validate:

- Primary key / ID generated
- ID persisted in DB
- ID usable in next API

Correct validation mindset

After CREATE API:

Validate at least ONE of these:

- ID exists in DB
- GET API works using returned ID
- Event published with same ID

If none are validated → test has no value.

---

6. Persistence-level assertions (DB / downstream)

When DB validation is justified

Use DB checks when:

- Data correctness is critical
- Async processing involved
- No reliable GET API exists

Do NOT validate every column blindly.

What to validate in DB

- Record exists
- Primary key matches response
- Status column is correct
- No duplicate records

Avoid:

- UI-driven DB checks
- Over-asserting every field

---

7. Negative business assertions (most teams miss this)

Example: Duplicate create

Test:

- Send same payload twice

Validate:

- Second request rejected OR
- Same ID returned (idempotency)

Not:

- Two records created silently

---

8. Assertion anti-patterns (call these out in reviews)

❌ Asserting only status code
❌ Asserting response message text
❌ Hardcoding IDs in assertions
❌ Asserting everything blindly

These tests pass but catch nothing.

---

9. Minimum assertion checklist (use this)

For every API test, ask:

- ✅ Is status correct?
- ✅ Is response schema valid?
- ✅ Is business state correct?
- ✅ Is data persisted / retrievable?

If you can’t answer one → test is incomplete.

---

10. One-line senior QA rule

> **If your assertion doesn’t protect the business, it’s not an assertion — it’s noise.**

If you want next:

- Sample Postman / automation assertions
- Async API assertion patterns
- DB validation dos & don’ts
- Review checklist for API test cases

Say the word.