01/01/2026

PART 1: WHAT AN API REALLY IS (BRUTAL TRUTH)

Most people think:

> "API = URL + request + response"

That's child-level understanding.

Reality:

An API is a doorway into a system. You're not testing the door. You're testing what happens inside the building after you enter.

1. **API returned 200 OK. Is it successful?**

   200 only means the request was processed at the API layer. Actual business success depends on downstream operations.

   200 OK only confirms the request was accepted by the API layer. I'd verify downstream processing like database persistence, async jobs, and external integrations to confirm business success

2. You hit **Order API**, get 200 OK. Users say orders are missing after 10 minutes. Name **3 downstream checks** you'd do.

   First I'd check whether the data was persisted correctly in the database and verify the order state. Then I'd confirm whether any async processing or message queues handled the request successfully. Finally, I'd validate downstream integrations like payment or inventory services to see where the failure occurred and how it impacted the user.

PART 2: REQUEST ≠ BUSINESS COMPLETION

Let's take a real example.

Order Placement API
POST /orders

Payload:

```
{
 "productId": 123,
 "quantity": 2,
 "paymentMode": "UPI"
}
```

What you THINK happens:

1. API called
2. Order placed
3. Response returned

What ACTUALLY happens:

1. API receives request
2. Validates payload
3. Saves order as PENDING
4. Pushes message to queue
5. Returns response immediately
6. Payment service processes later
7. Inventory updated
8. Order marked CONFIRMED or FAILED

Steps 6–8 happen AFTER the response

2. How do you test if order was really created?

Verify persistence in DB or by fetching order status after processing completes.

I verify persistence by querying the database using the order ID returned by the API and confirm the final business state.

If the API is asynchronous, I validate order creation by polling the order status API until it reaches a terminal state.

Response confirms request acceptance, not business completion.

I don't rely on response body alone. I verify order creation by either validating database persistence using the order ID or by calling a follow-up API to confirm the order reaches a stable business state after processing completes."

POST /orders returns 200. Verified orderId exists in DB. Initial status PENDING. After async processing, status updated to CONFIRMED. This confirms real order creation.

SYNC TESTING MINDSET

"Send request → check response → done"

ASYNC TESTING MINDSET

"Send request → wait → verify state → verify side effects"

3. Why use async instead of sync?

Async is used to improve scalability, avoid timeouts, and handle slow or unreliable downstream systems. It allows the API to respond quickly while business processing continues in the background . But it requires additional testing to verify eventual consistency and downstream success

4. **If a payment API was made synchronous instead of async, name two production problems that could happen.**

Or

If a payment API was synchronous instead of async, what production problems could happen?

Synchronous payment APIs can cause client timeouts, leading to retries and duplicate payments.

Synchronous processing blocks server threads, reducing scalability and potentially causing outages under load.

If a payment API were synchronous, it could lead to client timeouts causing duplicate charges, and it could also block server threads under load, reducing scalability and potentially causing outages

5. If an async payment succeeds but inventory fails later, what is the *correct* system behavior?

The system should trigger a compensation flow such as refunding the payment or retrying inventory allocation, ensuring consistency between payment, inventory, and order state.

6. As a tester, how would you verify that a refund actually happened after inventory failure?

I'd verify the order status API reflects a failed or refunded state, confirm via payment service or database that a refund transaction was successfully processed, ensure inventory reservations were released, and finally validate there are no duplicate orders or payments created

Inventory allocation failed → order moved to FAILED → refund transaction created and completed → inventory reservation released → no duplicate order or payment records

7. Why is checking only order status API NOT enough to confirm a refund?

Order status alone is not sufficient because it only reflects the order service state. A refund must be verified in the payment system to confirm that the financial transaction actually occurred.

8. What is the single most dangerous assumption in API testing?

The most dangerous assumption in API testing is believing that a successful HTTP status code guarantees business success.

PART 3: SYNC vs ASYNC (NO DEFINITIONS, ONLY BEHAVIOR)

Synchronous API (SYNC)

- Client waits
- Processing completes
- Response = final truth

Example:

GET /users/123

If response says user exists → it exists.

Asynchronous API (ASYNC)

- Client sends request
- System says "Got it"
- Processing happens later

Example:

POST /payments

Response:

200 OK / 202 Accepted

This does NOT mean payment succeeded.

9. API returns 200, but users complain after 5 minutes. What do you check?

Was processing async?
Is there a queue?
Did worker fail?
Did callback/webhook fail?
Did DB update fail after response?

PART 5: WHAT A REAL API TESTER THINKS (DETAILED)

A real API tester thinks beyond request-response and validates downstream processing, data persistence, system interactions, and user impact.

PART 6: WHERE TO CHECK FAILURES (DB, LOGS, QUEUES)

DATABASE (FIRST STOP)

Why DB first?

- DB = system of record
- Tells you if business state exists

What you check

- Does record exist?
- What is status?
- Was it rolled back?
- Was it updated later?

DB answers:

- “Never created”
- “Created but stuck”
- “Created then deleted”

LOGS (SECOND STOP)

Logs explain **WHY DB looks the way it does**.

Types:

- API logs → request accepted?
- Service logs → business logic
- Error logs → exceptions
- Retry logs → async failures

What logs tell you

- Where failure happened
- Whether it was retried
- Whether it was ignored

API → DB → Queue → Worker → External System

PART 7: LOGS, QUEUES & REAL FAILURE DIAGNOSIS

Scenario

- API returns 200

- Orders missing after 10 minutes

Step-by-step diagnosis

1 Check DB

- Order exists?
- Status = PENDING?

2 Check logs

- Payment initiated?
- Any timeout?
- Any retry?

3 Check queue

- Message in queue?
- Consumer running?
- Message in dead-letter queue?

4 Check external service

- Payment gateway response?
- Timeout or rejection?

API accepted request, queue consumer failed due to timeout, order stuck in PENDING.

<u>PART 8: HOW TO TEST ASYNC APIs PROPERLY</u>

POLLING (MOST COMMON)

What it is

Repeatedly check status until final state.

Example

POST /orders

→ orderId

GET /orders/{id}

→ PENDING

→ PENDING

→ CONFIRMED

What you validate

- Status eventually changes
- Timeout handled

- No infinite pending

2 CALLBACKS / WEBHOOKS

What it is

System notifies you later.

What to test

- Callback triggered?
- Payload correct?
- Retry on failure?
- Signature/security?

3 RETRIES & IDEMPOTENCY

Why this matters

Async systems retry.
Retries cause duplicates if badly designed.

What you test

- Retry creates only ONE order
- Same request ID = same result

4 NEGATIVE ASYNC TESTING (ADVANCED)

Test:

- Queue down
- Worker crash
- External timeout

Verify:

- Order state updated
- Compensation triggered
- User informed

9. What are the FIRST 3 things you check and WHY (in order)?

I'd first verify the order record and status in the database to confirm persistence. If it's stuck in PENDING, I'd check the async processing mechanism—queues or workers—to see whether the message was consumed or failed. Finally, I'd inspect logs to identify the exact reason processing didn't progress