REST APIs – Explained Simply

This document explains REST APIs in a **child-simple way**, but with **real production accuracy**. No theory fluff. This is how APIs actually behave in real systems.

---

1. REST Is About STATE

What is State?

State means the **current condition** of something.

Example (real life):

- Door CLOSED → Door OPEN → Door LOCKED

Example (API world):

- Order **CREATED**
- Order **PROCESSING**
- Order **COMPLETED**
- Order **CANCELLED**

REST APIs are responsible for **moving a resource safely from one state to another**.

---

2. Resource State Transitions

State transitions must follow rules.

Order Example (Pizza Story)

1. Order placed → **CREATED**
2. Restaurant starts cooking → **PROCESSING**
3. Pizza delivered → **COMPLETED**

Invalid transitions:

- COMPLETED → PROCESSING
- COMPLETED → CREATED

A good API **blocks invalid transitions**.

---

3. PUT vs PATCH (Interview Favorite)

PUT – Full Replacement

- Replaces the **entire resource**
- Client sends **all fields**, even unchanged ones
- Always **idempotent**

Example:

```
PUT /orders/101
{
  "status": "PROCESSING",
  "amount": 500
}
```

Sending this request 1 time or 10 times → **same final result**.

PATCH – Partial Update

- Updates **only specific fields**
- Used for small changes
- May or may not be idempotent (depends on implementation)

Example:

```
PATCH /orders/101
{
  "status": "PROCESSING"
}
```

Easy Memory Trick

- **PUT = replace everything**
- **PATCH = change one part**

---

4. HTTP Is NOT Just Status Codes

Most beginners think:

- 200 = success
- 400 = failure

Reality is more dangerous.

---

5. Important HTTP Status Codes

200 OK

- Work completed **now**
- Data is final

201 Created

- Resource created
- New ID usually returned

202 Accepted (VERY IMPORTANT)

- Request accepted
- Work is happening **later**
- Result is **not ready yet**

202 is where **most payment and async bugs happen**.

---

6. What Happens After 200 / 202?

Typical real flow:

Client
→ API Gateway
→ Auth / Rate Limit
→ Service
→ Database
→ Queue
→ Background Worker

Even after 200 or 202:

- DB may fail
- Message queue may fail
- Worker may crash
- Network may retry

Status code does **not guarantee business success**.

---

7. Idempotency (Production Killer Concept)

Child-Simple Meaning

> Clicking the same button many times should do the work **only once**.

Example:

- User clicks **Pay ₹500**
- Network slow
- User clicks again

Without idempotency → Charged twice

With idempotency → Charged once

---

8. Idempotency-Key (How Systems Prevent Duplicates)

Simple Rule

Each request has a **unique key**.

Example header:

Idempotency-Key: abc-123

What Server Does

1. Receives request with key
2. Stores result for that key
3. Same key comes again
4. Returns **same result**, no re-processing

---

9. Real Payment Failure Scenario

Scenario

- Payment API returns **202 Accepted**
- User refreshes page
- Payment charged twice

Why This Happens

- Client retried request
- No Idempotency-Key
- Backend processed request again

Who Is At Fault?

- Client retry logic
- Missing idempotency
- Backend not protecting duplicates

---

10. Sample Order API Design

API Flow

POST /orders → CREATED
POST /orders/{id}/process → PROCESSING
POST /orders/{id}/complete → COMPLETED

Rules

- Cannot complete before processing
- Same request repeated → same result
- Payment actions must be idempotent

---

11. Why QA Must Understand This

If you don't understand this:

- You test only happy paths
- You miss retry bugs
- You miss async failures

If you do understand this:

- You catch real production issues
- You design meaningful API tests
- You think like backend + QA

Final Takeaway

REST is not about URLs or JSON.

REST is about:

- **State control**
- **Safe transitions**
- **Handling retries and failures**

Master this → you move beyond beginner QA.