Rest API Development Example: Patterns, Strengths, and Pitfalls

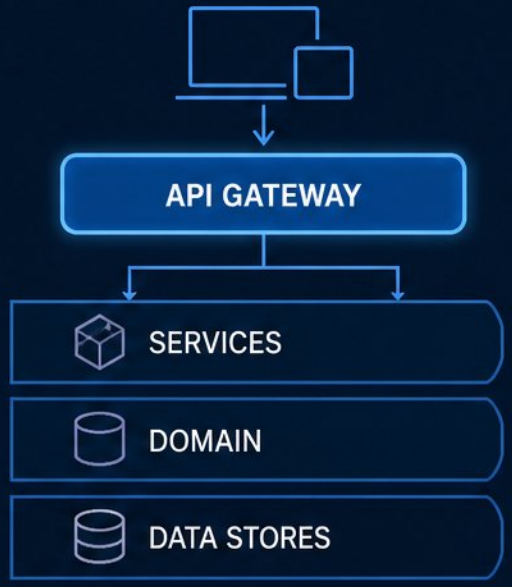
- REST remains the default for public, cross-language APIs in 2026
- Resource modeling, HTTP semantics, security, and contract automation
- For backend developers and API designers seeking durable, predictable contracts
- Learn resource shapes, status codes, pagination, versioning, and error formats

REST Foundations and the Richardson Maturity Model

- REST is an architectural style, not a protocol, defining core constraints like statelessness and cacheability.
- The Richardson Maturity Model has 4 levels: RPC, resources, HTTP verbs, and hypermedia.
- Most production APIs intentionally stop at Level 2, using OpenAPI for discoverability instead of HATEOAS.
- 'RESTful' implies stricter compliance than simply using HTTP + JSON; semantics are key.

Resource Modeling and URI Design

- - Model domain nouns, not database tables or storage internals
- - Use plural collections with singular items: /orders, /orders/{id}
- - Kebab-case paths, lowercase, no trailing slashes, no verbs in URLs
- - Nest ownership up to two levels; flatten deeper with query filters
- - Non-CRUD actions: POST /orders/{id}/cancel, not scattered RPC verbs



Anatomy of a Well-Designed URI

/ orders / {id} / items / {item-id}

- orders — Plural collection (resource set)
- {id} — Resource ID (singular item)
- items — Sub-resource (nested up to two levels)
- {item-id} — Sub-resource ID (singular item)

Good examples

/orders

/orders/{id}

/orders/{id}/items

Lowercase, kebab-case, no trailing slashes, no verbs.

Anti-Patterns to Avoid

- /getOrders — No verbs in URLs
- /orders/ — No trailing slashes
- /Orders — No uppercase letters
- /orders/{id}/items/{id}/details — Avoid deep nesting

HTTP Semantics in Practice

- - GET, PUT, DELETE are idempotent; POST is not
- Map outcomes to precise status codes (200, 201, 204, 404)
- Never return 200 with an error body
- PATCH is for partial updates; may not be idempotent
- Content negotiation keeps resources stable across versions

Consistent Errors with RFC 9457

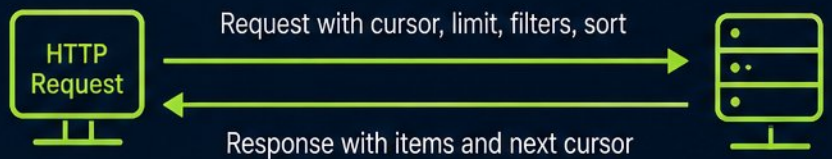
- RFC 9457 standardizes errors with application/problem+json
- Core fields: type, title, status, detail, instance
- Add extensions for field-level validation errors
- Use absolute type URIs linking to documentation
- Never leak stack traces; return all validation errors at once
- Keep detail client-focused, not a debugging tool

Pagination, Filtering, and Sorting

CURSOR PAGINATION

Seek efficiently, stay consistent

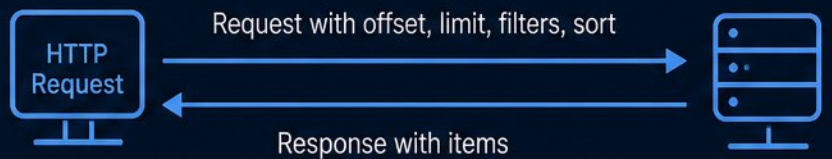
A bookmark (cursor) remembers your place.
Jump forward/backward in **O(log n)** time.



OFFSET PAGINATION

Scan and count, less reliable

Offset requires counting from the start.
Slower **(O(n))** and can cause duplicates or skips.



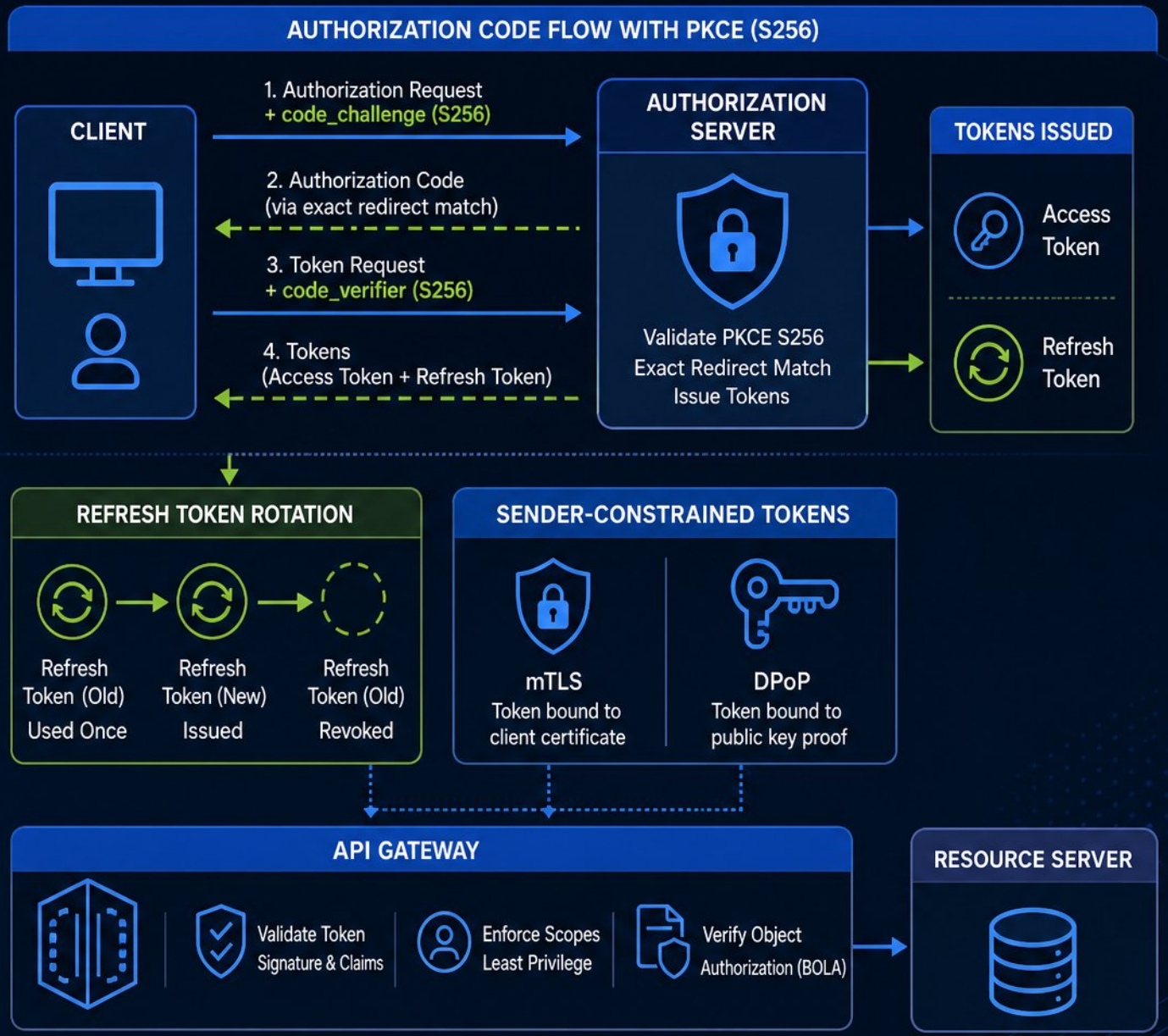
- Cursor pagination: O(log n) seek, stable under writes; offset degrades to O(n), causes duplicates or skips
- Sort by (created_at, id) composite cursor for deterministic ordering
- Enforce limit defaults and maximums to protect against accidental DDoS
- Stable filter and sort conventions (sort=-createdAt, limit/after) for predictable results

Versioning, Compatibility, and Deprecation

- URI path versioning (/v1/) is the pragmatic default: visible, cacheable, debuggable
- Date-based header versioning works at Stripe/GitHub scale but needs transformation layers
- Additive-only changes are safe; renames, removed fields, tightened validation are breaking
- Use Deprecation and Sunset headers with a 12-month minimum runway
- OpenAPI deprecated flags automate migration signals for clients

Authentication, Authorization, and Token Hygiene

- OAuth 2.1: PKCE S256, exact redirect match, short-lived tokens
- Implicit and password grants removed; Authorization Code flow for all
- Scopes enforce least privilege; verify object authorization (BOLA)
- Sender-constrained tokens (mTLS/DPoP) and refresh rotation reduce replay



Rate Limiting and Traffic Control

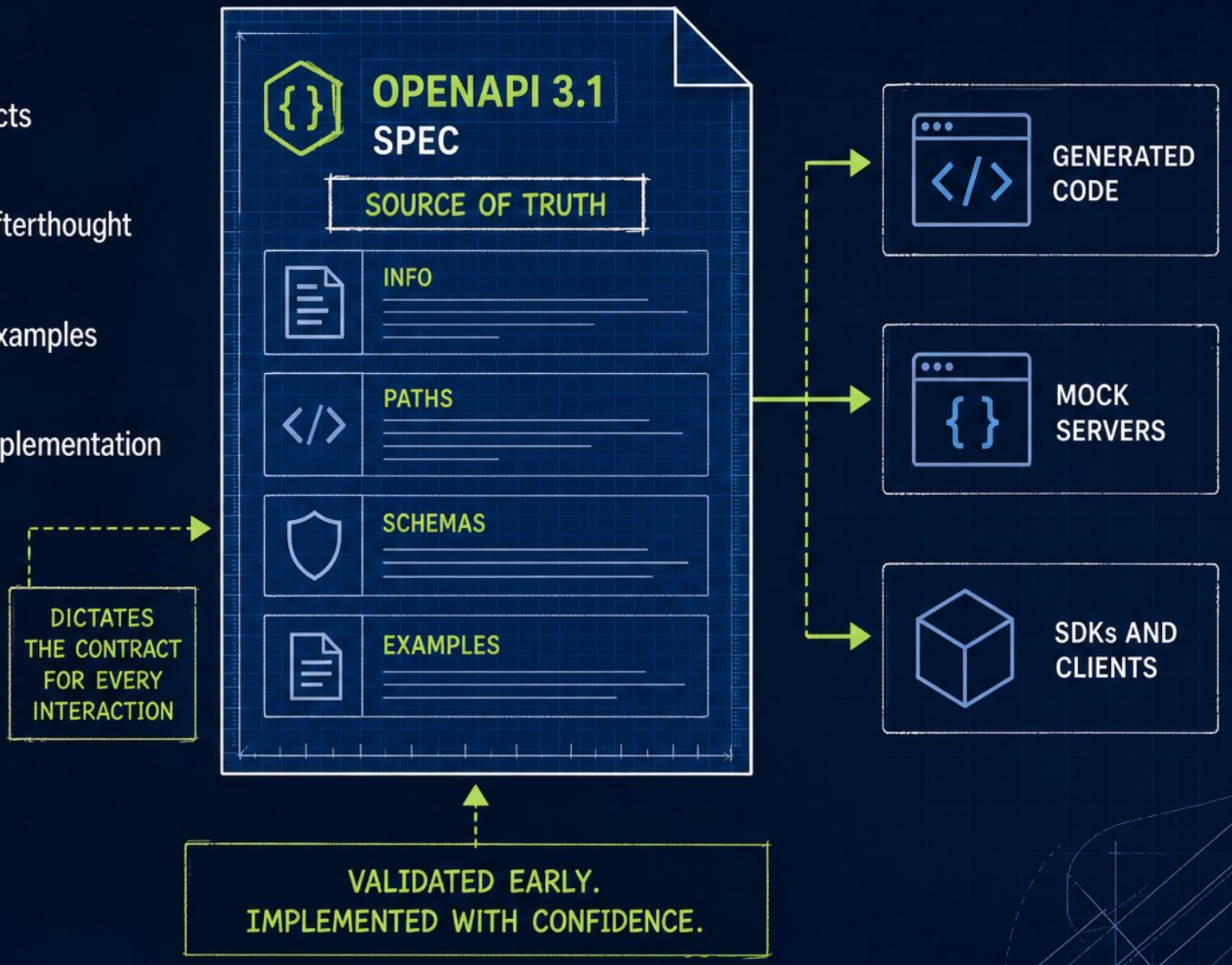
- Enforce rate limits at the gateway with per-tenant and per-endpoint policies
- Return Retry-After on 429; expose X-RateLimit-Limit, Remaining, and Reset headers
- Exponential backoff with jitter prevents synchronized retry storms
- Conditional requests (If-None-Match) avoid consuming rate-limit quota

Caching, Performance, and Observability

- HTTP caching with Cache-Control, ETag, and Last-Modified reduces load
- Track latency at p50, p95, and p99; averages hide tail behavior
- Golden signals: latency, traffic, errors, saturation
- OpenTelemetry unifies traces, metrics, and logs
- Correlation IDs link logs to traces for debugging

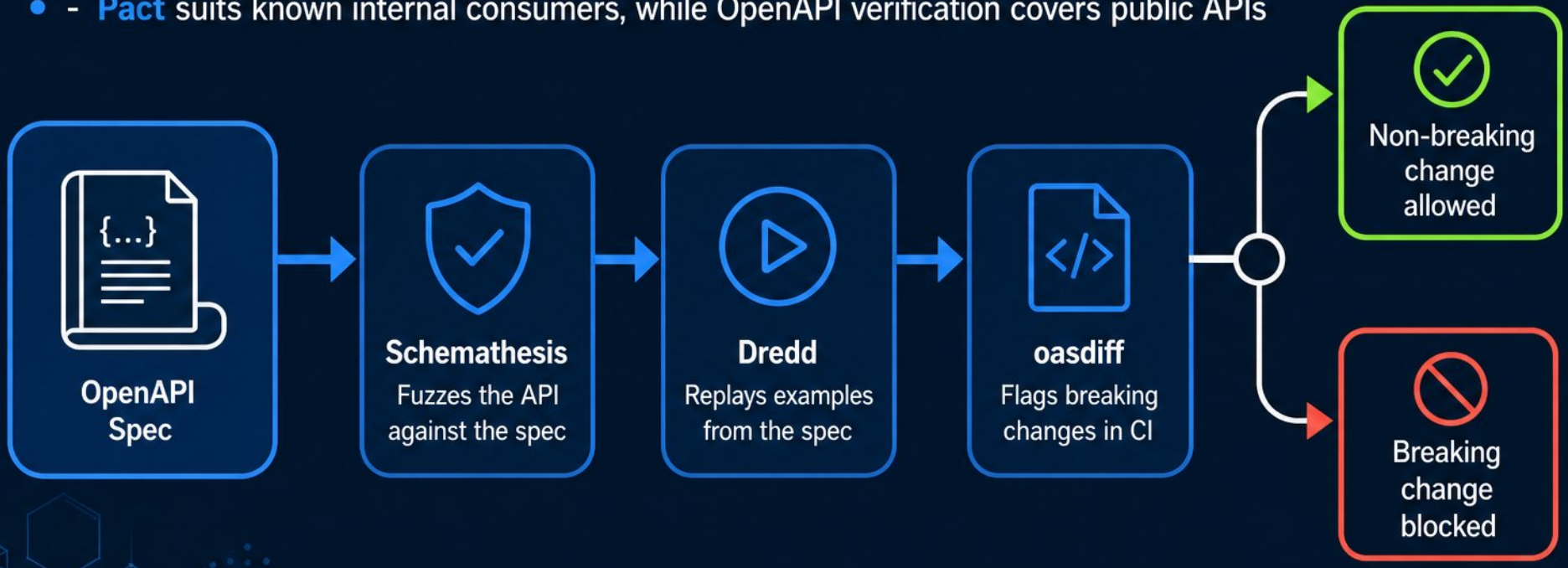
OpenAPI and Contract-First Development

- OpenAPI 3.1 dictates contracts
- Spec is source of truth, not afterthought
- Lint with Spectral; validate examples
- Run contract tests before implementation
- Specs guide AI agents: use intent-revealing fields



Contract Testing and Change Detection

- - **Contract testing**: live responses match OpenAPI spec (status, types, schemas, headers)
- - **Schemathesis** fuzzes, **Dredd** replays examples, **openapi-examples-validator** checks static examples
- - **oasdiff** flags breaking changes in CI before merge
- - **Pact** suits known internal consumers, while OpenAPI verification covers public APIs



Documentation, SDKs, and Developer Experience

- OpenAPI-generated docs, SDKs, and mocks keep artifacts in sync
- Generated SDKs beat hand-written ones for most APIs
- Surface request IDs; document retry behavior and sandbox
- Spec fields act as copy for AI agents and human developers

Pitfalls, Trade-offs, and When REST Is Not the Right Fit

- Avoid anti-patterns: 200-for-errors, mixed casing, deep nesting
- REST: broad client support, cacheability, stable contracts
- HATEOAS adds complexity; cursor pagination loses random access
- Choose gRPC for mesh, GraphQL for flexible queries, SSE for streaming

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/438c6844/public