

API Development Checklist:

A Step-by-Step Review

- Why structured reviews reduce incidents, support load, and integration friction
- Covers full lifecycle: design, security, launch, monitoring, deprecation
- Built for developers, technical leads, and API product owners
- Fourteen focused sessions mapped to real review gates and team workflows



The Cost of Skipping Structured Review



- Breaking changes, inconsistent errors, missing auth, unclear pagination



- Silent contract drift that human review and CI checks miss



- Real costs: agent retries, schema mismatches, multi-day detection gaps



- Lightweight review for additive changes; full review for new contracts



Design Review: Contracts, Naming, and Versioning



- Verify resource naming aligns with internal and ecosystem conventions



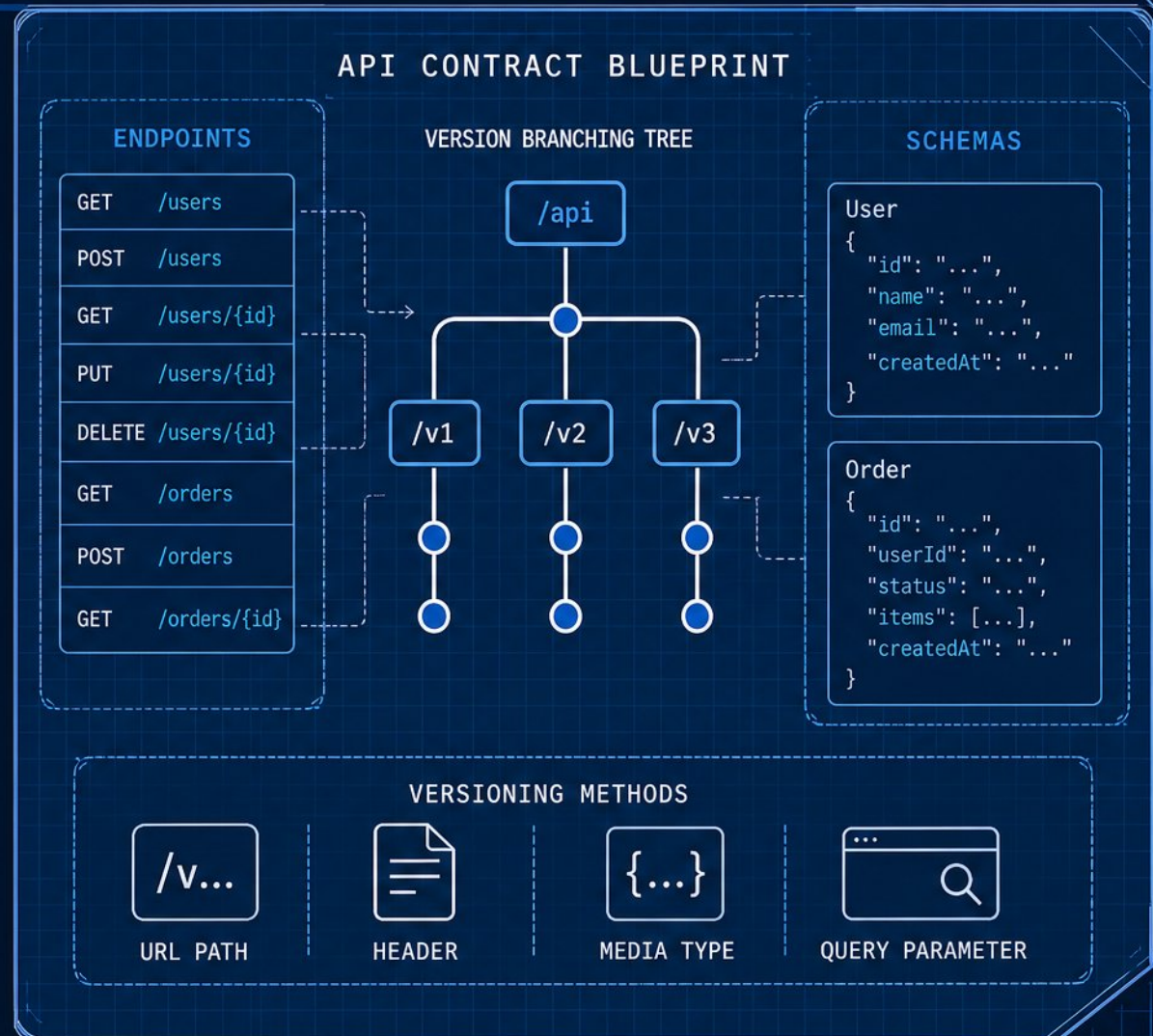
- Review schemas for ambiguity, forward compatibility, and evolution strategy



- Select versioning method: URL path, header, media type, or query parameter



- Treat the OpenAPI spec as a binding contract before implementation, not after



Security and Access Control Checks

- OAuth 2.0, API keys, and mTLS for internal versus external consumers
- Object-level, function-level, and property-level authorization with least privilege
- BOLA, mass assignment, SSRF, and resource consumption mapped to OWASP API Top 10
- Secrets handling, transport security, and data exposure checks before release



Error Handling Standards

- ✓ - Adopt one consistent error envelope, such as RFC 9457 problem details
- ✓ - Use stable machine-readable codes; keep human-readable messages separate
- ✓ - Document retry behavior per error type; include field-level validation errors
- ✓ - Include correlation IDs for support; never leak stack traces or internals

ANATOMY OF A STANDARDIZED ERROR ENVELOPE

```
{
  "type": "about:blank",
  "title": "Invalid input",
  "status": "400",
  "detail": "The request contains
  invalid data.",
  "instance": "urn:example:request:abc123",
  "code": "VALIDATION_ERROR",
  "errors": [
    { "field": "email", "message": "Must be
    a valid email address" }
  ]
}
```



Type

Identifies the problem type (RFC 9457)



Title

Short, human-readable summary



Status

HTTP status code as a string



Detail

Human-readable explanation



Instance

URI reference for this occurrence



Code

Stable, machine-readable error code



Errors

Field-level validation errors



Correlation ID

Include a correlation ID in headers or the envelope for support and traceability.



Never leak stack traces or internal details.

Observability and Alerting Requirements



- Instrument request rate, error rate, latency, saturation, and consumer failures



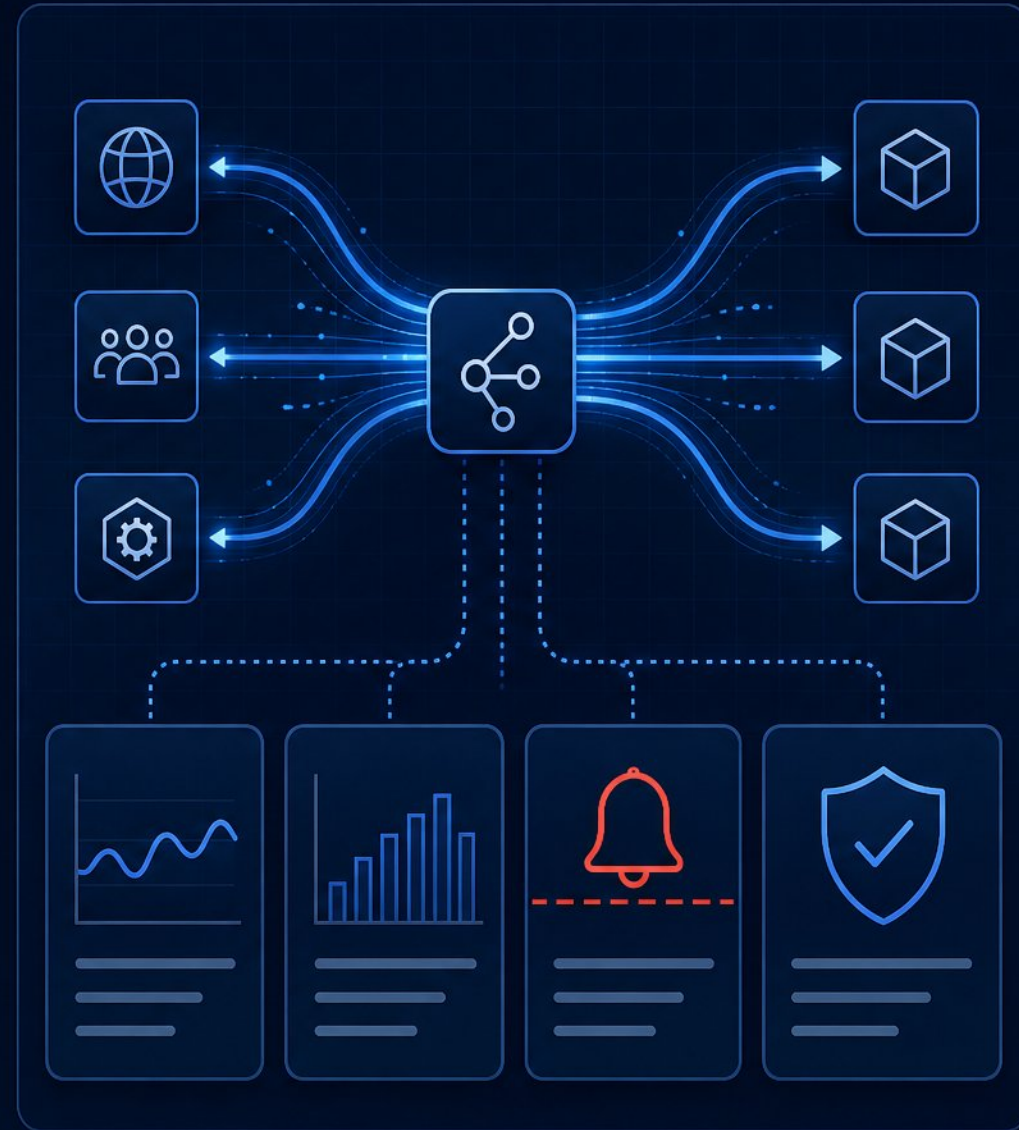
- Use structured logs with correlation IDs, trace propagation, and redacted fields



- Define SLOs and alert on consumer symptoms, not every spike

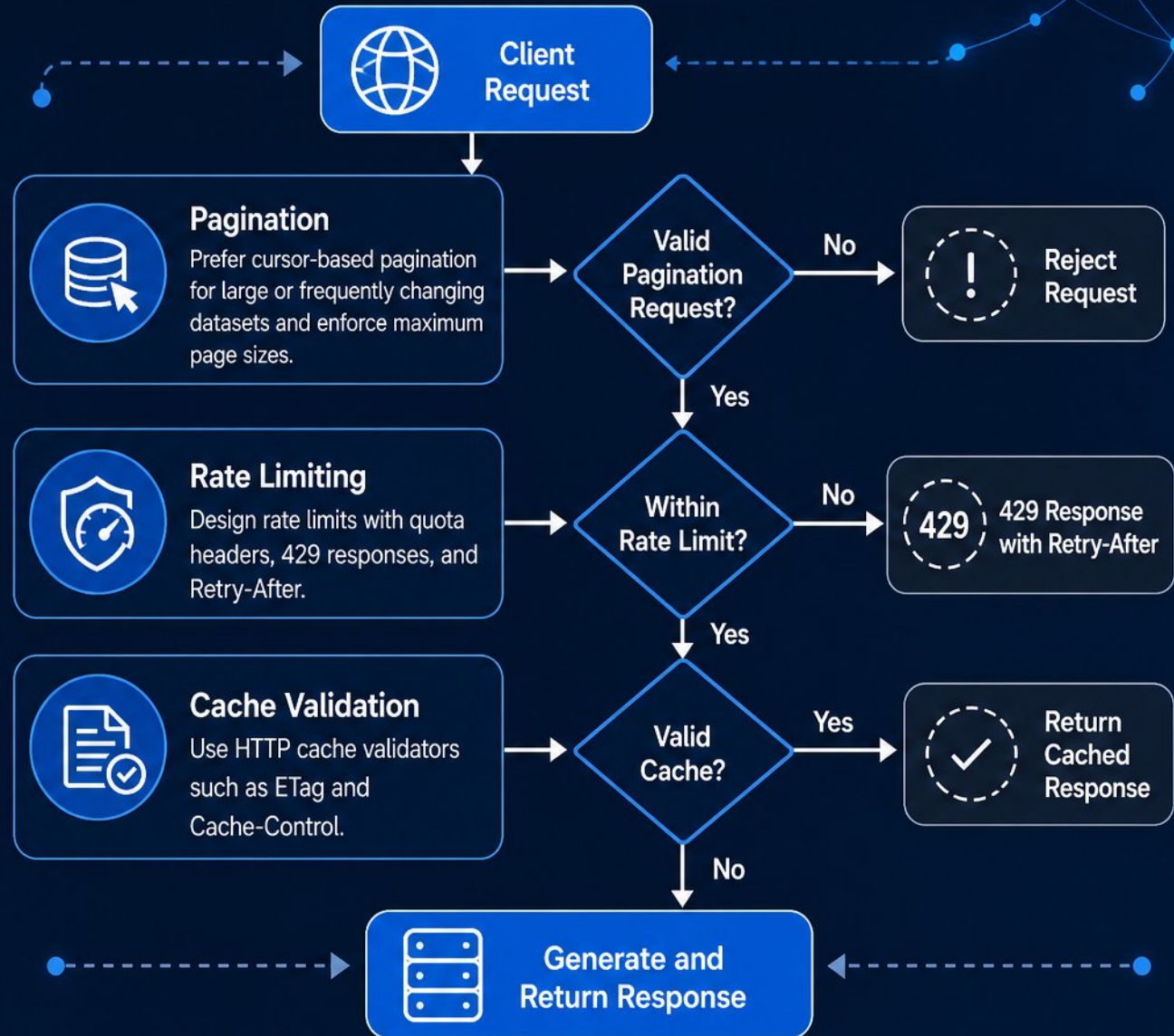


- Verify dashboards, runbooks, and alert paths before launch



Performance, Pagination, and Rate Limiting

- Prefer cursor-based pagination for large or frequently changing datasets
- Set and enforce maximum page sizes; return useful pagination metadata
- Design rate limits with quota headers, 429 responses, and Retry-After
- Use HTTP cache validators such as ETag and Cache-Control
- Cache-Control: private for user data; public for reference data



Idempotency and Safe Retries



- Require Idempotency-Key headers on all state-changing operations



- Cache responses for a documented deduplication window, typically 24 hours



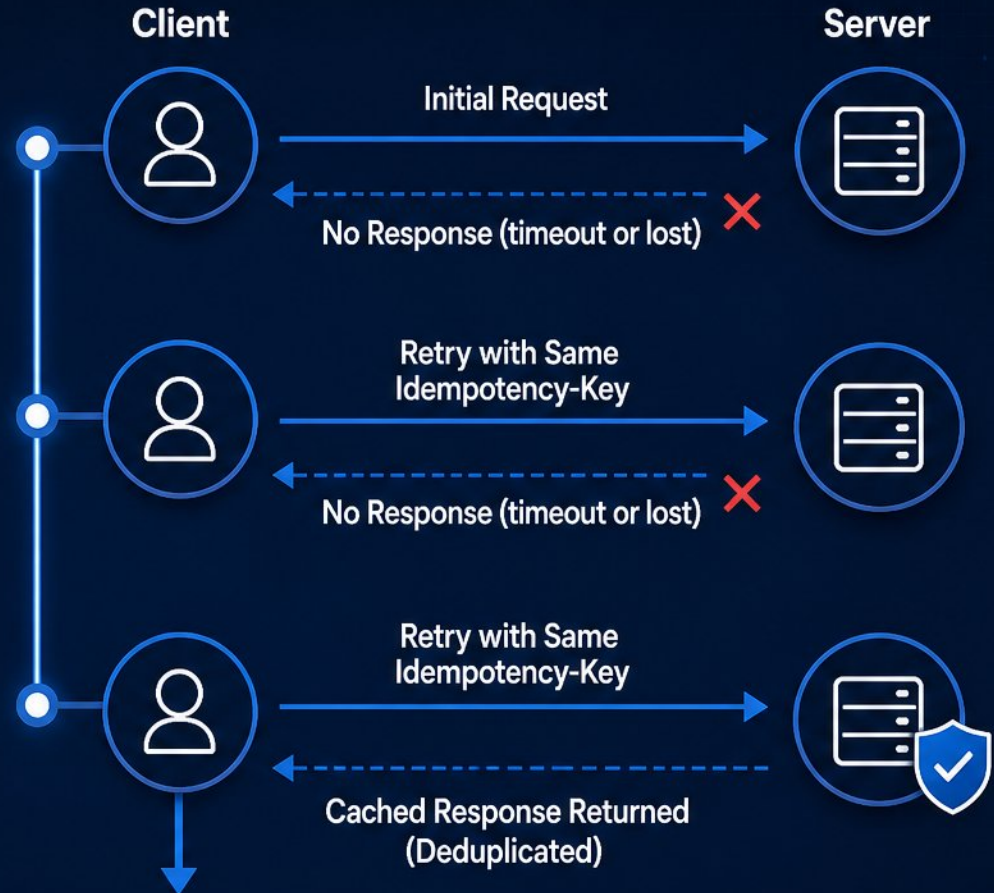
- Distinguish retryable transient failures from permanent validation errors



- Define conflict handling and duplicate request behavior in the contract



- Test retry paths against real timeouts and lost responses



Idempotency-Key ensures duplicate requests within the deduplication window return the original response safely.

Testing and Pre-Release Verification

- Contract, integration, load, negative, and backward compatibility tests are mandatory.
- Catch breaking changes early with spec diffs and consumer-driven contract tests.
- Validate examples and edge cases using mock servers and sandbox environments.
- Verify compatibility against real production traffic, not only spec diffs.

Contract Tests



Ensure provider and consumer agree on the API contract.

Use spec diffs and consumer-driven contract tests.

Load Tests



Validate API behavior under expected and high load conditions.

Confirm stability across integrations.

Backward Compatibility Checks



Verify new changes do not break existing consumer integrations.

Maintain compatibility across versions.



API Gateway
Portal



Release

Documentation and Developer Experience Review



- Verify endpoint references, realistic examples, and quick-start guides



- Confirm interactive docs, SDK generation, and auth explanation



- Review changelog, migration guides, and deprecation notices



- Treat documentation as part of the release: test in CI

API Gateway Portal

Endpoint Reference

```
GET /v1/resources
GET /v1/resources/{id}
```

Example Request

```
curl
curl -X GET https://api.example.com/v1/resources
-H "Authorization: Bearer <token>"
-H "Accept: application/json"
```

Interactive Documentation

Try it

Request

>_

Response

{...}

SDK Generation

Python

JavaScript

Go

Java

```
# Install SDK
pip install api-gateway-sdk
```



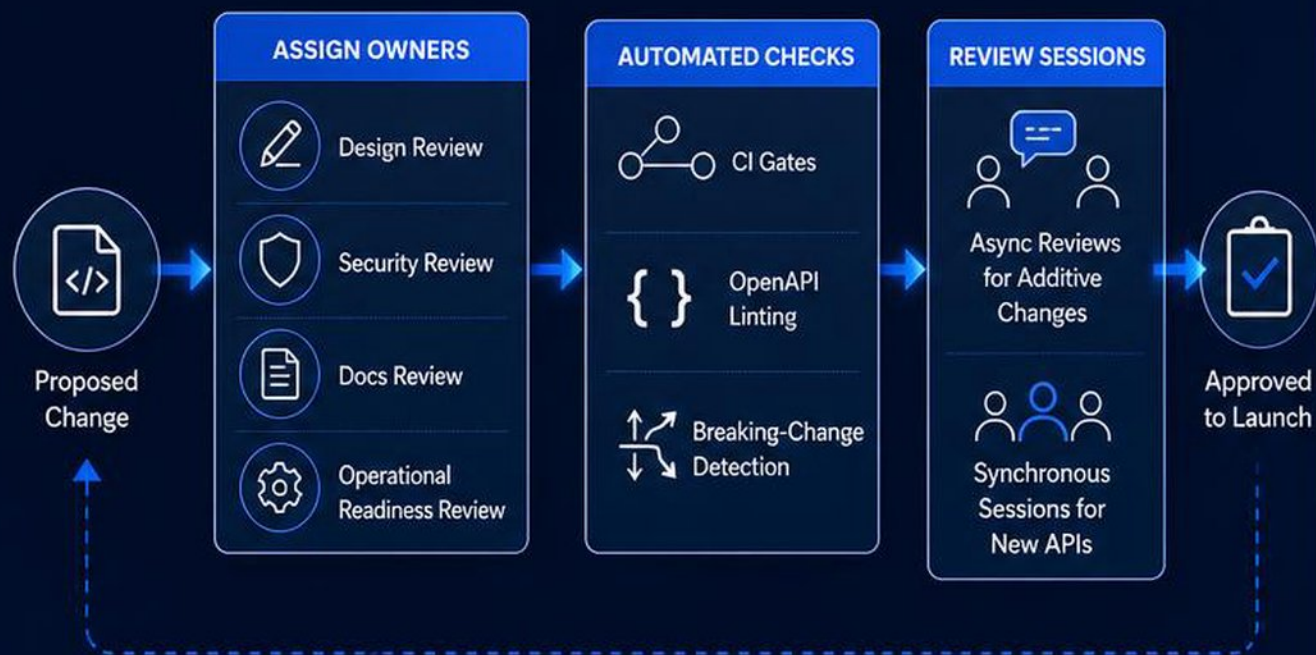
Authentication Guide

This API uses Bearer token authentication.
Include your token in the Authorization header
of every request.



Authentication
Bearer <token>

Pre-Launch Review Workflow for Teams



Assign owners for design, security, docs, and operational readiness review



Automate checks with CI gates, OpenAPI linting, and breaking-change detection



Async reviews for additive changes; synchronous sessions for new APIs



Combine automated checks with human review for semantic decisions

Post-Release Monitoring and Feedback Loops

- - Monitor adoption, error rates, latency, and support tickets
- - Track deprecated endpoint usage and consumer migration
- - Feed support tickets and incident reviews into the checklist
- - Measure registry coverage, lint pass rate, and escaped defects



Deprecation Planning and Consumer Communication



- Define deprecation windows by API type: internal, partner, public



- Use Sunset and Deprecation headers for machine-readable signals



- Publish migration guides with concrete before-and-after examples



- Reach known consumers directly using usage telemetry, not just email

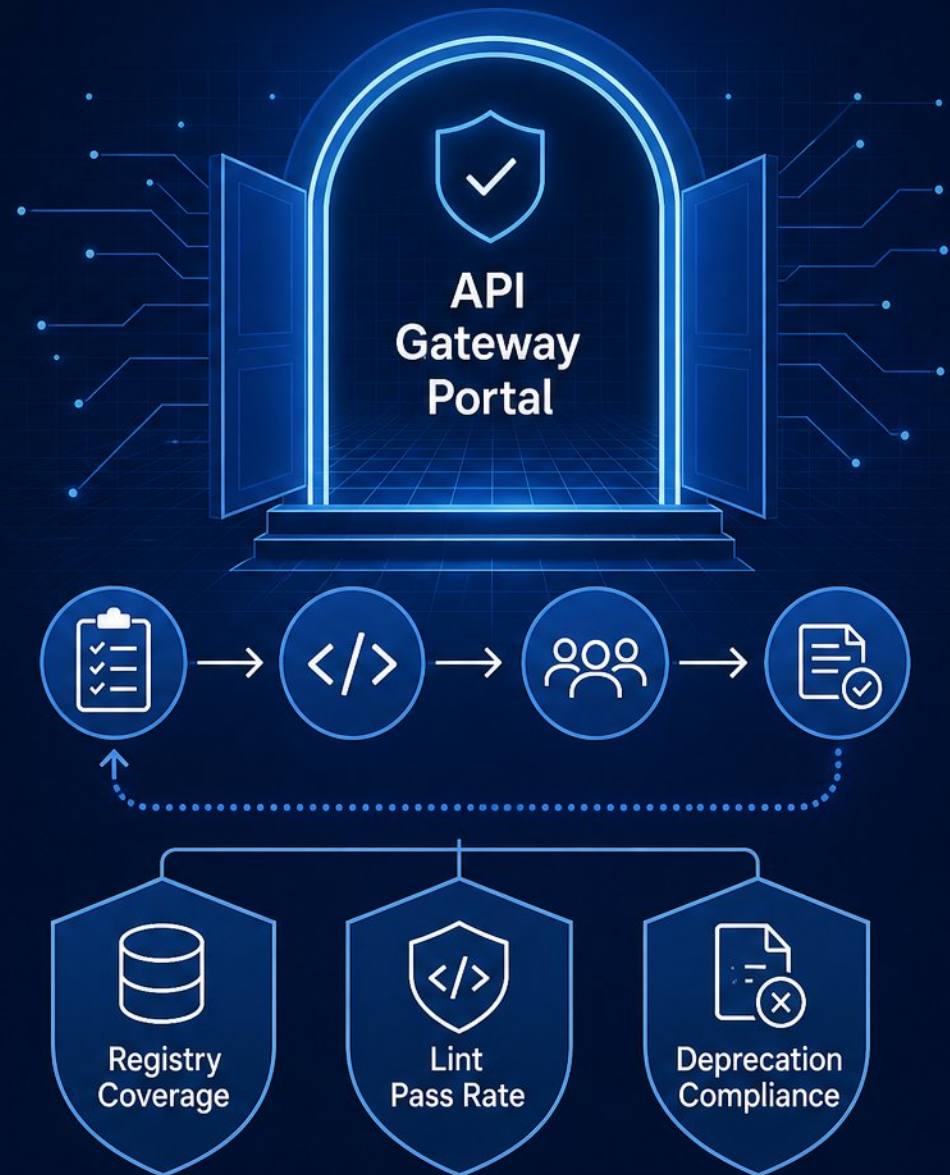


- Return 410 Gone with actionable migration guidance after sunset



Putting the Checklist into Practice

-  - Turn the review checklist into an operational workflow with clear ownership
-  - Combine automated CI gates with human review for semantic decisions
-  - Start with high-impact gates, then expand coverage as maturity grows
-  - Use post-incident reviews to add or refine checklist items
-  - Measure governance with registry coverage, lint pass rate, and deprecation compliance



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/188a1a17/public