Jentic

You May Have OpenAPI, But Is It AI-Ready?

Date:
11th Dec, 2025

Presented by:
Frank Kilcommins

I’m Frank Kilcommins

› Head of Enterprise Architecture @ Jentic

› Engineer and Architect (❤ APIs & Developer Experience)

› Governance Board member @ OpenAPI Initiative (OAI)

› Co-Author / Maintainer of the Arazzo Specification

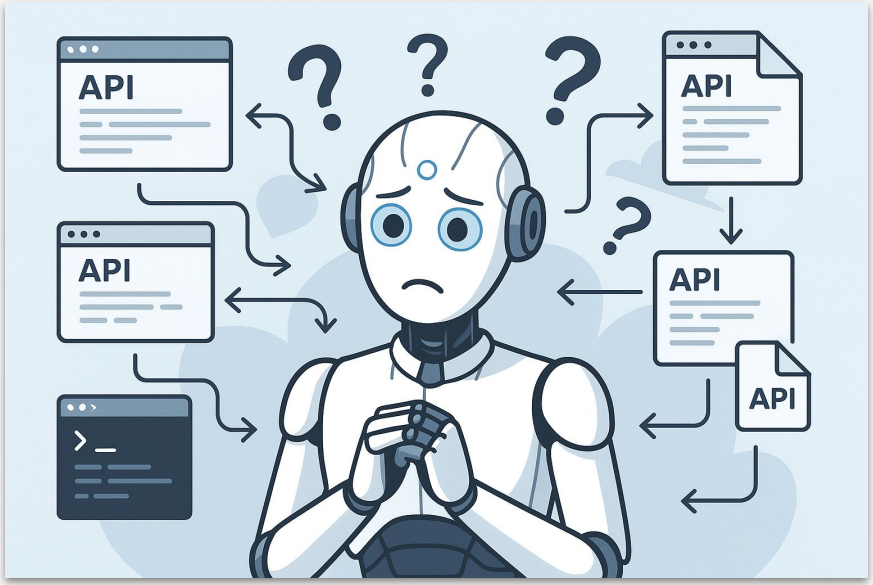
Connect:

@fkilcommins.bsky.social

@frank-kilcommins

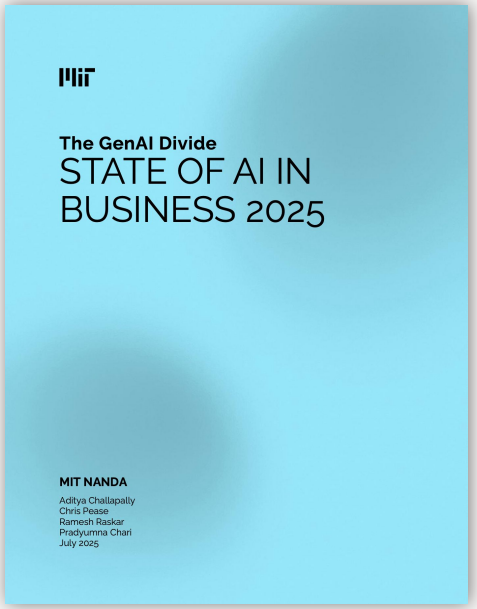
frank@jentic.com

Most APIs were designed for Humans



... (most) APIs are intended for developers, but not to be *interpreted, reasoned about, or safely executed* by AI systems

AI Initiatives Have Mixed Success



Aditya Challapally, Chris Pease, Ramesh Raskar, and Pradyumna Chari, "The GenAI Divide: State of AI in Business 2025", MIT NANDA, July 2025

Alex Singla, Alexander Sukharevsky, Lareina Yee, and Michael Chui, "The state of AI in 2025: Agents, innovation, and transformation", McKinsey, November 2025

AI Integration Is the New Moat

Tim O'Reilly "AI Integration Is the New Moat",O'REILLY, Oct 2025
https://www.oreilly.com/radar/integration-is-the-new-moat/

Jentic

But We Have OpenAPI! Isn't That Enough?

*... you can have **valid OpenAPI** that is **semantically useless** to both humans and machines ...*

Humans Tolerate, Machines Cannot

Human Interpretability:

- Vague descriptions
- Inconsistent naming
- Wrong servers
- Missing authentication info
- Missing examples

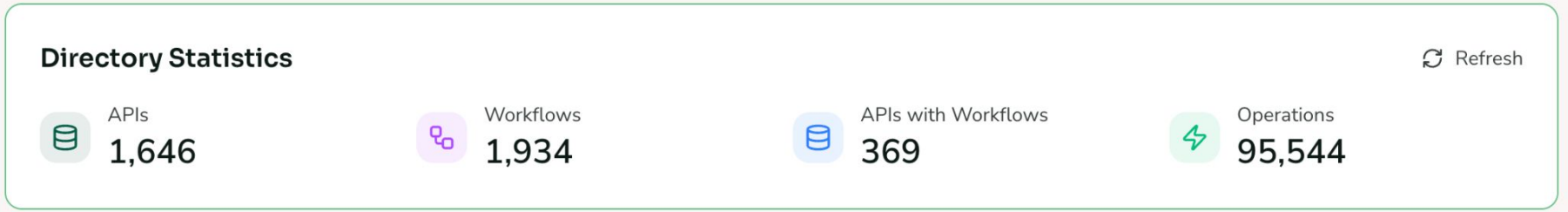
Humans Tolerate, Machines Cannot

Human Interpretability:

- Vague descriptions
- Inconsistent naming
- Wrong servers
- Missing authentications info
- Missing examples

Machine Interpretability:

What We Learned Across 1500+ Public APIs



See https://github.com/jentic/jentic-public-apis

Findings from 1500+ Public APIs

- Invalid or unparsable OpenAPI Docs
- Missing path parameters / broken `$ref` chains
- Invalid examples

- Examples missing
- Inconsistent naming + unclear verb usage
- Vague or generic descriptions

- No servers or wrong envs
- Auth info missing (outside OpenAPI)
- Stale / outdated OpenAPI Documents

Findings from 1500+ Public APIs

Invalid OpenAPI

- Missing path parameters
- broken `$ref` chains
- Invalid examples

Security vulnerabilities

- No servers defined
- Wrong environments
- Auth info missing
- Stale or outdated OpenAPI Documents

Missing semantics

- Examples missing
- Inconsistent naming
- Unclear verb usage
- Vague or generic descriptions

Typical Example

A valid but confusing path item

```
POST /users/{id}/update
summary: "Performs the update operation"
```

Agent interpretation:

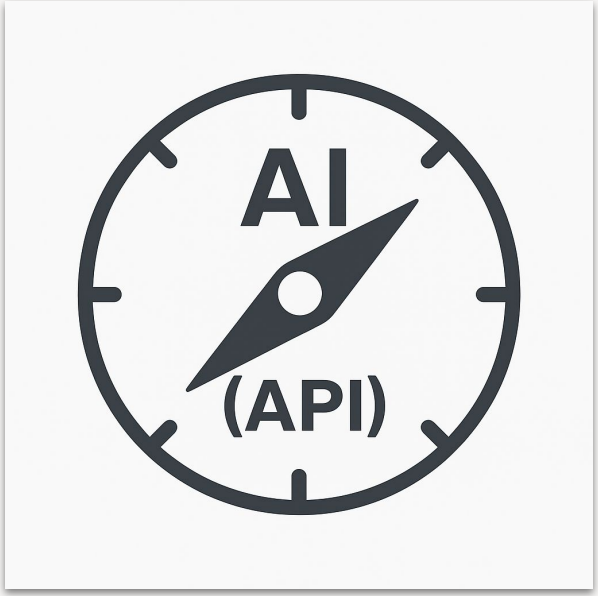
- It might *create* a user
- It might *replace* a user
- It might *partially update* a user
- It might *patch* only some fields

What AI Systems Actually Need

Agents need APIs to be:

- **Interpretable**: Intent must be explicit
- **Composable**: Clear inputs / outputs
- **Safe**: Auth & safety signals machine readable
- **Discoverable**: able to find and classify API
- **Predictable**: idempotency, stable responses
- **Resilient:** Clear signals for success and failure

You Can’t Improve What You Can’t Measure



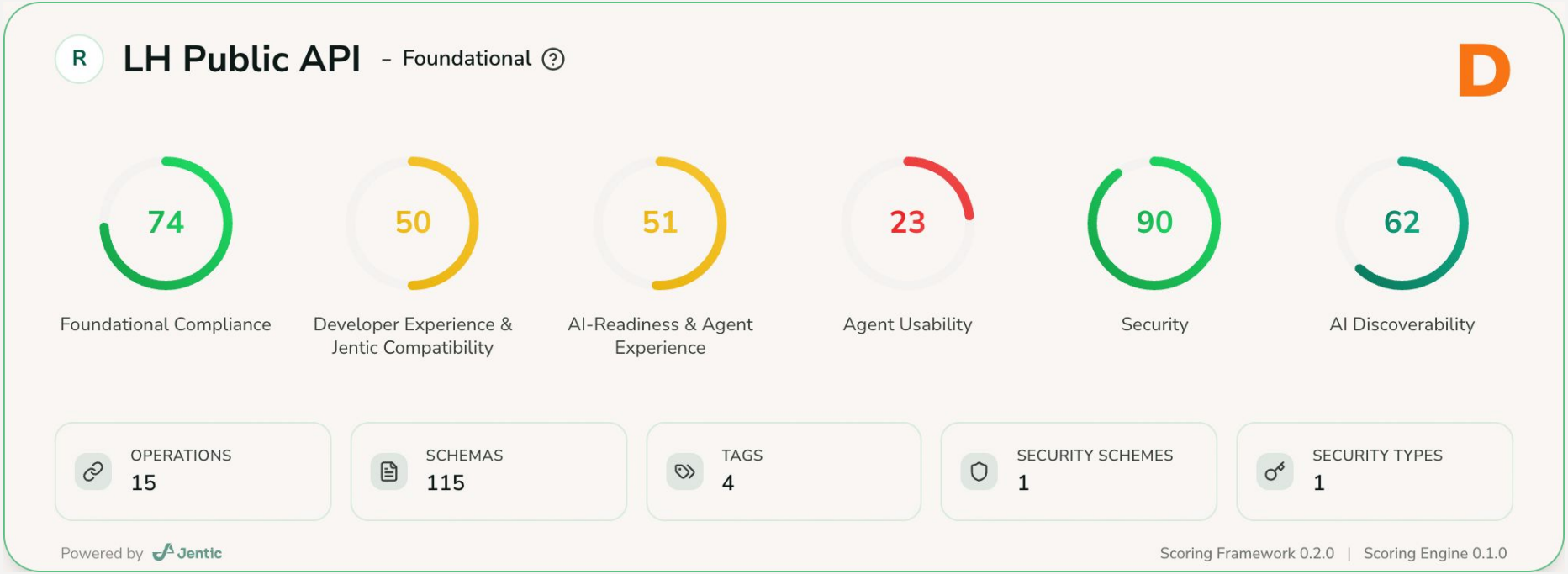
AI-readiness starts with understanding your current API landscape

API AI-Readiness Framework

Evaluates APIs across six human & AI-readiness dimensions:

- Foundational Compliance
- Developer Experience (including tooling compatibility)
- Agent Experience (AI-Readiness)
- Agent Usability
- Security
- AI Discoverability

AI-Readiness Scorecard



Jentic

See https://jentic.com/scorecard

Foundational Compliance

Evaluates if an API is structurally valid, standards-compliance, and parseable by tooling. The following signals comprise the dimension:

Signal	Description	Normalisation
`Spec validity`	Checks whether the API parses as a valid OpenAPI	binary
`Resolution Completeness`	What proportion of $ref references successfully resolve	coverage
`Lint Results`	Aggregated quality score from linter diagnostics, weighted by severity.	Inverse weighted categorical
`Structural Integrity`	Measures whether the API's underlying data model is coherent enough for automated reasoning (semantic or logical defects that impact reliable interpretation)	Logarithmic dampening

```
structural_integrity = max(0, 1 - (log10(1 + structural_issues / log10(1 + structural_issue_threshold)))
```

Foundational Compliance

Where:

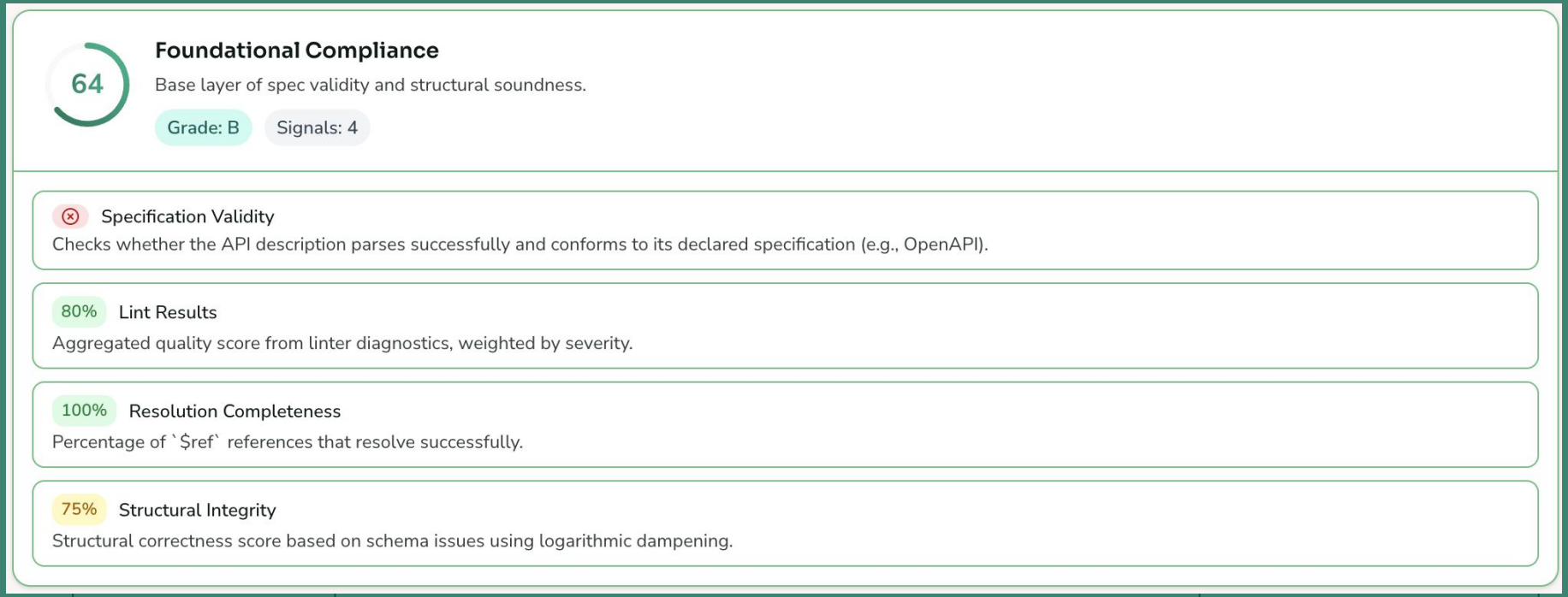
- `issues` is the total count of structural defects detected.
- `structural_issue_threshold` represents the point where structural reliability collapses. Once an API has more than ~15 schema-breaking or integrity flaws, automated interpretation is no longer trustworthy.

The formula yields a smooth decay curve, prevents early collapse, but penalises structural issues more heavily than cosmetic issues.

A *structural issue* MUST be recorded when any of the following occur:

Category	Examples
Invalid model shape	`type: object` but no `properties` defined; objects with additionalProperties: false but empty
Contradictory typing	`type: string` + `format: int32` ; arrays using incompatible items definitions
Impossible constraints	`minimum > maximum` ; exclusiveMinimum > exclusiveMaximum; enum values violating type, and contradictory schema constructs
Broken polymorphism	`oneOf` / `anyOf` / `allOf` inconsistent; missing or invalid discriminator; unreachable or contradictory sub-schemas
Response/request undefined	`requestBody: {}` ; missing schemas; empty content definitions
Non-evaluable example	Examples that are invalid JSON, violate the declared schema, or contradict field constraints
Unresolvable or circular schema structures	Schemas that reference non-existent fields; recursive references without a valid base schema

Foundational Compliance



Developer Experience

Assesses clarity, documentation quality, example coverage, and compatibility with developer tooling. The following signals comprise the dimension:

Signal	Description	Normalisation
`Example Density`	Measures coverage of examples across all eligible specification locations	coverage
`Example Validity`	Checks all examples for Schema Conformance (JSON + XML)	coverage
`Doc Clarity`	Quantifies linguistic clarity of summaries and descriptions. Basically, how easy is it to understand the intent, and the complexity of verbiage.	Min-max inverted
`Response Coverage`	Presence of meaningful success and error responses across full surface area	coverage

Developer Experience

51

Developer Experience & Jentic Compatibility

Clarity, completeness, and ingestion readiness for developers and tooling.

Grade: C- Signals: 4

8% Example Density

How richly the API is illustrated with examples.

98% Example Validity

Percentage of examples that conform to their schemas.

0% Response Coverage

Percentage of operations that define both success and error responses with realistic examples.

100% Tooling Readiness

Health of API ingestion, bundling, and resolution within Jentic pipelines.

AI-Readiness & Agent Experience

Evaluates where an API gives enough context for AI systems to understand its intent, constraints, and expected behaviours. The following signals comprise the dimension:

Signal	Description	Normalisation
`Summary Coverage`	Measures presence of concise summaries across eligible locations	coverage
`Description Coverage`	Measures presence of concise descriptions across eligible locations	coverage
`Type Specificity`	Rewards APIs that model values semantically, not just as loosely typed strings	weighted categorical
`Error Standardisation`	Favour structured error formats (RFC 9457/7807)	coverage
`Operation Identifier Quality`	evaluate presence and distinctiveness of operationId values	composite
`Policy Presence`	Promote inclusion of SLA/rate-limit/policy metadata	coverage
`AI Semantic Surface`	Bonus uplift for AI-oriented metadata	bonus multiplier

Agent Usability

Assesses how efficiently agents can navigate, combine, and execute API operations. Measures orchestration safety and agent ergonomics. The following signals comprise the dimension:

Signal	Description	Normalisation
`Complexity Comfort`	Measures document size, endpoint density, and schema complexity, penalised using a logistic curve	logistic shaping
`Distinctiveness`	Quantifies semantic separation between operations	inverse semantic similarity
`Navigation (pagination, hypermedia)`	Evaluates pagination and hypermedia affordances (HATEOAS, JSON:API, HAL)	composite
`Intent Legibility`	Evaluates verb-object semantic clarity	similarity (LLM assisted)
`Safety`	Evaluates idempotency & sensitive operation protection	heuristic penalty
`Tool Calling Alignment`	Represents alignment with LLM tool-calling expectations	coverage

Security

Assesses if your API protects data, enforces access controls, and follows secure-by-design practices. The following signals comprise the dimension:

Signal	Description	Normalisation
`Auth Coverage`	Evaluates whether authentication is correctly applied to sensitive or modifying operations, using intent-aware heuristics	heuristic penalty
`Auth Strength`	Measures robustness and correctness of authentication mechanisms declared by the API using normative scores based on IANA auth-schemes, OAuth2,, OIDC, API Key placement, and mutual TLS	weighted categorical
`Transport Security`	Requires HTTPS for externally exposed hosts	binary
`Secret Hygiene`	Detect and penalise hardcoded credentials	binary
`Sensitive Handling`	Evaluates the protection of PII/sensitive fields	coverage
`OWASP Posture`	Reflect severity-weighted risk findings	severity weighted

```
# auth_strength = average_strength_of_security_schemes
auth_strength = safe_divide(sum(strength_scores), count(schemes))
```

The following table outlines the `auth_strength` scoring weights:

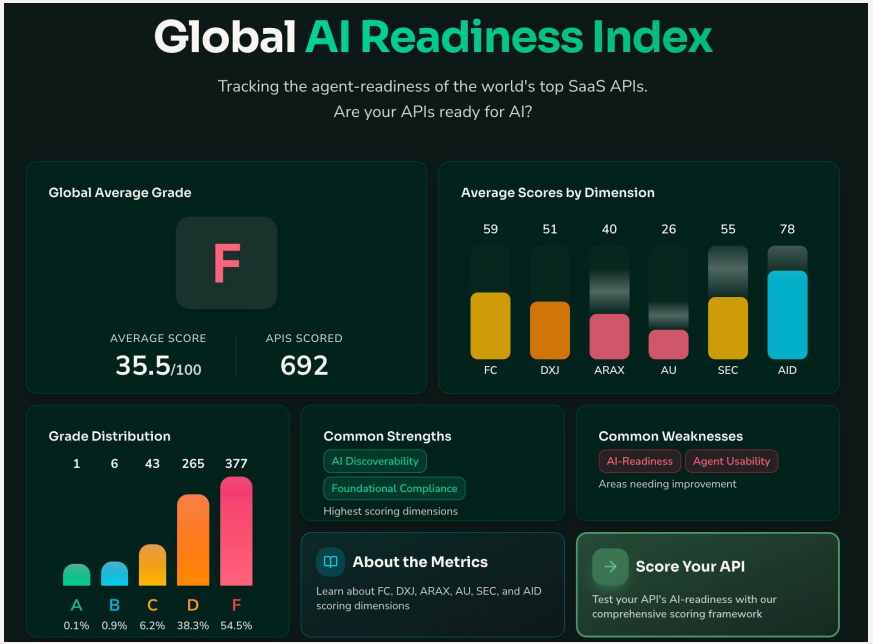
Scheme Type	Description	Example	Strength	Rationale
`none`	No authentication mechanism	no `security:` block	`0.00`	Unsafe for sensitive APIs; permitted only when `sensitive_ops_expected = 0`.
`http / basic`	Base64 user:pass	`scheme: basic`	`0.10`	Plaintext credentials; easily leaked (RFC7617).
`http / oauth`	OAuth 1.0	`scheme: oauth`	`0.20`	Deprecated; insecure signature model (RFC5849).
`http / digest`	Digest Access Auth	`scheme: digest`	`0.20`	Outdated; limited protection (RFC7616).
`apiKey (query)`	API key in query string	`in: query`	`0.15`	Very high leakage risk (logs, proxies, URLs).
`apiKey (header/cookie)`	API key in header or cookie	`in: header`	`0.50`	Moderate security; lacks identity, scoping, or rotation controls.
`http / scram-sha-1`	SCRAM with SHA-1	`scheme: scram-sha-1`	`0.25`	Uses deprecated SHA-1 hashing (RFC7804).
`http / negotiate`	Kerberos/NTLM	`scheme: negotiate`	`0.35`	Legacy; violates HTTP semantics (RFC4559).
`http / bearer (opaque)`	Opaque bearer token	`scheme: bearer`	`0.60`	Security depends entirely on token distribution (RFC6750).

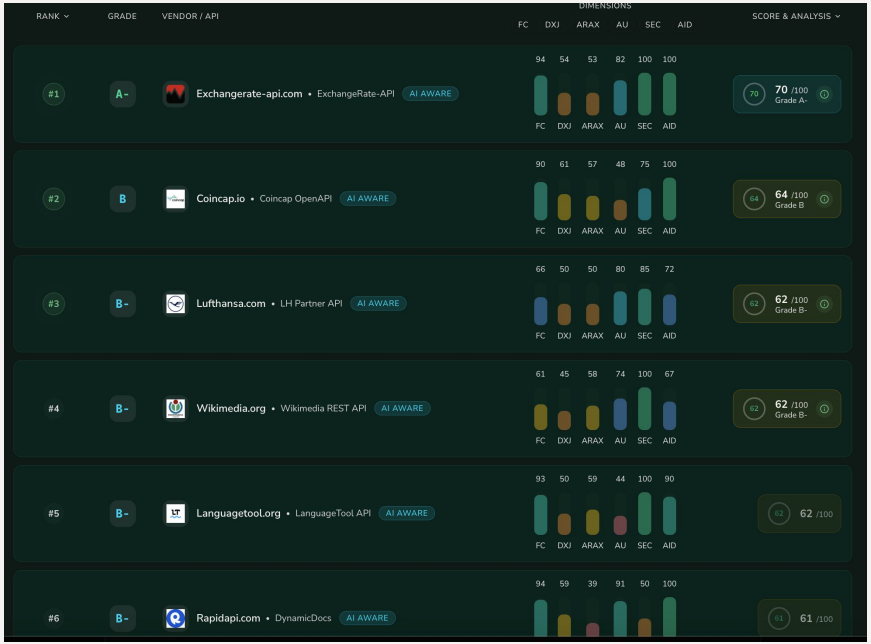
AI Discoverability

Evaluates how easily AI systems can locate, classify, and route to the API across registries, workflows, and knowledge bases. The following signals comprise the dimension:

Signal	Description	Normalisation
`Descriptive Richness`	Evaluates the semantic value of textual descriptions within an API description.	coverage
`Intent Phrasing`	Evaluates verb-object clarity of summaries and descriptions (Similar to Intent_legibility)	semantic similarity (LLM assisted)
`Workflow Context`	Evaluates indicators to workflow specifics including Arazzo/MCP/workflow references. Also analyses existence of callbacks etc.	coverage
`Registry Signals`	Detects metadata related to API gateways, llms.txt, APIs.json, MCP registries, externalDocs links to Dev portals.	coverage
`Domain Tagging`	Detect domain/taxonomy classification through the use of tags	coverage

Landscape Readiness Overview





Jentic

Coming soon at: https://jentic.com/leaderboard

Making OpenAPI Descriptions AI-Ready

Readable summaries & descriptions

Semantically rich schemas

Correct HTTP Methods

Full use of “Examples”

Use RFC 9457 Problem Details of Errors

Appropriate HTTP status code usage

Defined Servers & Environments

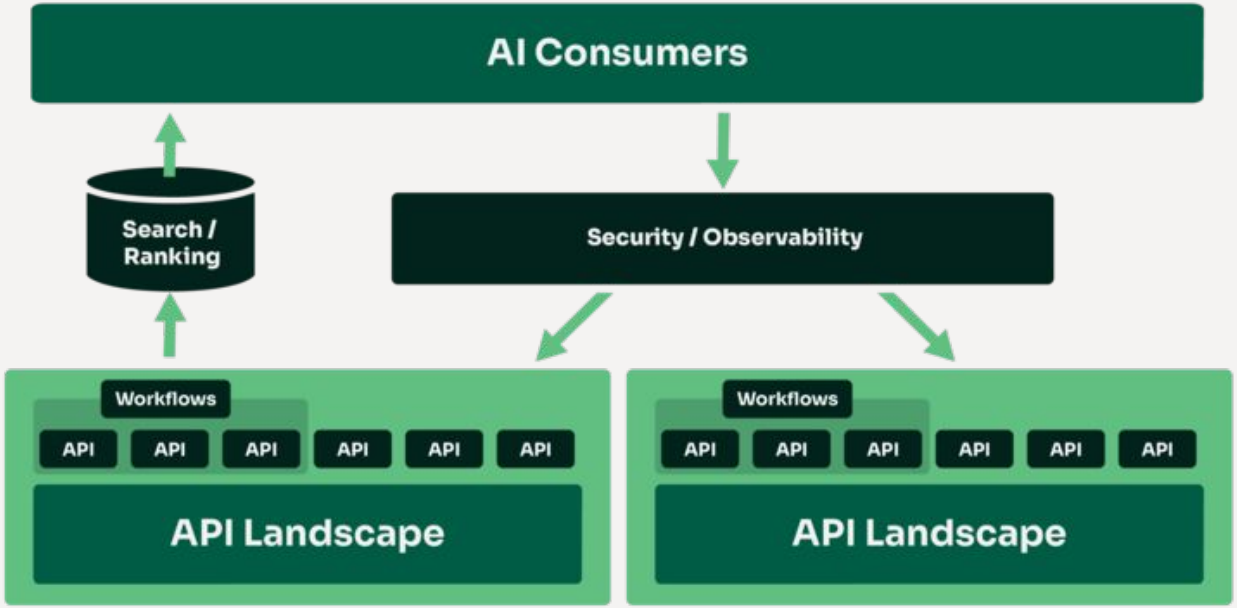
Codified security & SLA details

Prompt hints and AI-metadata extensions

Codify Intent-based use cases and workflows

Jentic

Entering the “Context Stage” of APIs



“Endpoints != Use-cases. Think about the workflows” - Adam DuVander

The Future: Machine-First API Design

OpenAPI 3.3:

- Encourage formal semantics (JSON-LD perhaps)
- Improve OAuth / OIDC support
- Improved Header support

Arazzo 2.0:

- gRPC, MCP, GraphQL step support
- Human-in-loop support
- Agent-in-loop support
- Transformer / function support
- Loops

Get your AI-Readiness score: jentic.com/scorecard

Thank you!

Connect:

@fkilcommins.bsky.social

@frank-kilcommins

frank@jentic.com

Jentic