

Workshop “Pre-Flight” Check



Requirements:

- Node.js 20 LTS or newer
- Docker installed & running
- Wifi Connected

Repo with steps and Exercises: <https://github.com/frankkilcommins/APIOps-AI-Ready-APIs>

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

Date:

3rd June, 2026

Presented by:

Frank Kilcommins

Workshop “Pre-Flight” Check



Requirements:

- Node.js 20 LTS or newer
- Docker installed & running
- Wifi Connected

Repo with steps and Exercises: <https://github.com/frankkilcommins/APIOps-AI-Ready-APIs>

I'm Frank Kilcommins

- › Head of Enterprise Architecture @ Jentic
- › Engineer and Architect (❤️ APIs & Developer/Agent Experience)
- › Governance Board member @ OpenAPI Initiative (OAI)
- › Co-Author / Maintainer of the Arazzo Specification

Connect:



@fkilcommins.bsky.social



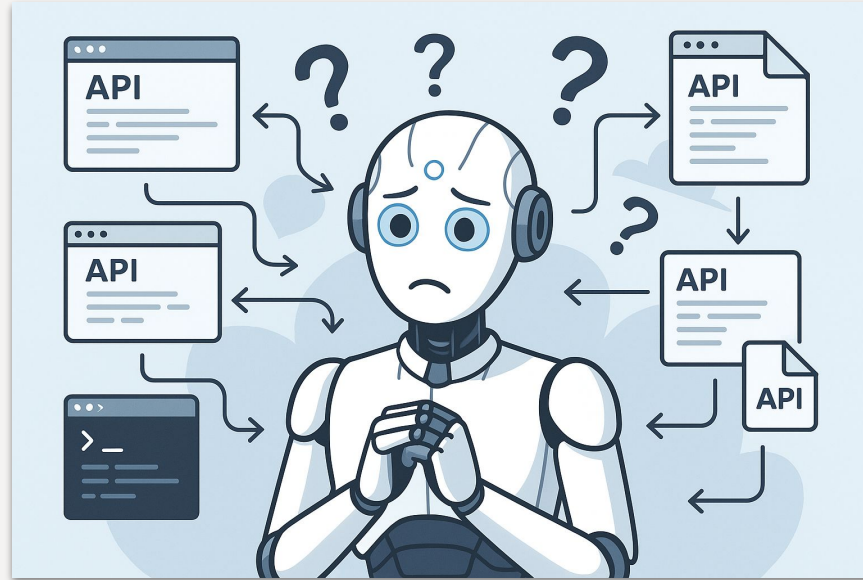
@frank-kilcommins



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/>

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 authentication info
- Missing examples

Machine Interpretability:



What We Learned Across 6000+ Public APIs

Directory Statistics

 Refresh

 APIs
6,358

 Workflows
2,000
(across 381 Arazzo Documents)

 APIs with Workflows
381

 Operations
220,491

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

Findings from 6000+ 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

Typical Example

```
A valid but confusing path item  
POST /users/{id}/update  
summary: "Performs the update operation"
```

Agent interpretation:

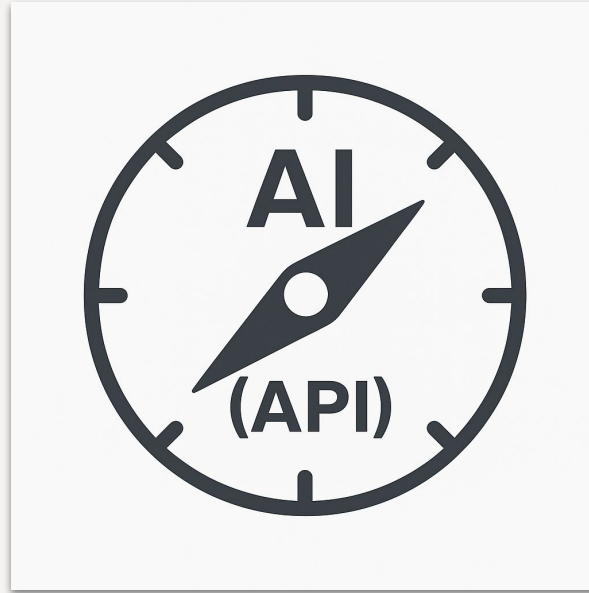
- It might *create* a user
- It might *replace* a user
- It might *partially update* a user
- It might *patch* only some fields

What AI Systems Actually Need

Agents need APIs to be:

- **Discoverable:** able to find and classify API
- **Interpretable:** Intent must be explicit
- **Composable:** Clear inputs / outputs
- **Safe:** Auth & safety signals machine readable
- **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

See <https://github.com/jentic/api-ai-readiness-framework>

API AI-Readiness Scorecard

R LH Partner API - Foundational ?

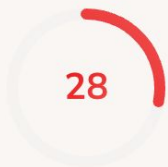
C-



Foundational Compliance



Developer Experience &
Jentic Compatibility



AI-Readiness & Agent
Experience



Agent Usability



Security



AI Discoverability


OPERATIONS
16

SCHEMAS
143

TAGS
6

SECURITY SCHEMES
1

SECURITY TYPES
1

Powered by  Jentic

Scoring Framework 0.2.0 | Scoring Engine 0.3.8

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	<code>type: object</code> but no <code>properties</code> defined; objects with <code>additionalProperties: false</code> but empty
Contradictory typing	<code>type: string + format: int32</code> ; arrays using incompatible items definitions
Impossible constraints	<code>minimum > maximum</code> ; <code>exclusiveMinimum > exclusiveMaximum</code> ; enum values violating type, and contradictory schema constructs
Broken polymorphism	<code>oneOf / anyOf / allOf</code> inconsistent; missing or invalid discriminator; unreachable or contradictory sub-schemas
Response/request undefined	<code>requestBody: {}</code> ; 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

72

Foundational Compliance

Base layer of spec validity and structural soundness.

Grade: A-

Signals: 4

⊗ Specification Validity

Checks whether the API description parses successfully and conforms to its declared specification (e.g., OpenAPI).

⊗ 14

88% Lint Results

Aggregated quality score from linter diagnostics, weighted by severity.

⊗ 14

⚠ 192

ℹ 55

100% Resolution Completeness

Percentage of ``$ref`` references that resolve successfully.

ℹ 2

100% Structural Integrity

Structural correctness score based on schema issues using logarithmic dampening.

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

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

Asses
secu

Signal

Auth C

Auth S

Transp

Secret

Sensit

OWASP



rical

d

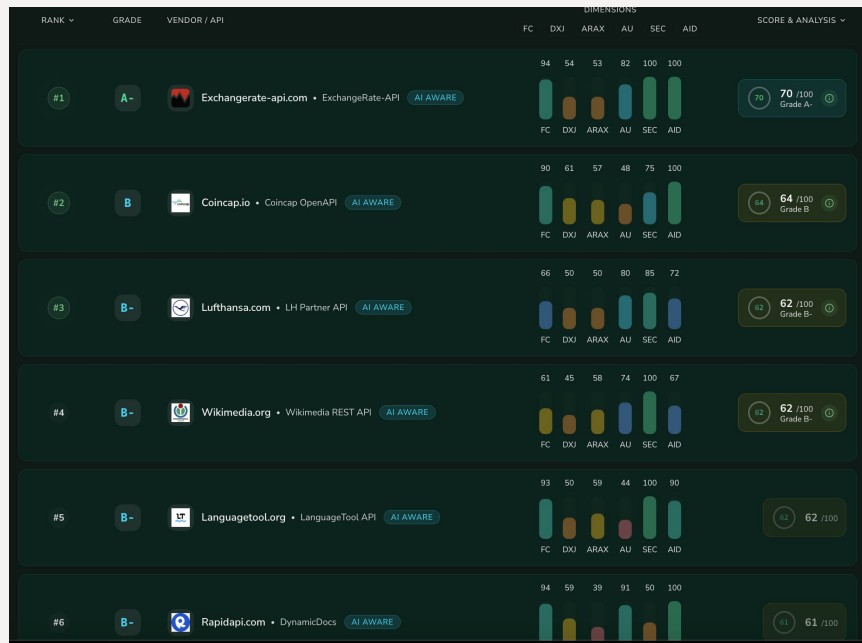
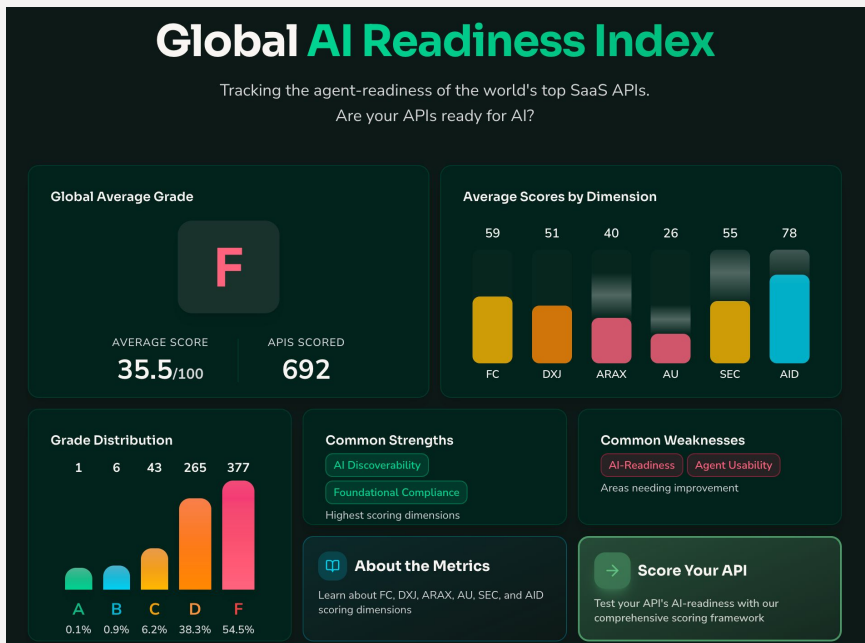
erved

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



Readable summaries & descriptions

Semantically rich schemas

Correct HTTP Methods

Full use of "Examples"

Use RFC 9457 Problem Details of Errors

Making OpenAPI Descriptions AI-Ready

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



Let's Score some APIs!



Hands-On Time: Pre-Flight Reminder



Requirements:

- ❑ Node.js 20 LTS or newer
- ❑ Docker installed & running
- ❑ Wifi Connected

*Can't install? Use the browser scorer at
<https://jentic.com/scorecard>*

Everything Needed is in the Repo: <https://github.com/frankkilcommins/APIOps-AI-Ready-APIs>

Exercises Overview



- **Install & verify:** Score the Petstore - confirms your setup works
- **Score local files:** Run all 5 samples - find the spread
- **Jentic Public APIs:** Pick your favourite from 6000+ and score it
- **Your own API:** Bring your OpenAPI - see where you stand
- **Improve a dimension:** Target 1/2 signals for a dimension, improve score by 10+

Everything Needed is in the Repo: <https://github.com/frankkilcommins/APIOps-AI-Ready-APIs>

What's Coming Next

- **Direct API Access:** Integrate scoring into your own portals and platforms
- **GitHub Action:** AI-Readiness gates in your CI/CD pipelines
- **MCP Tool + Skill:** Scoring from inside your AI assistant via natural language
- **Improvements Skill:** Auto-improvements of your APIs presented as Overlays

Final Actions

- ★ Star the repo: <https://github.com/jentic/jentic-api-scorecard>
- 🔄 Share the CLI with **your team**
- ✅ Start **improving** your API's **DX** and **AX** experiences today

*The gap between valid and AI-ready APIs is measurable.
Now you have the tool to close it!*

Thank you!

Connect:



@fkilcommins.bsky.social



@frank-kilcommins



frank@jentic.com

