



# Application Programming Interfaces (APIs)

Adopting the NIAID Blueprint

**SDSC**  
SAN DIEGO SUPERCOMPUTER CENTER

**GO FAIR US** **GO FAIR**  
oundation

**NCAR** | NATIONAL CENTER FOR  
ATMOSPHERIC RESEARCH

Knowledge Motifs LLC  
*Mapping sensible data relationships*

# Hosted By



## [Office of Data Science and Emerging Technologies \(ODSET\)](#)

ODSET's mission is to collaborate strategically across divisions to advance the NIAID mission through data infrastructure, data innovation and data-driven solutions.



Based at the San Diego Supercomputer Center (SDSC), [GO FAIR US \(GFU\)](#) fosters a collaborative community where FAIR approaches can be shared and advanced. GFU promotes practical implementation of FAIR across research domains.

## [NIAID Data Landscaping and FAIRification Project](#)

Since March 2025, ODSET and GFU have partnered with NIAID-funded repositories to identify opportunities for increasing data discovery and reuse through Blueprint implementation. This collaboration bridges technical standards with real-world research needs.

# The NIAID Blueprint lowers barriers to data access and reuse through a shared framework.

## *5 Key Categories Outlined in the Blueprint*

- 1. NIAID Minimal Metadata Schema:** a minimal set of NIAID-relevant metadata elements and a standard schema with default formats.
- 2. Persistent Identifiers:** persistent identifiers for relevant metadata elements.
- 3. APIs for Metadata:** exposure of metadata elements through a standard, open API.
- 4. Citation Guidance:** explicit guidance to the research community on how digital objects should be cited.
- 5. Outreach and Training:** establishment of a point of contact (at major repositories).



# What You'll Learn from this Webinar

## **The relationship between web standards and APIs**

A review of using core web technologies—such as RESTful resource-oriented URLs ("Cool URIs"), standard HTTP methods (GET, POST), and consistent status codes—to create stable and scalable data services

## **APIs and FAIR Principles**

Leveraging standards and web architecture to ensure APIs can be expressed aligning to the FAIR principles of the NIAID Blueprint

## **APIs and machine readable and AI ready goals**

This focuses on documenting APIs using the OpenAPI/Swagger specification and adopting emerging standards like the Model Context Protocol (MCP) to enable AI agents and coding tools to interact seamlessly with research data.

### **Audience:**

Attendees range from world experts to beginners and everything in between.

Focus on key points.

Please add questions to the Q&A tool. Longer topics will shape future materials.

# Play along!

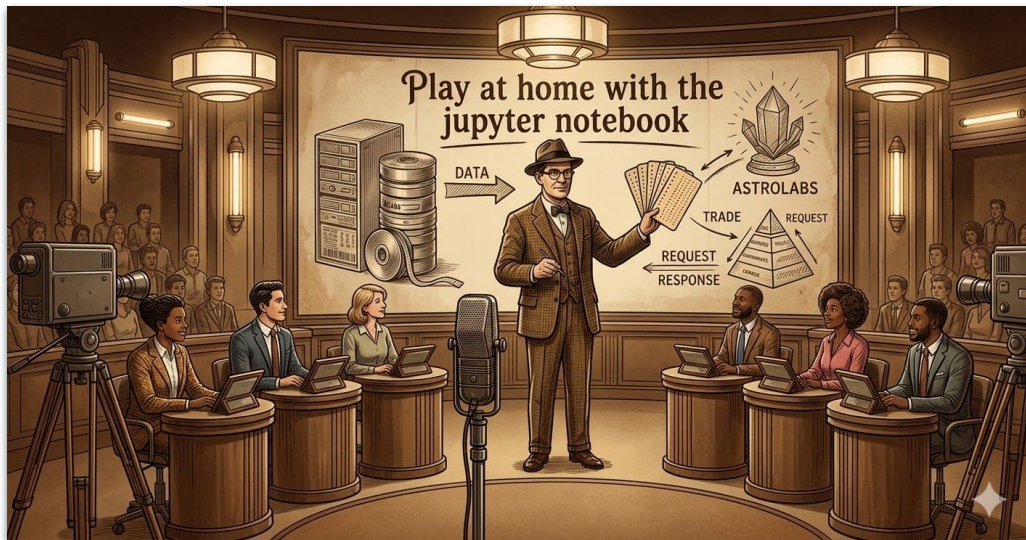
Enjoy the **"code edition"** of this talk in Colab or download and run it yourself from GitHub.

[Colab Link](#) / [GitHub Link](#)

Presentation, notebook, example code and more details at:

<https://github.com/go-fair-us/apireference>  
(or use QR code to the right)

***Issue creation and pull requests always welcome from all!***



Images throughout this presentation created with Gemini.

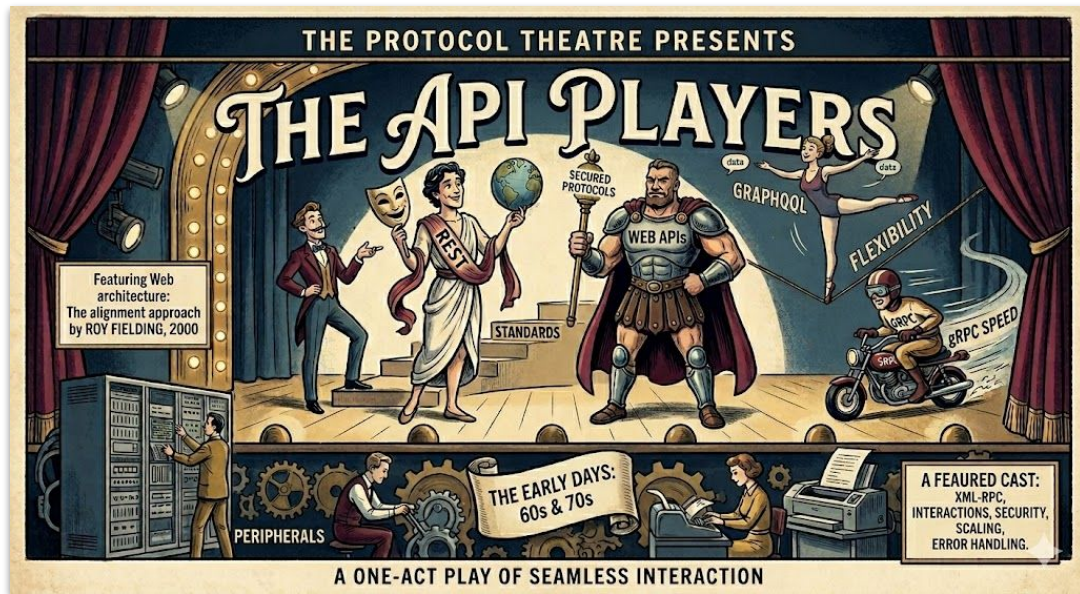




# API Backstory: Our cast of ~~characters~~ acronyms

## The *Application Programming Interface* (API)

- Started in the late 60s. Described a means to interact between systems, main frames for example, and peripherals.
- This presentation, API = mostly "Web APIs"
- REST, or Representational State Transfer, from the Roy Fielding's 2000 dissertation. *It is an approach to APIs aligned with web architectural standards*



Think of Web APIs as the different ways software applications "talk" to one another. Just as we use different languages or protocols (a formal document vs. say, this presentation), developers choose different API styles based what they need.

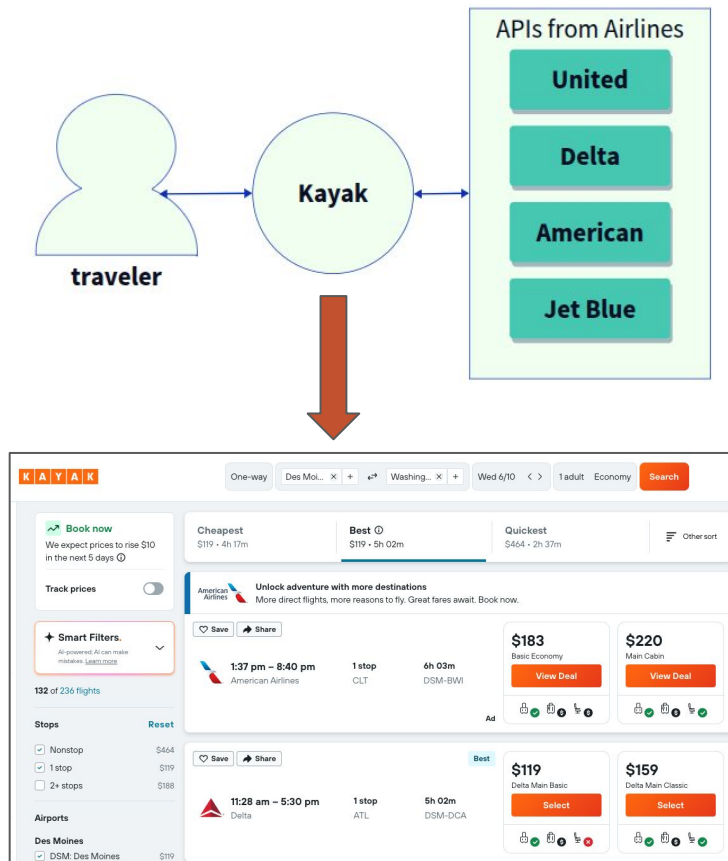
# APIs in Daily Life Example

## The Digital Middleman

Imagine you are the **Traveler**. You want to fly from Des Moines to Washington D.C., but you don't want to spend hours manually checking the individual websites of United, Delta, American, and JetBlue to compare prices.

This is where **Kayak** (the "Client" or interface) comes in. But Kayak doesn't have all the flight information, it needs a way to ask the airlines for their latest data of what is available.

**Kayak leverages APIs** from the airlines to assemble disparate data from different sources into a interface that provides you the various options based on your date, time of day and other travel preferences.

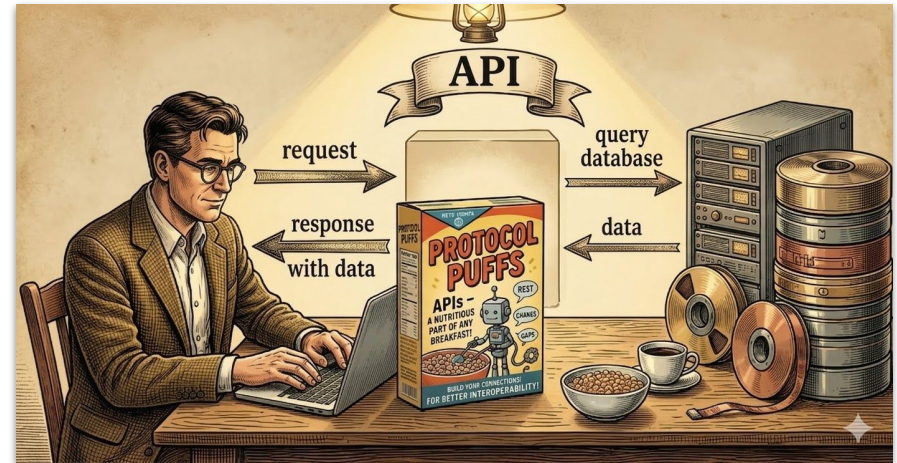


# APIs and the Blueprint

For standardized machine **access to metadata**, APIs should expose metadata elements... In general, an API should meet these minimum objectives:

- **Metadata Encoding:** API responses should return metadata encoded in JSON-LD... This would follow the types and properties guidance in the minimal metadata specification above.
- **IRI (URL) Structure:** API endpoints should be designed as resource-oriented IRIs (e.g., /datasets/{dataset\_id}), avoiding verbs and complex query parameters in the IRI structure\*. This ensures that the IRIs can function as persistent identifiers (IRIs) within the JSON-LD @id field, enabling seamless integration into knowledge graphs.
- **HTTP Method:** Metadata retrieval should be performed using the HTTP GET method.
- **Documentation:** Should adhere to OpenAPI/Swagger specifications for machine-readability and ease of use.

*Though this guidance is focused on metadata, many elements apply to any resource request.*



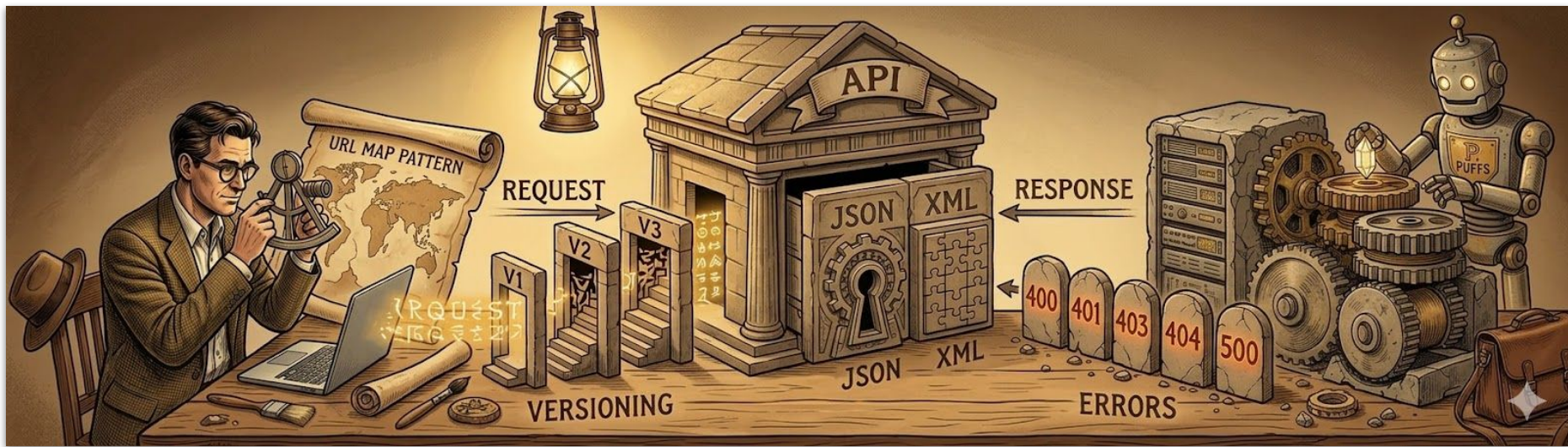
*An example of API-exposed metadata formatted to meet these objectives is given in Supplemental Table 7.*

<https://www.niaid.nih.gov/research/data-blueprint>

Mayer et al. (2025) DOI: [10.5281/zenodo.17161561](https://doi.org/10.5281/zenodo.17161561)







## Exploring the Properties of The Request–Response

# The Web, APIs and the *Request Response* Sequence

```
curl -request GET --url  
'https://import.org/data/query/api/search/study?term=influenza%20vaccine&fromRecord=0&pageSize=20&PreTag=%3Cem%3E&PostTag=%3C%2Fem%3E&format=json&sortFieldDirection=asc&conditionOrDisease=asthma%20COVID-19'
```

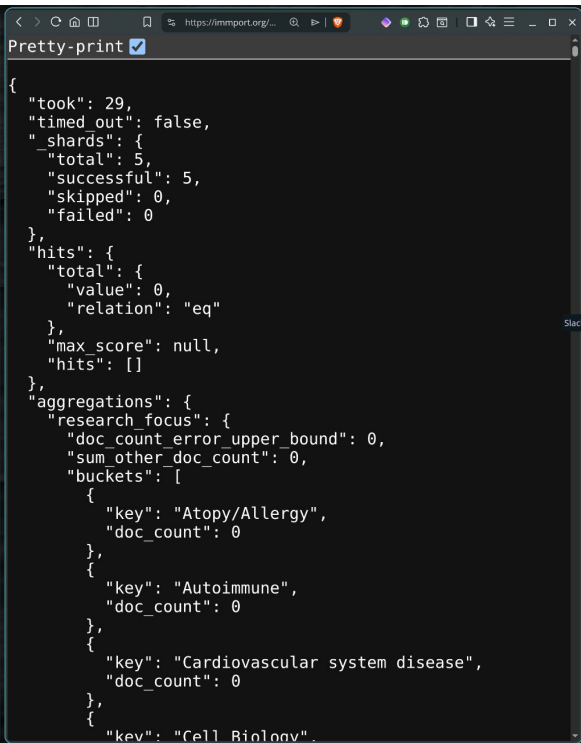
Same [URL](https://import.org/data/query/api/search/study?term=influenza%20vaccine&fromRecord=0&pageSize=20&PreTag=%3Cem%3E&PostTag=%3C%2Fem%3E&format=json&sortFieldDirection=asc&conditionOrDisease=asthma%20COVID-19), dropped into the browser

Interacting with APIs is typically expressed as a **requests and its response**.

Though the "curl" request looks opaque, in this case we can simply drop it into a browser and get the same response.

***APIs are mostly just URLs that return formatted data as a response, vs an HTML page***

```
{  
  "took": 31,  
  "timed_out": false,  
  "_shards": {  
    "total": 5,  
    "successful": 5,  
    "skipped": 0,  
    "failed": 0  
  },  
  "hits": {  
    "total": {  
      "value": 0,  
      "relation": "eq"  
    },  
    "max_score": null,  
    "hits": []  
  },  
  "aggregations": {  
    "research_focus": {  
      "doc_count_error_upper_bound": 0,  
      "sum_other_doc_count": 0,  
      "buckets": [  
        {  
          "key": "Atopy/Allergy",  
          "doc_count": 0  
        },  
        {  
          "key": "Autoimmune",  
          "doc_count": 0  
        }  
      ]  
    }  
  }  
}
```



https://import.org/...  
Pretty-print ☒

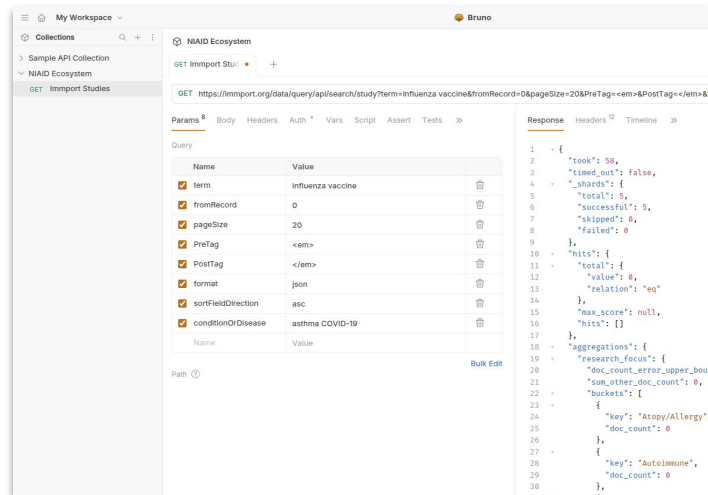
```
{  
  "took": 29,  
  "timed_out": false,  
  "_shards": {  
    "total": 5,  
    "successful": 5,  
    "skipped": 0,  
    "failed": 0  
  },  
  "hits": {  
    "total": {  
      "value": 0,  
      "relation": "eq"  
    },  
    "max_score": null,  
    "hits": []  
  },  
  "aggregations": {  
    "research_focus": {  
      "doc_count_error_upper_bound": 0,  
      "sum_other_doc_count": 0,  
      "buckets": [  
        {  
          "key": "Atopy/Allergy",  
          "doc_count": 0  
        },  
        {  
          "key": "Autoimmune",  
          "doc_count": 0  
        },  
        {  
          "key": "Cardiovascular system disease",  
          "doc_count": 0  
        },  
        {  
          "key": "Cell Biology",  
          "doc_count": 0  
        }  
      ]  
    }  
  }  
}
```

# The Web, APIs and the *Request Response* Sequence

We see this daily when we load a site  
URLs into a client we call a "browser".

We use METHODS like GET, POST and  
others (HEAD, PUT, DELETE, CONNECT, OPTIONS, TRACE).

Method	What it does	Real-world example
GET	Retrieves data	Viewing this presentation
POST	Sends/Creates data	Submitting a new "Sign Up" form to attend this webinar.



Example API clients include things  
like curl, [HTTPIe](#), PostMan, [Bruno](#) or  
custom code.

Think of these as like browsers for the  
API web

# Web Architecture

We can leverage the core web architecture to be an "APIs for Metadata records".

This is because URLs represent fetchable resources we can work with!

ImmPort example:

<https://www.dev.immport.org/shared/study/SDY2176/summary>

We can also fetch it via: [validator.schema.org](https://validator.schema.org/) to obtain the structured JSON-LD record.

**NOTE:** The JSON-LD is loading dynamically into the document object model via Javascript. This means a browser or headless browser is needed to execute that process.

The screenshot displays the ImmPort website interface. At the top, there's a navigation bar with links like 'Shared Data', 'Data Catalogs', and a search bar. Below the navigation bar, a banner for 'SDY2176: A safe and highly efficacious measles virus-based vaccine expressing SARS-CoV-2 stabilized prefusion spike' is visible. The main content area is divided into two tabs: 'Summary' and 'Dataset'. The 'Summary' tab is currently active, showing a detailed description of the study, including its title, DOI, and objectives. A red arrow points from the 'Summary' tab to the 'Dataset' tab, which is partially visible on the right side of the screen. The 'Dataset' tab shows a JSON-LD record for the study, with fields like 'id', 'type', 'name', and 'description'. The 'Dataset' tab also includes a 'Run test' button and a 'Dataset' section with a table of data.

ImmPort | Upload | Shared | Analysis | Resources | Data Management and Sharing Plan | Search www.immport.org | Data Catalogs

Have questions about ImmPort? Try out ImmPort's new documentation chatbot (beta).

SDY2176: A safe and highly efficacious measles virus-based vaccine expressing SARS-CoV-2 stabilized prefusion spike

Usage Metrics | Download Options

Pageviews: 214 | Downloads: 0 (Web: 0, API: 0) | Released: Feb 2023 (DR47), Updated: Oct 2025 (DR58)

Study Package (Web) | API | HL7 FHIR

Access 1 additional study descriptors for SDY2176 in SeroNet.

Summary | Design | Adverse Event | Assessment | Interventions | Medications | Substance | Demographics | Lab Tests | Mechanistic Assays | Study Files

Summary

Accession: SDY2176

Title: A safe and highly efficacious measles virus-based vaccine expressing SARS-CoV-2 stabilized prefusion spike

DOI: 10.21430/M39VJJKEMO

Brief Description: To develop a safe, efficacious, and durable SARS-CoV-2 vaccine, using a measles virus (MeV) vaccine strain as the backbone

Research Focus: Vaccine Response

Condition Studied: COVID-19

Start Date: [Redacted]

Detailed Description: The current plan is to develop a safe, efficacious, and durable SARS-CoV-2 vaccine, using a measles virus (MeV) vaccine strain as the backbone. The vaccine will be developed as a live-attenuated virus, and will be produced in a cell-based system. The vaccine will be tested in a phase I/IIa clinical trial, and will be compared to a placebo. The vaccine will be tested in a phase I/IIa clinical trial, and will be compared to a placebo. The vaccine will be tested in a phase I/IIa clinical trial, and will be compared to a placebo.

Objectives: Basic Research

Hypothesis: Not Applicable

Endpoints: SARS-CoV-2

Intervention: other

Agent: [Redacted]

Sex Included: Female, Other

Subjects: 19

Schema.org | Documentation | Schemas | Validate | About

https://www.dev.immport.org/shared/study/SDY2176/summary

Dataset

Dataset

id: https://www.immport.org/shared/study/SDY2176

type: [Redacted]

name: [Redacted]

description: [Redacted]

dataset: [Redacted]

alternatename: SDY2176

alternatename: A safe and highly efficacious measles virus-based vaccine expressing SARS-CoV-2 stabilized prefusion spike

alternatename: [Redacted]

conditions/offices: [Redacted]

description: To develop a safe, efficacious, and durable SARS-CoV-2 vaccine, using a measles virus (MeV) vaccine strain as the backbone

# Addressing Addresses



## Cool URIs

"URIs don't change: people change them."

<https://www.w3.org/Provider/Style/URI>

- From the W3C (web standards body)
- Stable, governed service URLs
  - Like GUPRIs for PIDs

## Cool URIs for the Semantic Web

<https://www.w3.org/TR/cooluris/>

- Dereferenceable
  - "Be on the web", "follow your nose"
  - distinguish web docs from real-world objects
- Hash URIs (e.g., [example.com/about#alice](https://www.w3.org/TR/cooluris/))
  - Leverage hash striping, used a lot in ontologies for example
- 303 URIs (e.g., [example.com/id/alice](https://www.w3.org/TR/cooluris/)): Redirect (303 See Other) to HTML/RDF; flexible for large/dynamic data.
- Use content negotiation

Example of what a typical GET API Call resemble:



Characteristic	Cool URI Example	Not Cool URI Example
Persistence: Avoids dates (unless they have meaning) or tech specifics, or names that might change.	.../people/alice	.../1999/alice.php
Simplicity: No query strings or scripts	.../books/123	.../cgi-bin/book.pl?id=123
Meaningfulness: Readable and descriptive	.../cities/paris	.../x7k9p2m
Unicode Support (IRI): Native characters over encoding	.../人名/alice	.../%E4%BA%BA%E5%90%8D/alice



# Caching

## Introduction to HTTP Caching

At its core, **caching** is the practice of storing a copy of a given resource (like a JSON response or an image) and serving it back when requested again. Instead of a request traveling all the way to the application server or database, it is intercepted by a "cache" layer that stays closer to the user, providing a shortcut for data retrieval.

- Caching cuts backend compute and database load
- Caching helps services scale to more users and AI/ML workloads without proportional cost increases.
- Faster response times

NOTE: This is about server side caching, not client side. In this approach caching is an architecture component.

### Better design = Better caching

- Use path for stable identifiers:  
/patients/123/reports/latest
- Put filters/sorting in query only when needed (and set Cache-Control wisely)
- Clean, persistent URLs
  - a. More cache hits
  - b. Faster APIs, less load

### Quick wins

- Avoid dynamic/temporary params in paths
- Use ETag/Last-Modified for conditional requests
- Set strong Cache-Control
  - a. max-age on stable resources

# APIs Need Versioning

API Versioning is the practice of managing multiple versions of an API simultaneously. It allows you to introduce modern features (e.g., v2.0) while keeping the original structure (v1.0) active for those who need it.

- Versioning enables safe updates without breaking researcher tools
- Provides stable URLs for reliable long-term citations and impact
- Ensures data reproducibility across years
- Reduces maintenance effort and support tickets dramatically
- Strengthens FAIR Reusability and machine-actionability
- Future-proofs services as science and data models evolve

## Types of Versioning

- URI/Path Versioning  
(e.g., <https://api.data.niaid.nih.gov/v1/query>)
  - Best for Public/scientific APIs
- Header-Based Versioning (e.g., **Accept: application/vnd.myapi.v2+json** or custom header like **X-API-Version: 2**)
- Query Parameter Versioning (e.g., **/api/resources?version=2**)
  - Can make caching harder
- Media Type / Content Negotiation

There are some very robust and tested approaches to versioning like in Stripe and GitHub. An open source approach to this model can be seen in: <https://docs.cadwyn.dev/>

# Formats Facilitate Correct Interpretation (Encoding)

As anywhere, incompatible formats make interoperability difficult. So standard formats returned by APIs is critical.

## Examples Standards

- **Community Format Examples**
  - a. FHIR (Fast Healthcare Interoperability Resources)
  - b. GA4GH (Global Alliance for Genomics & Health)
- **CSV + Semantically Expressed Data Model:**  
Even "good old" CSV is fine is out metadata can provide semantic context to the columns (see CVS on the Web, W3C)

**Easier Integration & Reduced "Data Wrangling":** By using common formats, scientists spend less time manually fixing errors or converting files and more time performing actual analysis.

Modern APIs use "content negotiation" to deliver the best format for the task. While a simple table (**CSV**) is popular for its ease of use, adding **JSON-LD** provides "semantic enhancement." This means the data isn't just a value; it's linked to a specific, globally defined concept.

- **The Power of PIDs (Persistent Identifiers):** PIDs ensure that data remains aligned and unambiguous over time. For example:
  - **In a standard CSV:** A column named temp could mean temperature, temporary status, or a template.
  - **In JSON-LD:** The value for temp points directly to a controlled vocabulary (e.g., "Body Temperature in Celsius"), removing all guesswork for the computer.

## Metadata and Knowledge Graphs

- **JSON-LD as RDF:** This format allows data to be treated as part of a **Knowledge Graph**. Because JSON-LD is representation of RDF (Resource Description Framework), it can plug directly into evolving global networks like the **Open Knowledge Network (OKN)**, connecting disparate research findings into a single, searchable web of data.

# APIs Should Use Standard Error Handling & Status Codes

## Understanding HTTP Status Codes

When an API client sends a request to a server, the server doesn't just send back data; it sends a **Status Code**. These three-digit numbers act as a universal shorthand, telling the client exactly how the request was handled without needing a long text explanation.

For researchers and developers, these codes are the foundation of **Linked Data**. They provide a machine-readable way to understand if a digital object (like a dataset or a paper) has moved, is missing, or is currently unavailable.

1xx	2xx	3xx	4xx	5xx
The request was received but there isn't a response yet	The request was received successfully and the server can return the requested data	The requested resource is in another location, so a separate call is necessary	The request can't be completed due to an issue on the client's end	The request can't be completed due to an issue on the server's end

**1xx** → Keep going / processing: rare, 102

**2xx** → Success! 200 (everyone's favorite)

**3xx** → Go somewhere else (redirect): 301, 303

**4xx** → Your fault (fix the request): 404, 406

**5xx** → Server's fault (try later or contact support): 500, 503

# Describe (Document) Your API

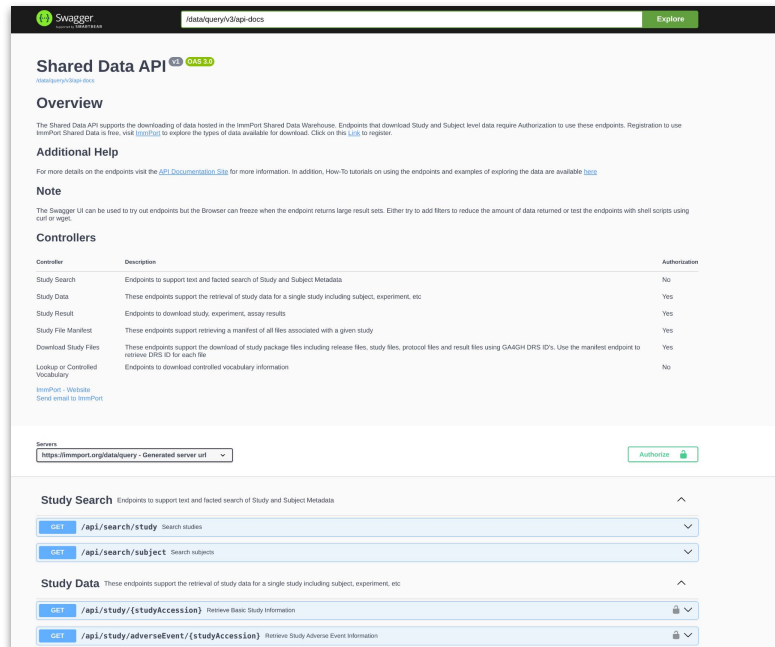


There are well-established approaches for describing your service in a format that's readable by both humans and machines.

One of the most popular is OpenAPI (formerly known as Swagger): <https://swagger.io/specification/>

**OpenAPI** = the official name of the **specification** (current versions: OpenAPI 3.0 and 3.1).

**Swagger** = the brand name for the popular suite of **tools** (Swagger UI, Swagger Editor, Swagger Codegen, etc.) that work with the OpenAPI Specification.



- Documentation is done in a simple text document
- Tooling provides for a common web based UI where APIs can be explored and tested directly
- Can be leveraged by machines/AI



# Fun with Descriptions

Well documented APIs can be used directly with current generation AI coding agents. Leveraging good documentation to generate code and even execute results for the user.

Additionally, these can be enhanced with some skill/tool descriptions layered around them to enhance their use.

## Generated Clients:

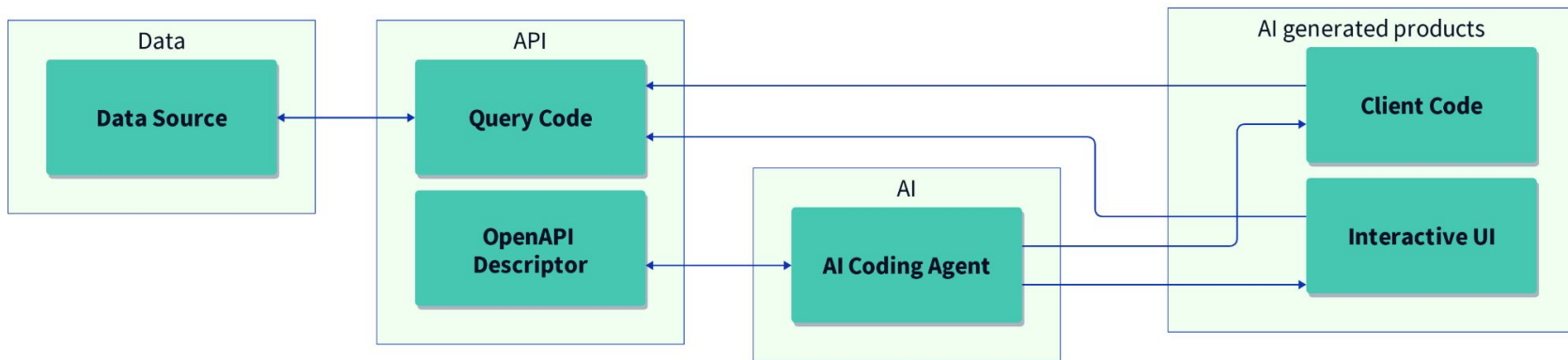
Current code generation system can produce simple clients based on OpenAPI specifications:

See: <https://github.com/go-fair-us/apireference/tree/master/code/clients/Import>

Additionally, many can execute code in an interactive "REPL" loop.

See: <https://github.com/go-fair-us/apireference/blob/master/docs/cluadeSession.md>

*Example flow for AI generated clients. There would be a similar workflow for human use.*



# A quick Example From the Notebook

Let's revisit some of these topics we just covered with an example pulled from the notebook.

Here is a Geospatial/Geocoding example API call that scopes many of the elements we have just reviewed.

The notebook:

- 1) Makes an API call to the [OpenStreetMap Nominatim](#) (Geocoding: Address ↔ Coordinates)
- 2) Converts the results to WKT and/or GeoJSON
- 3) Plots the results with Leaflet.

This simple process leverages much of what we just reviewed and brings up some new ones.

Unofficial OpenAPI:

<https://sparkfabrik.github.io/nominatim-openapi/#/>

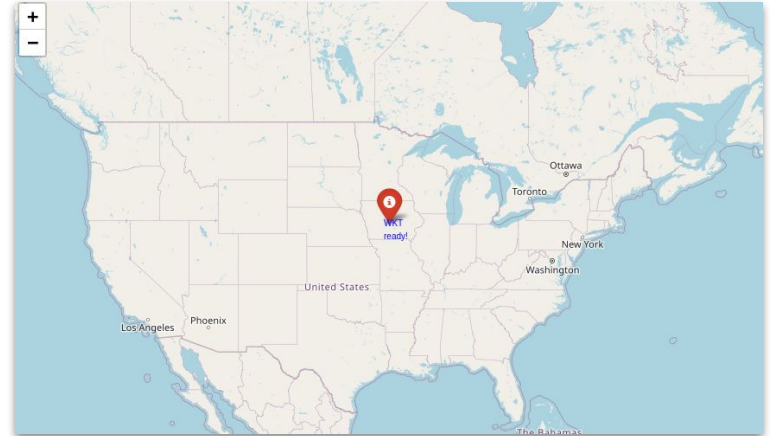
```
# ===== CONFIG =====
address = "Slater, Iowa"          # CHANGE THIS to anything
user_agent = "MyGeospatialAPIDemo/1.0 (NEMO@gmail.com)"
# REQUIRED by Nominatim policy – use your own name/email

# ===== API CALL SETUP =====
url = "https://nominatim.openstreetmap.org/search"

params = {
    "format": "json",             # ask for JSON
    "q": address,                 # the search query
    "limit": 1,                   # just the best match
    "addressdetails": 1           # extra info (optional but nice)
}

headers = {"User-Agent": user_agent}

response = requests.get(url, params=params, headers=headers)
```





# Ecosystem of Resources

# NIAID Data Ecosystem Discovery Portal Use of APIs

NDE Portal (<https://data.niaid.nih.gov/>) uses APIs and other methods for accessing metadata published by the community.

- 17 out of 41 resources catalogued in the Discovery Portal have discoverable APIs
- 37 out of 45 resources ingested by the Discovery Portal have APIs that are leveraged for the ingestion
- Primary metadata encoding format leveraged is JSON. ImmPort and others already express JSON-LD.
  - There is also XML and CSV formatted metadata records.
- Some APIs don't provide a full metadata coverage, which is then augmented with site crawling or exports from alternative API endpoints.

The screenshot shows the NIAID Data Ecosystem Discovery Portal search results for the query 'hasAPI:true'. The page features a search bar with the query, a 'Type' dropdown, and a 'Search' button. Below the search bar, there are search filters including a date range from 2000-01-01 to 2026-12-31, and a 'Clear All' button. The results are displayed in a grid of resource catalogs, including BacDive, Bacterial and Viral Bioinformatics Resource Center, and ClinVar. Each catalog entry shows the date, open access status, API availability, and content types. The page also includes a 'Notice' banner at the top and a 'Read More' button.

<https://data.niaid.nih.gov/search?q=hasAPI%3Atrue&filters=%28date%3A%5B%222000-01-01%22+TO+%222026-12-31%22%5D%29>

```
curl -X 'GET' \
  'https://api.data.niaid.nih.gov/v1/query?q=hasAPI%3Atrue&facet_size=10&fetch_all=true' \
  -H 'accept: */*'
```

Thanks to Ginger Tsueng for the material for this slide.

# APIs in NIAID

When resources are online and machine readable, there are many ways it can be leveraged to provide views into the landscape of resources.

This graph is formed from the data exposed by the NDE Portal and then translated to a graph.

**NOTE: The use of PIDs and vocabularies is vital in expressing networks like this.**

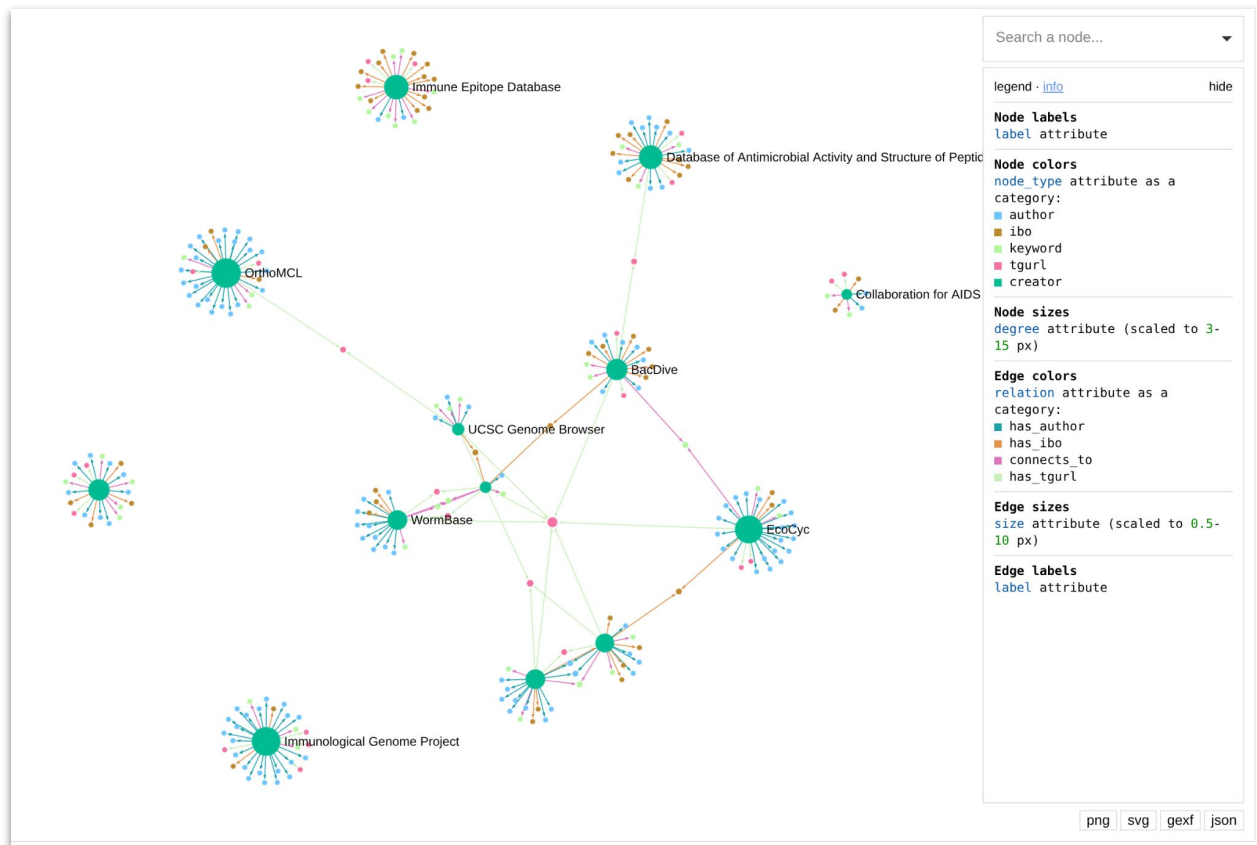
Notice the red dot near the middle with seven connecting edges. It is:

["http://edamontology.org/topic\\_0622"](http://edamontology.org/topic_0622)

Which redirects (303) to  
[https://bioportal.bioontology.org/ontologies/EDAM?p=classes&conceptid=topic\\_0622](https://bioportal.bioontology.org/ontologies/EDAM?p=classes&conceptid=topic_0622)

which is "Genomic"

```
curl -I http://edamontology.org/topic_0622
HTTP/1.1 303 See Other
Date: Thu, 26 Mar 2026 20:27:15 GMT
Server: Apache/2.4.63 (AlmaLinux) OpenSSL/3.5.1
X-Frame-Options: SAMEORIGIN
Location:
http://bioportal.bioontology.org/ontologies/EDAM?p=terms&conceptid=topic\_0622
Content-Type: text/html; charset=iso-8859-1
```





# AI and APIs: The Model Context Protocol (MCP)

**Model Context Protocol (MCP)** offers a clean, AI-ready way to expose your resources, tools, and capabilities so that AI models and agents can easily discover and use them.

It builds directly on familiar API concepts that many teams already understand, while adding the structure that AI systems need. In many ways, MCP can be thought of as **“APIs by convention for AI.”**

MCP is a stateful JSON-RPC 2.0 protocol that defines a clear, consistent set of methods. This gives AI models and agents a reliable way to:

- Discover available *tools, prompts, and resources*
- Understand exactly how to call them

Important Caution:

While MCP is powerful, it comes with unique security considerations. Teams and research groups should approach it with care and be diligent about security practices.

See more details at:

<https://github.com/go-fair-us/apireference/tree/master/docs/mcp>





CAUTION: Don't just wrap your existing APIs in an MCP wrapper. You really aren't accomplishing much, other than more maintenance requirements.

# MCP? Or, just APIs with descriptions or CLIs?

## ***Do I need MCP?***

Maybe, maybe not.

You might want to explore MCP if:

- If you have expressed interest from your community as an approach for access
- If you want a generic "AI Ready" service to provide a broad community via the network

If, you feel the maintenance and administration of a new service isn't justified, then you might tick the "AI Ready" box with:

- Machine readable API descriptions to allow client generation (*though not necessarily AI workflows*)
- A Command Line Interface (CLI) that an AI harness can leverage

Examples of resources that AI might be able to leverage already to address access and readiness goals.

Command Line Interface (CLI) access:

- BV-BRC: [https://www.bv-brc.org/docs/cli\\_tutorial/index.html](https://www.bv-brc.org/docs/cli_tutorial/index.html)
- Many others via shell scripts (ie, bash + curl)
  - Recently Google released a CLI for Google Workspace (<https://github.com/googleworkspace/cli>). Why was this done? A key driver was to allow agents to leverage it.

Examples of some of the OpenAPI 3.0/Swagger resources in NIAID

- IEDB: <https://query-api.iedb.org/docs/swagger/>
- TBPortals: <https://analytic.tbportals.niaid.nih.gov/index.html>
- ImmPort: <https://import.org/data/query/swagger-ui/index.html>
- ClinicalTrials: <https://clinicaltrials.gov/data-api/api>
- NIAID Data Ecosystem Discovery API
- 1.0.0: <https://api.data.niaid.nih.gov/>

# APIs and FAIR Principles

*The topics discussed here provide a foundation to establish a shared foundation to applying FAIR principles to APIs.*

## **Enhanced Discoverability & Access:**

- Make the API easily findable and usable by improving registration, providing clear access instructions, and ensuring persistent metadata URIs.

## **Machine-Readable & Interoperable Design:**

- Standardize documentation (OpenAPI/Swagger)
- Through data formats (JSON-LD, CSV with Schema, FIHR, etc) to ensure seamless machine interaction and integration with other systems.
- Leverage PIDs and semantics to help provide context, to humans and machines, for these.

## **Consistent & Shared API Practices:**

- Maintain uniformity across the API by using consistent approaches, HTTP methods, meaningful naming conventions, and aligning with established standards.

## **Robust Versioning & Management:**

- Implement clear versioning strategies and document changes to ensure API stability and support long-term reusability.

# APIs, good for humans and machines!



## ***For the human***

Things like formats, URL versioning, API approaches and explicit semantics **represent a shared context among the research community.**

## ***For the machine***

Things like formats, URL versioning, API approaches and explicit semantics **represent a shared *execution context* across distributed systems, services, and agents.**

# Upcoming Webinars

Date (12 PM ET)	Topics
March 31	<ul style="list-style-type: none"><li>• <b>Introduction to APIs</b> Covers Web API fundamentals, including stable “Cool URIs,” interoperable formats like JSON-LD and FHIR aligned with FAIR principles, and documentation practices such as HTTP status codes and OpenAPI/Swagger. It also explores how APIs are being integrated into emerging AI and biological data ecosystems, including approaches like the Model Context Protocol (MCP).</li></ul>
April 29	<ul style="list-style-type: none"><li>• <b>Improving Impact through Identifiers, Metadata and Citations</b> Explore how persistent identifiers for research assets creates the foundation for tracking and understanding research impact. Learn strategies for developing clear, domain-aligned citation recommendations for data, and gain exposure to how identifiers are indexed along with the metrics, tools, and techniques used to measure citation impact.</li></ul>
May 27	<ul style="list-style-type: none"><li>• <b>Increasing Findability and Interoperability: Introducing the Core Metadata Schema</b> Introduces the core metadata schema (MS) from the Blueprint, including its key elements and the motivations for adopting it.</li></ul>
June	<ul style="list-style-type: none"><li>• <b>Improving Impact through FAIR Implementation Profiles</b> Describe and share your repo’s use of FAIR identifiers, metadata, services, and other resources.</li></ul>





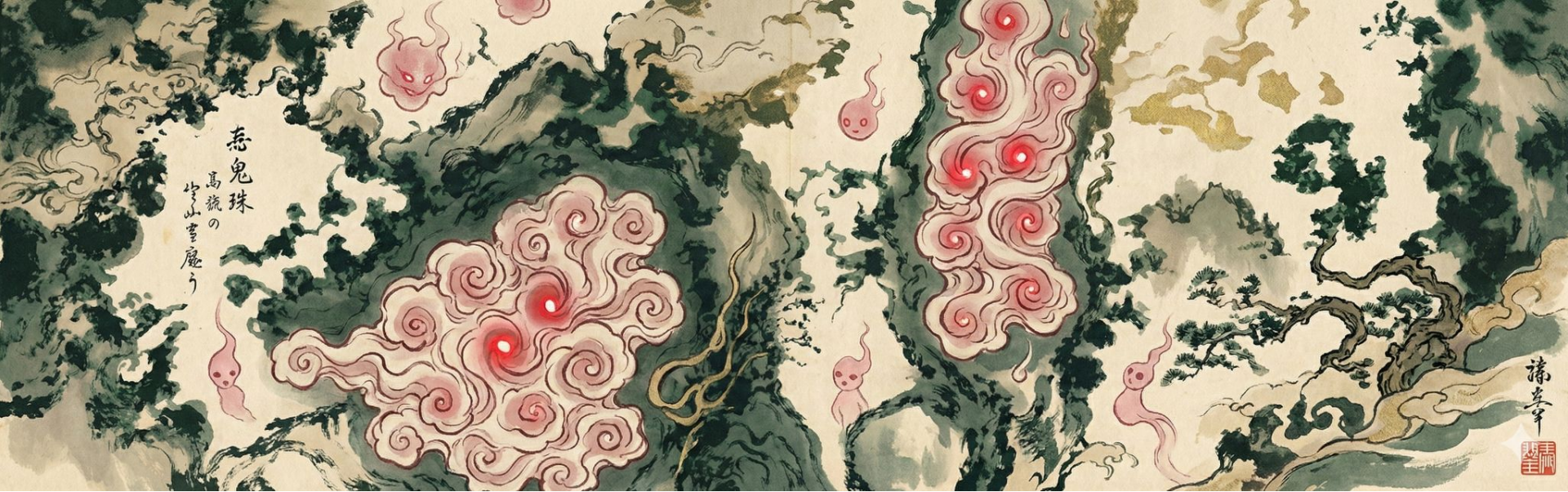
## Acknowledgements

We thank the NIH staff and community members who contributed to the Blueprint and NIAID FAIR landscaping project.

**Funding Acknowledgement:** This project has been funded in whole or in part with Federal funds from the National Cancer Institute, National Institutes of Health, under Contract No. 75N91019D00024, Task Order No. 75N91020F0022. The project is funded by the Frederick National Laboratory for Cancer Research, operated by Leidos Biomedical Research, Inc., on behalf of the National Institute of Allergy and Infectious Diseases.

**Disclaimer:** The content of this publication does not necessarily reflect the views or policies of the Department of Health and Human Services. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s).

Image source: <https://www.niaid.nih.gov/>



## Discussion

# APIs Need Versioning



- Versioning enables safe updates without breaking researcher tools
- Provides stable URLs for reliable long-term citations and impact
- Ensures data reproducibility across years
- Reduces maintenance effort and support tickets dramatically
- Strengthens FAIR Reusability and machine-actionability
- Future-proofs services as science and data models evolve

## Types of Versioning

- URI/Path Versioning (e.g., **/api/v1/resources**, **/api/v2/resources**):
- Header-Based Versioning (e.g., **Accept: application/vnd.myapi.v2+json** or custom header like **X-API-Version: 2**)
- Query Parameter Versioning (e.g., **/api/resources?version=2**)
  - Can make caching harder
- Media Type / Content Negotiation

There are some very robust and tested approaches to versioning like in Stripe and GitHub. An open source approach to this model can be seen in: <https://docs.cadwyn.dev/>

Strategy	Pros	Cons	Best For
URI/Path	Simple, visible, cacheable	URL fragmentation	Public/scientific APIs
Header/Media Type	Clean URLs, RESTful	Less discoverable	Long-term stability (FAIR)
Query Param	Flexible	Caching issues	Quick experiments

# APIs Should Use Standard Error Handling & Status Codes

Response codes communicate status between client and server

Code like 301, 404 and other are fundamental to the web and to linked data patterns.

**1xx** → Keep going / processing

**2xx** → Success!

**3xx** → Go somewhere else (redirect)

**4xx** → Your fault (fix the request)

**5xx** → Server's fault (try later or contact support)

**1xx**

The request was received but there isn't a response yet

**2xx**

The request was received successfully and the server can return the requested data

**3xx**

The requested resource is in another location, so a separate call is necessary

**4xx**

The request can't be completed due to an issue on the client's end

**5xx**

The request can't be completed due to an issue on the server's end

## Status Code Examples:

**200 OK:** The request was successful, and the server is returning the requested resource (e.g., a web page or API data loads perfectly).

**301 Moved Permanently:** The requested resource has been permanently moved to a new URL, and browsers/search engines should update their links to the new location.

**302 Found** (or 307 Temporary Redirect): The resource is temporarily available at a different URL, so the client should use the new address for this request but keep trying the original one in the future.

**404 Not Found:** The server cannot find the requested resource (the classic "page not found" error you see when a link is broken or a page was deleted).

**500 Internal Server Error:** A generic server-side failure occurred (something broke on the website's backend, and it couldn't process your request properly).

**403 Forbidden:** The server understood the request but is refusing to fulfill it, usually because you lack permission to access the resource (e.g., trying to view a private admin page).

**429 Too Many Requests:** You (or your device/IP) have sent too many requests in a short time, hitting the server's rate limit, so it's temporarily blocking you.

# Caching

## Better design = Better caching

- Use path for stable identifiers: /patients/123/reports/latest
- Put filters/sorting in query only when needed (and set Cache-Control wisely)
- Clean, persistent URLs
  - a. More cache hits
  - b. Faster APIs, less load

## Quick wins

- Avoid dynamic/temporary params in paths
- Use ETag/Last-Modified for conditional requests
- Set strong Cache-Control
  - a. max-age on stable resources

## Some patterns might break persistence

- /report?id=42&ver=3&cachebust=abc123
  - a. Looks temporary
  - b. People hesitate to link/share

## Query params

Many caches (CDNs, browsers, proxies) treat query strings as unique keys

- a. Low/no TTL or no caching at all for parameterized URLs
- b. Result: More server hits, slower responses, higher load