Digital NAO × TECMILENIO

Techno Ready

In-Mexico Program Backend Certification

Server and Database Commands | Sprint 1

Jorge Armando Avila Carrillo
NAO ID: 3310

September 30th, 2025

Index

Server and Database Commands

The project aims to automate the integration of researcher profiles and published articles from Google Scholar into a university's research database using the SerpApi Google Scholar Author API. This addresses key business challenges in academic institutions, such as manual data collection, which is time-consuming, error-prone, and inefficient for tracking scholarly output. By automating API-based data retrieval, parsing, and storage, the system enables real-time updates on metrics like citations, h-index, and publications, facilitating informed decisions on research funding, faculty evaluations, and collaborations. Ultimately, this enhances institutional efficiency, boosts research visibility, and supports strategic goals like improving university rankings and attracting talent. The solution leverages Java for robust backend processing, MVC patterns for modularity, and Postgresql for simple data persistence, ensuring scalability and ease of integration with existing systems.

Sprint 1

Sprint 1 focuses on researching and documenting the Google Scholar API via SerpApi and establishing project management foundations. The main tasks are:

- Create a SerpApi account under the free plan.
- Prepare a Technical Report summarizing essential information about the Google Scholar API (endpoints, authentication, query parameters, response formats, usage limits, code examples).
- Create a GitHub repository with a README.md file describing the project at a high level (purpose, key functionalities, relevance).
- Include the technical document in the repository.
- Configure access permissions to ensure the Digital NAO team can easily access the repository.

Documentation requirements:

- Create the Challenge Backlog with at least 5 user stories, both tracking tables (Product Backlog and Sprint Backlog), and Table 1 completed with requirements list.
- Create the Roadmap based on tutorial instructions.
- Compile the evidence into PDF reports (Backlog, Roadmap, Technical Report, GitHub link).
- Save all in a ZIP folder for submission.

Technical report: Google scholar API

Google Scholar is a web search engine focused on scholarly literature: journal articles, theses, books, conference papers, patents, and legal opinions.

Important Note: Google does *not* provide an official public API for Google Scholar. What are commonly referred to as "Google Scholar APIs" are actually **third-party services** (e.g., SerpApi, ScrapingDog, SearchAPI) that scrape and reformat Google Scholar results into developer-friendly APIs.

This report summarizes the essential information about such APIs, including **endpoints, authentication methods, query parameters, response formats, usage limits, and implementation examples** in Java (Spring Boot with org.json), Python, and JavaScript.

1 Key concepts

To effectively work with a Google Scholar API (or any API), it's crucial to understand these foundational concepts:

1.1 API (Application Programming Interface)

An **API** allows different software systems to communicate. Instead of manually navigating Google Scholar, you can programmatically send a request to an API and receive structured data (usually JSON). In this case, "Google Scholar API" refers to **third-party APIs that wrap scraped data**.

1.2 Endpoint

An **endpoint** is a specific URL that performs a function.

Example endpoints:

- **/search**→ search for publications.
- **/author** → retrieve author profile details.
- **/cites**→ get works citing a given paper.
- **/export**→ get citation in BibTeX/EndNote.

1.3 Authentication

APIs use **keys or tokens** to identify users.Typically, you sign up with a provider and receive an api_key.

Example:

https://api.example.com/search?q=AI&api_key=YOUR_KEY

1.4 Query parameters

Options you add to the endpoint URL to customize the request. Example:
?q=deep+learning&as_ylo=2020&as_yhi=2023

Typical parameters:

- **q** → search keywords.
- **cites** → citation ID to fetch citing works.
- **as_ylo, as_yhi** → filter by year range.
- **num, page** → pagination.
- **hl** → language code.

1.5 Response formats

Most APIs return **JSON** (nested objects/arrays).
Example:

1.6 Usage Limits

Providers impose restrictions:

- **Rate limits**: e.g., 100 requests/day on free plans.
- **Result caps**: e.g., max 100 results per search.
- **Quota resets**: daily or monthly.

If limits are exceeded, you may see **HTTP 429 (Too Many Requests)**.

1.7 Error handling

Common error codes:

- **400** → Bad request (wrong parameters).
- **401/403** → Authentication error.
- **429** → Too many requests.
- **500** → Server error.

1.8 Frameworks for Implementation

To use the API programmatically, you'll rely on frameworks:

- **Java (Spring Boot)** + org.json.
- **Python** + requests.
- **JavaScript (Node.js)** + axios of fetch.

2 Endpoints

Typical endpoints (provider-dependent):

Endpoint	Description
/search	Keyword-based search for publications.
/author	Author profile metadata (name, h-index, publications).
/cites	Fetch works citing a given publication.
/cluster	Group alternate versions of a paper.
/export	Export citation in BibTeX, EndNote, RIS.

Example (SerpApi):
https://serpapi.com/**search**?**engine**=google_scholar&**q**=quantum+physics&**api_key**=YOUR_KEY

3 Authentication

Most third-party Google Scholar APIs are **paid services**, so authentication is essential. Here's how it typically works:

3.1 API Key

Definition: A unique identifier (string) issued to you when you register with the provider.
Usage: Passed either as a **query parameter** or in the **request header**.
Examples:

As query parameter:

https://api.example.com/search?q=AI&api_key=YOUR_KEY

3.2 OAuth 2.0

Most Google Scholar scrapers **don't use OAuth**, but some large data providers use it. OAuth provides token-based access (time-limited tokens instead of static API keys). Safer but more complex (involves refresh tokens, scopes, etc.).

3.3 IP Whitelisting

Some providers allow you to bind your API key to **trusted IP addresses**. Prevents abuse if your API key leaks.

4 Query parameters

Common query parameters:

Parameter	Example	Description
q	machine learning	Search term.
cites	123456789	Return papers citing a given work.
as_ylo	2020	Lower bound year filter.
as_yhi	2023	Upper bound year filter.
hl	en	Language code.
num	10	Number of results.

5 Response formats

Most third-party APIs return data in **JSON**, but other formats may be supported. Here's what to expect.

- **JSON:** Human-readable and easy to parse in any language.
- **HTML (Raw Scholar Page):** Some APIs can return the raw HTML of Google Scholar results. Useful for custom scraping but less structured

6 Usage Limits

Every provider enforces **limits** to balance server load and monetize their service. Limits vary depending on your plan:

6.1 Rate Limits

Maximum requests per minute/hour/day

- Free tier: 100 requests/day.
- Paid tier: 5,000–50,000 requests/month.

6.2 Pagination Limits

Google Scholar itself only shows ~1,000 results max per query. APIs often cap at 10–20 results per request, with a page or start parameter to paginate.

6.3 Data Limits

Certain fields may be restricted on free plans (e.g., no author profile data, no citation exports).

6.4 Error Handling

When you hit a limit, you'll see HTTP errors:

- 401 Unauthorized → invalid/missing API key
- 403 Forbidden → plan restriction (e.g., citation endpoint not included).
- 429 Too Many Requests → exceeded rate limit (wait before retry).
- 500/503 → provider server issues.

7 Code examples

7.1 Java (Spring boot and org.json)

```java
import org.springframework.web.client.RestTemplate;
import org.json.JSONObject;
import org.json.JSONArray;

public class ScholarService {
    private final String API_KEY = "YOUR_KEY";
    private final String BASE_URL = "https://api.example.com/search";

    public JSONArray searchScholar(String query) {
        RestTemplate rt = new RestTemplate();
        String url = BASE_URL + "?engine=google_scholar&q=" + query + "&api_key=" + API_KEY;
        String body = rt.getForObject(url, String.class);
        JSONObject obj = new JSONObject(body);
        return obj.getJSONArray("results");
    }
}
```

7.2 Python

```python
import requests

API_KEY = "YOUR_KEY"
BASE_URL = "https://api.example.com/search"

def scholar_search(query):
    params = {"engine": "google_scholar", "q": query, "api_key": API_KEY}
    resp = requests.get(BASE_URL, params=params)
    data = resp.json()
    return data["results"]

for r in scholar_search("machine learning"):
    print(r["title"], r["year"], r["cited_by"])
```

7.3 Javascript (Node.js)

```javascript
const axios = require("axios");

const API_KEY = "YOUR_KEY";
const BASE_URL = "https://api.example.com/search";

async function scholarSearch(query) {
  const resp = await axios.get(BASE_URL, {
    params: { engine: "google_scholar", q: query, api_key: API_KEY }
  });
  return resp.data.results;
}

scholarSearch("deep learning").then(results => {
  results.forEach(r => console.log(r.title, r.year, r.cited_by));
});
```

References

SerpApi, “Google Scholar API - SERPAPI,” *SerpApi*. https://serpapi.com/google-scholar-api