Nombre completo: Juan Lucio Aurelio

Título del desafío: Sprint 1, Challenge 3

Nombre de la vía en la que está inscrito: In-México Program

Identificación de la NAO: 3345

Technical Report: Google Scholar API

1. Endpoints

Theory:

In an API, endpoints are the specific URLs that act as access points to different functionalities. Each endpoint represents a resource (e.g., researchers, publications, or citations). For Google Scholar, endpoints are typically accessed via third-party libraries or scraping-based APIs, since Google does not provide an official open API.

Google Scholar Common Endpoints (via third-party tools):

- /search?query=<keyword> → Retrieves articles matching a keyword.
- /author?author_id=<id> → Retrieves information about a specific author.
- /citations?paper_id=<id> → Retrieves citation data for a specific publication.

Endpoints allow the system to **communicate with Google Scholar's data layer**. Each endpoint provides a different view of the information, ensuring modular and flexible integration.

2. Authentication Methods

APIs typically require authentication to verify user identity and usage rights. This is usually managed with **API keys**, **OAuth tokens**, or session cookies. While Google Scholar does not provide an official API key, third-party APIs often require registration and issue an authentication token.

Example (with third-party API):

- Obtain an API key after creating an account.
- Add the key to the request header:
- Authorization: Bearer <your_api_key>

Authentication ensures that **only authorized users can access the service**, providing both security and usage tracking. Without authentication, usage could be abused and data access restricted.

3. Query Parameters

Query parameters are values sent with a request to **filter or refine search results**. They increase flexibility by allowing the same endpoint to serve different needs.

Common Google Scholar Parameters:

- query → The search term (e.g., "machine learning").
- author → Restrict results to a specific author.
- year_from and year_to → Filter results by publication year.
- num → Number of results per page.

Query parameters act as **customization tools**, allowing the user to extract precise data instead of overwhelming results. They are essential for efficient research workflows.

4. Response Formats

APIs return data in a structured format, typically **JSON (JavaScript Object Notation)** or **XML**. JSON is widely used because it is lightweight and easily parsed by most programming languages.

Example JSON Response (simplified):

{

"title": "Deep Learning in Neural Networks",

"authors": ["Jürgen Schmidhuber"],

"year": 2015,

"citations": 12345

5. Usage Limits

To prevent server overload and ensure fair use, APIs impose **rate limits** (number of requests allowed per minute/hour/day). For Google Scholar unofficial APIs, typical limits are:

- 100–200 requests per day for free tiers.
- Higher limits with paid subscriptions.

Exceeding limits may cause errors such as 429 Too Many Requests.

Usage limits encourage **responsible data access**. They ensure performance stability while balancing availability for all users.

6. Code Examples

Python Example (using scholarly library):

```
from scholarly import scholarly

# Search for an author

author = scholarly.search_author('Albert Einstein')

print(next(author))
```

Java Example (HTTP GET Request):

```
import java.net.*;

import java.io.*;



public class ScholarAPI {

    public static void main(String[] args) throws Exception {
```

```
        URL url = new
URL("https://api.scholar.example/search?query=machine+learning");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");


        BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer content = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();
        System.out.println(content.toString());
    }
}
```

Code examples illustrate the **practical implementation** of theory. They bridge the gap between abstract API concepts and real-world usage, making the process tangible and actionable.