

Wikipedia Streaming Project

Stetsyshyn Victoria, Savkiv Yaryna

Project Overview

The main task of the project was to process streaming data from Wikipedia using stream and batch processing so that the user could make API requests and receive analytical answers based on the collected data.

The system implements two types of API requests:

- ❖ **Category A** - requests to pre-computed aggregated reports, which are updated hourly based on batch processing.
- ❖ **Category B** - ad-hoc queries to the stored data, which allow you to get results in real time.

Project Architecture

The architecture of our project consists of 11 components.

- ★ **Wikipedia Stream** - data of real-time page creation events (JSON format)
- ★ **Kafka Producer** - reads the Wikipedia stream and pushes raw messages into the Kafka Input-Topic.
- ★ **Input-Topic** - stores the JSON format raw data from Wikipedia Stream, which means that the data isn't preprocessed yet at this stage.
- ★ **Spark Streaming Job** - reads from Kafka Input-Topic, transforms data (parsing and filtering), and writes to Processed-Topic
- ★ **Processed-Topic** - stores clean, structured messages, which are ready for the further processing

We decided to use **two Kafka topics** instead of writing directly to the database **in order to separate concerns and improve system flexibility**. This approach enables easy **replay of raw data**, **protects against invalid records**, and allows **asynchronous**, fault-tolerant processing — all of which contribute to a more scalable and resilient architecture.

Category A:

- ❖ **PostgreSQL** - intermediate relational database to store non-aggregated events coming from Kafka Processed-Topic after processing in Spark Streaming.

We chose PostgreSQL as the database for intermediate storage because it is a powerful relational database that is well suited for further analytics and aggregation computing, unlike Cassandra.

- ❖ **Pandas Aggregator** - scheduled batch processor that reads data from PostgreSQL, computes aggregates, and stores results into Cassandra.

Initially, we planned to implement full data processing and aggregation within a Spark Streaming Job, using periodic calls to PostgreSQL for further analysis using Cron jobs.

However, in the implementation process, we faced computing resource limitations, which made it impossible to implement this approach in full.

In this regard, it was decided to use a less resource-intensive approach - Python script (**Pandas Aggregator**) that is periodically executed and processes data from PostgreSQL.

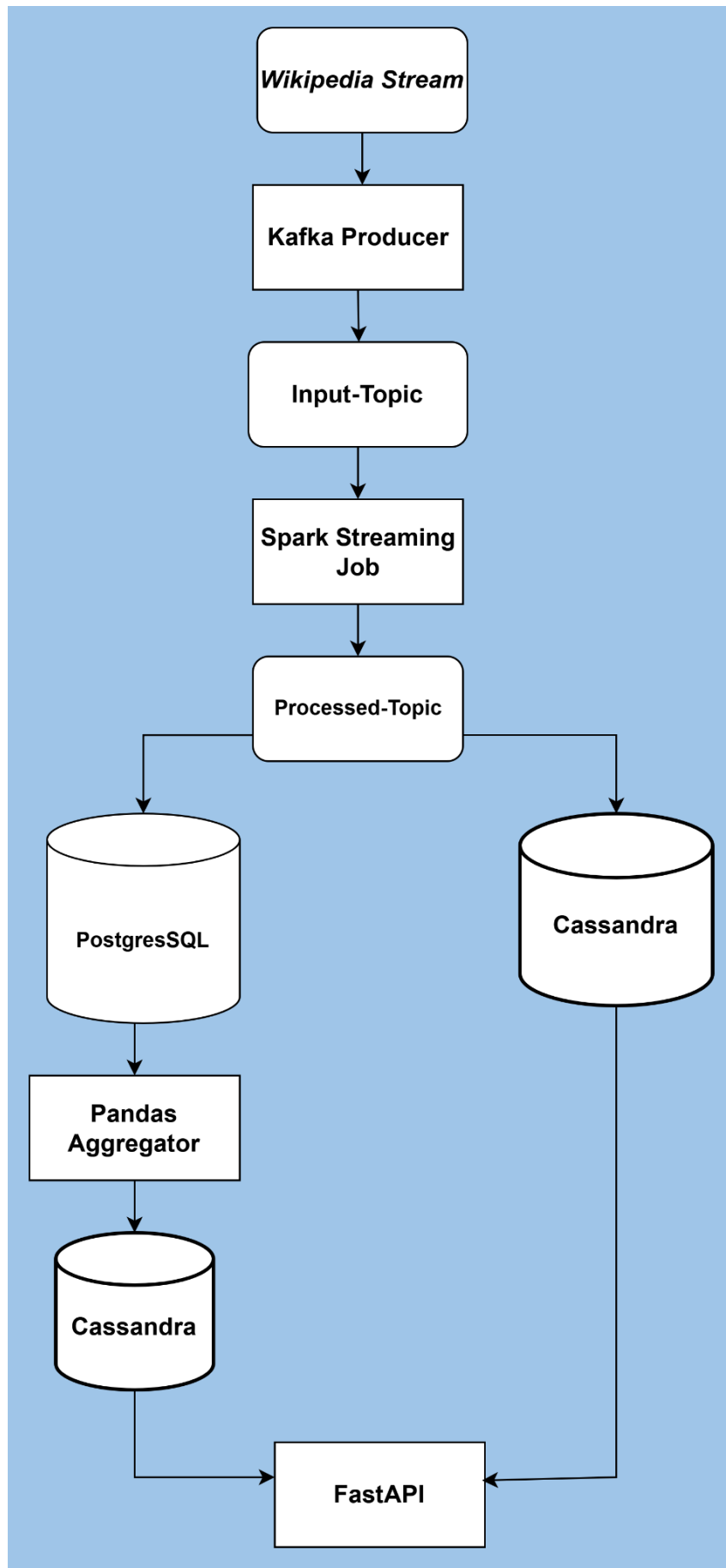
- ❖ **Cassandra** - database, which has 3 tables (*domain_stats_by_hour*, *bot_stats_last_6h*, *top_users_last_6h*), which are designed to store the final aggregated results required to serve the API requests from Category A, and are optimized for direct lookup based on time intervals.

Category B:

- ❖ **Cassandra** - database, which has 6 tables (*domains_created*, *pages_by_user*, *domain_page_counts*, *pages_by_id*, *user_page_counts_by_date*, *users*), which are designed to store the necessary data to serve the API requests from Category B.

For Both Categories:

- ❖ **FastAPI** - endpoints rely entirely on the precomputed data for category A and ad hoc queries for category B. Since the aggregations are already stored in Cassandra databases(for category A), API responses are extremely fast and efficient, as no real-time processing is required during request time. Category B requires a little more time to respond, but not significantly. This design ensures consistent low-latency performance even under high query loads. You can see the documentation and interact with the available API endpoints using **Swagger UI**



Data Models

1. PostgreSQL (*Relational database*)

Role: stores raw cleaned data for further batch analytics in Category A.

Why we used:

- simple structure writing from Kafka.
- supports JOIN, SQL queries, aggregation
- Gives opportunity to easily process necessary data for further analysis

```

1 CREATE TABLE IF NOT EXISTS wiki_events (
2     id SERIAL PRIMARY KEY,
3     domain TEXT,
4     created_at TIMESTAMP,
5     page_id BIGINT,
6     page_title TEXT,
7     user_id BIGINT,
8     user_name TEXT,
9     comment TEXT,
10    user_is_bot BOOLEAN
11 );|
```

2. Cassandra (*NoSQL database*)

Role: stores both pre-aggregated data (for Category A) and ‘fresh’ non-aggregated events for queries (Category B).

Why we used: provides very fast REST API reading

3. Kafka (*Message Broker*)

Role: handles real-time ingestion of events from Wikipedia Stream into the data pipeline.

Why we used: enables real-time, fault-tolerant data delivery, integrates seamlessly with Spark Streaming

4. Spark Streaming (*Stream Processing*)

Role: consumes messages from Kafka Input-Topic, parses and filters them, and outputs cleaned data into Processed-Topic, and then writes the structured results to both PostgreSQL (for Category A) and Cassandra (for Category B).

Why we used: integrates seamlessly with Kafka, and allows us to validate schemas and transform incoming Wikipedia event data before storing it in the topics and databases.

Created tables in Cassandra for Category A and Category B

- "Return the list of existing domains" - *domains_created*

Purpose: stores domains for which at least one page has been created.

```
CREATE TABLE IF NOT EXISTS domains_created (  
  
    domain TEXT,  
    created_at TIMESTAMP,  
    PRIMARY KEY (domain)  
);
```

- "Return all the pages created by the user with a specified user_id" - *pages_by_user*

Purpose: stores a list of pages created by a particular user

```
CREATE TABLE IF NOT EXISTS pages_by_user (  
  
    user_id BIGINT,  
    page_id BIGINT,  
    dt TIMESTAMP,  
    PRIMARY KEY (user_id, page_id)  
);
```

- "Return the number of articles created for a specified domain" - *domain_page_counts*

Purpose: stores the total number of pages created in a particular domain.

```
CREATE TABLE IF NOT EXISTS domain_page_counts (
```

```
    domain TEXT,  
    page_count COUNTER,  
    PRIMARY KEY (domain)
```

```
);
```

- **"Return the page with the specified page_id" - pages_by_id**

Purpose: allows you to find a page by its page_id

```
CREATE TABLE IF NOT EXISTS pages_by_id (
```

```
    page_id BIGINT,  
    page_title TEXT,  
    domain TEXT,  
    dt TIMESTAMP,  
    PRIMARY KEY (page_id)
```

```
);
```

- **"Return id, name, and number of created pages for all users in a specified time range" - user_page_counts_by_date**

Purpose: allows you to get information about user activity in a certain time range.

```
CREATE TABLE IF NOT EXISTS user_page_counts_by_date (
```

```
    dt DATE,  
    user_id BIGINT,  
    page_count COUNTER,  
    PRIMARY KEY ((dt), user_id)
```

```
);
```

- This table is used as helper table for previous task, because in the table **user_page_counts_by_date** we used **COUNTER** and it is impossible to store **TEXT** values and **COUNTER** values at the same table, but we need to give the **user_name** - so we created table **users**

Purpose: stores the **mapping** **user_id** and **user_name**.

CREATE TABLE IF NOT EXISTS users (

user_id BIGINT PRIMARY KEY,
user_name TEXT);

- **"Return aggregated statistics per domain per hour" - domain_stats_by_hour**

Purpose: aggregate statistics - the number of pages created for each domain per hour.

CREATE TABLE IF NOT EXISTS domain_stats_by_hour (

time_start TIMESTAMP,
domain TEXT,
page_count INT,
PRIMARY KEY ((time_start), domain)
);

- **"Return statistics about pages created by bots" - bot_stats_last_6h**

Purpose: stores the number of pages created by bots in the last 6 hours.

CREATE TABLE IF NOT EXISTS bot_stats_last_6h (

time_start TIMESTAMP,
domain TEXT,
created_by_bots INT,
PRIMARY KEY ((time_start), domain)
);

- **"Return Top 20 users that created the most pages" - top_users_last_6h**

Purpose: stores the top 20 users who have created the most pages in the last 6 hours.

```
CREATE TABLE IF NOT EXISTS top_users_last_6h (
    time_start TIMESTAMP,
    user_id BIGINT,
    user_name TEXT,
    page_count INT,
    page_titles LIST<TEXT>,
    PRIMARY KEY ((time_start), page_count, user_id)
) WITH CLUSTERING ORDER BY (page_count DESC);
```

Results

Category A:

- **/analytics/domain-stats**

Returns a list of domain-level page creation statistics for each of the last 6 hours (excluding the current hour).

default

GET /analytics/domain-stats Get Domain Stats

Parameters

No parameters

Execute

Responses

Curl

```
curl -X 'GET' \
  'http://0.0.0.0:8000/analytics/domain-stats' \
  -H 'accept: application/json'
```

Request URL

```
http://0.0.0.0:8000/analytics/domain-stats
```

Server response

Code	Details
200	Response body

GET /analytics/bot-stats Get Bot Stats

Parameters

No parameters

Execute

Responses

Curl

```
curl -X 'GET' \
  'http://0.0.0.0:8000/analytics/bot-stats' \
  -H 'accept: application/json'
```

Request URL

```
http://0.0.0.0:8000/analytics/bot-stats
```

Server response

Code	Details
200	Response body

- **/analytics/bot-stats**

Returns hourly statistics on the number of pages created by bots per domain for the last 6 hours.

GET

/analytics/bot-stats

Get Bot Stats

Parameters

No parameters

Execute

Responses

Curl

```
curl -X 'GET' \
  'http://0.0.0.0:8000/analytics/bot-stats' \
  -H 'accept: application/json'
```

Request URL

http://0.0.0.0:8000/analytics/bot-stats

Server response

Code

Details

200

Response body

```
[
  {
    "time_start": "2025-05-16T23:00:00",
    "time_end": "2025-05-17T00:00:00",
    "statistics": []
  },
  {
    "time_start": "2025-05-17T00:00:00",
    "time_end": "2025-05-17T01:00:00",
    "statistics": []
  },
  {
    "time_start": "2025-05-17T01:00:00",
    "time_end": "2025-05-17T02:00:00",
    "statistics": [
      {
        "domain": "ar.wikipedia.org",
        "created_by_bots": 11
      },
      {
        "domain": "ckb.wikipedia.org",
        "created_by_bots": 1
      },
      {
        "domain": "commons.wikimedia.org"
```

Response headers

```
content-length: 1291
content-type: application/json
```

- **/analytics/top-users**
- **Returns the top 20 users by page creation count for each of the last 6 hours, along with page titles and user info.**

GET

/analytics/top-users

Get Top Users

Parameters

No parameters

Execute

Responses

Curl

curl -X 'GET' \
'http://0.0.0.0:8000/analytics/top-users' \
-H 'accept: application/json'

Request URL

http://0.0.0.0:8000/analytics/top-users

Server response

Code	Details
200	<div>Response body</div> <div>"users": [] }, { "time_start": "2025-05-17T01:00:00", "time_end": "2025-05-17T02:00:00", "users": [{ "user_id": 64151, "user_name": "Caviak80T", "page_count": 246, "page_titles": ["yaklaşmalarınız", "yaklaştırmaya", "yaklaştırmada", "yaklaşmanızı", "yaklaştırmalarımı", "yaklaştırmamın", "yaklaştırmamı", "yaklaştırmalarımın", "yaklaştırmalarından", "yaklaştırmalarının", "yaklaştırmamda", "yaklaştırmamda", "yaklaştırmamda", "yaklaştırmamdan"] }] } } </div> <div>Response headers</div> <div>content-length: 308638 content-type: application/json date: Sat, 17 May 2025 06:17:15 GMT server: uvicorn</div>

Responses

GET

/analytics/top-users

Get Top Users

Parameters

No parameters

Execute

Responses

Curl

curl -X 'GET' \
'http://0.0.0.0:8000/analytics/top-users' \
-H 'accept: application/json'

Request URL

http://0.0.0.0:8000/analytics/top-users

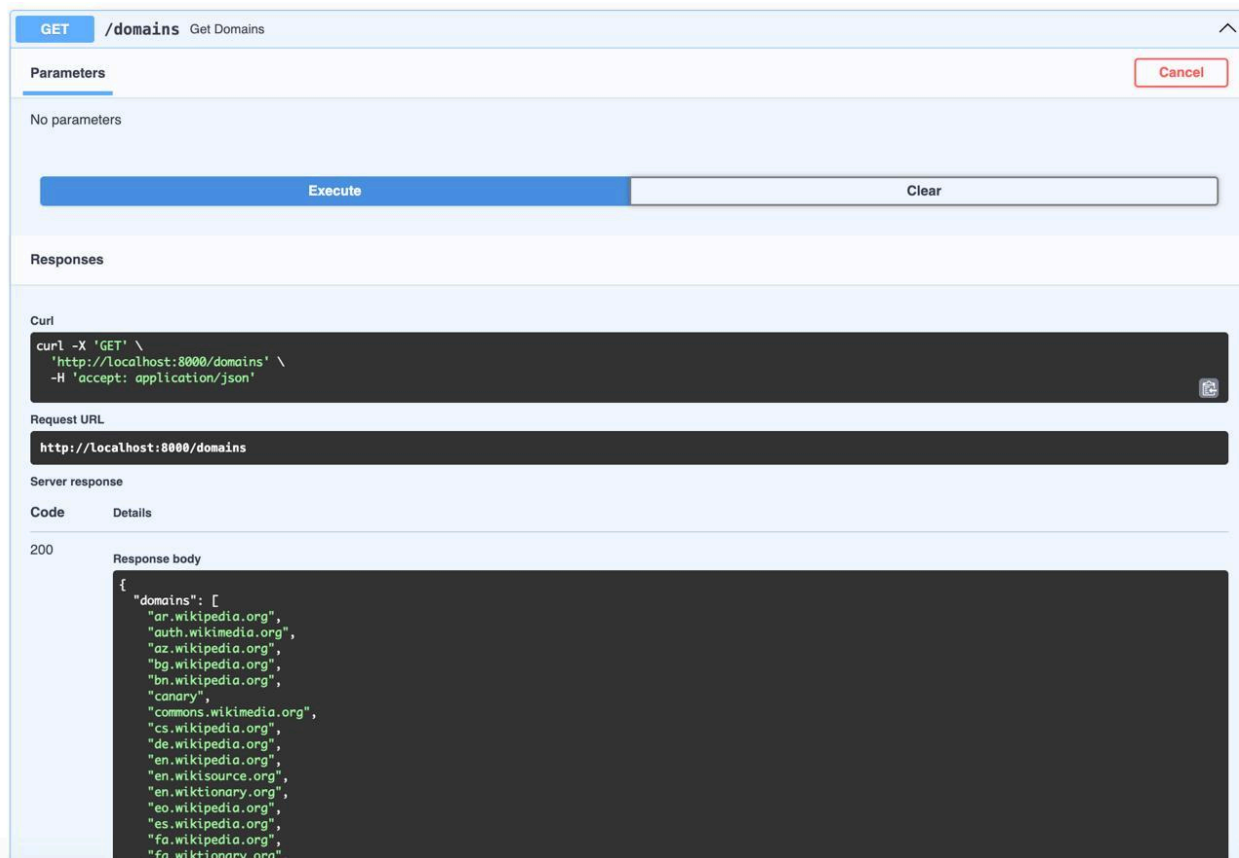
Server response

Code	Details
200	<div>Response body</div> <div>{ "user_id": 5142080, "user_name": "Gzen9280t", "page_count": 333, "page_titles": ["File:Oeuvres complètes..6..Hippolyte et Aricie_-_tragédie en cinq actes et un t_d'Indy_-_bpt6k3274266_(492_of_578).jpg", "File:6_Quatuors extraits des oeuvres de Boccherini, arrangés pour flûte, clarin File:IL_-_Fanatico per gli antici Romani_-_Del_Sig._r.D..Domenico Cimarosa_-_b File:Haydée._Opéra-comique en trois actes._Paroles de Eugène Scribe..._-_bpt6k File:IL_-_Fanatico per gli antici Romani_-_Del_Sig._r.D..Domenico Cimarosa_-_b File:Motets à une et deux voix pour tout le chœur à l'usage de l'église et co e_second_contenant_tous_les_motets..._-_btv1b53059255x_(059_of_179).jpg", "File:Motets à une et deux voix pour tout le chœur à l'usage de l'église et co e_second_contenant_tous_les_motets..._-_btv1b53059255x_(060_of_179).jpg", "File:Motets à une et deux voix pour tout le chœur à l'usage de l'église et co e_second_contenant_tous_les_motets..._-_btv1b53059255x_(061_of_179).jpg", "File:IL_-_Fanatico per gli antici Romani_-_Del_Sig._r.D..Domenico Cimarosa_-_b File:Haydée._Opéra-comique en trois actes._Paroles de Eugène Scribe..._-_bpt6k File:Motets à une et deux voix pour tout le chœur à l'usage de l'église et co e_second_contenant_tous_les_motets..._-_btv1b53059255x_(062_of_179).jpg", "File:6_Quatuors extraits des oeuvres de Boccherini, arrangés pour flûte, clarin File:IL_-_Fanatico per gli antici Romani_-_Del_Sig._r.D..Domenico Cimarosa_-_b File:IL_-_Fanatico per gli antici Romani_-_Del_Sig._r.D..Domenico Cimarosa_-_b </div> <div>Response headers</div> <div>content-length: 308638 content-type: application/json date: Sat, 17 May 2025 06:17:15 GMT server: uvicorn</div>

Responses

Category B:

- /domains



Returns list of unique text domains!

- **/pages/by-user**

Name	Description
user_id * required integer (query)	<input type="text" value="4238209"/>

Execute

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/pages/by-user?user_id=4238209' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8000/pages/by-user?user_id=4238209
```

Server response

Code	Details
200	<p>Response body</p> <pre>[{ "page_id": 165436748, "dt": "2025-05-17T07:36:44" }, { "page_id": 165436773, "dt": "2025-05-17T07:37:08" }, { "page_id": 165436794, "dt": "2025-05-17T07:37:41" }, { "page_id": 165436812, "dt": "2025-05-17T07:38:15" }, { "page_id": 165436833, "dt": "2025-05-17T07:38:53" }]</pre>

Returns a list of pages (with timestamps) created by a specific user.

- /domain/count

Parameters Cancel

Name	Description
domain * required string (query)	<input type="text" value="az.wikipedia.org"/>

Returns the total number of pages created for a specified domain.

- /page/by-id

Returns detailed information about a page with the specified ID, including title, domain, and creation time.

GET /page/by-id Get Page By Id

Parameters

Name	Description
page_id * required integer (query)	<input type="text" value="165437137"/>

Execute

Responses

Curl

```
curl -X 'GET' \  
  'http://localhost:8000/page/by-id?page_id=165437137' \  
  -H 'accept: application/json'
```

Request URL

```
http://localhost:8000/page/by-id?page_id=165437137
```

Server response

Code	Details
200	<div>Response body<pre>{ "page_id": 165437137, "page_title": "File:OJ_C_320_of_2013_-_DE_German.pdf", "domain": "commons.wikimedia.org", "dt": "2025-05-17T07:44:55" }</pre></div> <div>Response headers<pre>content-length: 134 content-type: application/json date: Sat, 17 May 2025 08:09:52 GMT server: uvicorn</pre></div>

- `/users/activity`

Returns the total number of pages created by each user within a given date range, sorted by activity.

GET `/users/activity` Get User Activity

Parameters

Name	Description
start_date * required string(\$date) (query)	<input type="text" value="2025-05-17"/>
end_date * required string(\$date) (query)	<input type="text" value="2025-05-17"/>

Execute

Responses

Curl

```
curl -X 'GET' \  
'http://localhost:8000/users/activity?start_date=2025-05-17&end_date=2025-05-17' \  
-H 'accept: application/json'
```

Request URL

```
http://localhost:8000/users/activity?start_date=2025-05-17&end_date=2025-05-17
```

Server response

Code	Details
200	<div>Response body</div> <pre>[{ "user_id": 5142080, "user_name": "Gzen92Bot", "page_count": 132 }, { "user_id": 7459497, "user_name": "Wilfried Mahlfeld", "page_count": 45 }, { "user_id": 3919627, "user_name": "HWY Shield Bot", "page_count": 43 }, { "user_id": 6938613, "user_name": "Muufii", "page_count": 31 }, { "user_id": 2570959, "user_name": "BrankaVV", "page_count": 29 }]</pre>