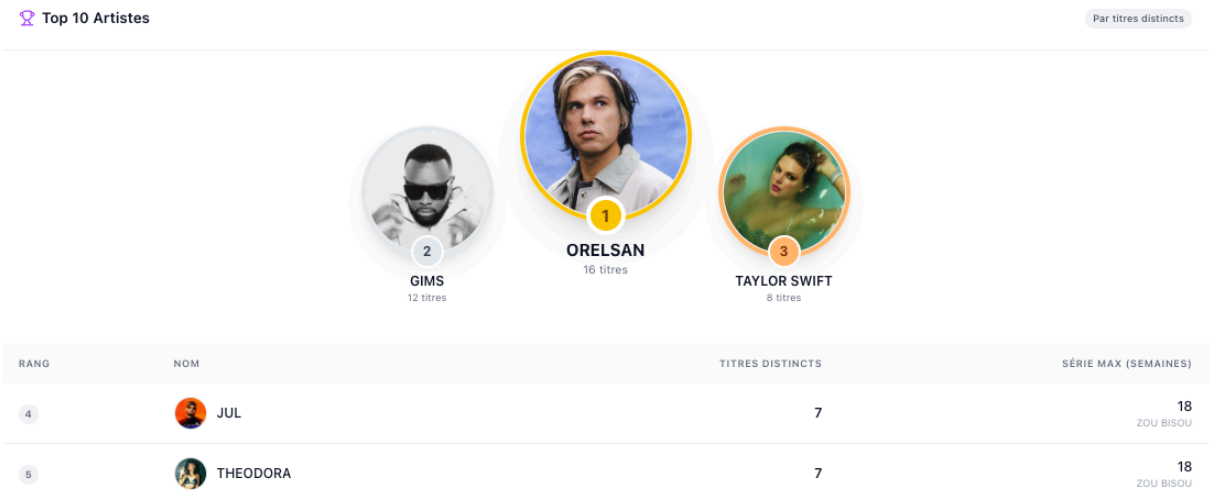
Data Analytics

French Top 200 Insight: Visualising the French Charts for Everyone



Hennebert Camil

Table of content

1. Context, Use Case, Objectives

1.1 Context

The French music industry has been profoundly transformed by streaming. Decisions made by labels, publishers and managers are increasingly data-driven: number of streams, chart presence, track longevity, and so on.

Every week, the SNEP (Syndicat National de l'Édition Phonographique) publishes the **Top 200 singles** in France. This ranking is a key reference for measuring the commercial success of a track. However, the SNEP website has several limitations from an analytical point of view:

- the display is strictly **week by week**, with no aggregated view or easily exploitable history;
- there is no **global view** that makes it easy to identify which artists or publishers have been most present in the charts since 2020;
- the site provides **no information at all on composers / beatmakers**: only the "artist / track / publishers" side is visible;
- the interface is not designed as an exploration or visualisation tool.

At the same time, there is a growing interest from the general public in **music statistics**. A good example is **Spotify Wrapped**: a personalised yearly recap that Spotify sends to its users, highlighting their most-listened artists, tracks and genres in the form of visuals optimised for social media sharing. This type of content generates huge engagement and shows how sensitive audiences are to data and "scores" associated with music.

In this context, there is a clear gap for a tool that offers a **different reading of the Top 200**: aggregating data over several years, bringing **composers/beatmakers** back into the analysis, and highlighting the artists and publishers that have actually shaped the French market since 2020.

1.2 Main Use Case

The application I am developing aims to:

Provide an aggregated view of the SNEP Top 200 in order to easily identify which artists, beatmakers and publishers have been most present in the French charts since 2020, and to analyse their "performance" over time.

Concretely, the web app allows users to:

- see which artists, beatmakers or publishers appear most frequently in the Top 200;
- observe how their chart presence evolves **week by week and year by year** (rather than only through isolated snapshots);
- analyse the "performance" of their favourite artists through a **search** feature;
- reconstruct the "behind the scenes" side of tracks by highlighting **composers and beatmakers** who are not visible on the SNEP website but can be identified via the Genius API;
- access a more readable and more engaging view of the charts than the simple weekly view displayed on the SNEP site.

The use case targets both:

- **professionals** (managers, publishers, A&Rs, music journalists) who want a better understanding of the recent French landscape;
- and **music enthusiasts**, who, like me, enjoy having this kind of data "at their fingertips" to follow chart activity, compare artists or simply satisfy their curiosity.

1.3 Project Objectives

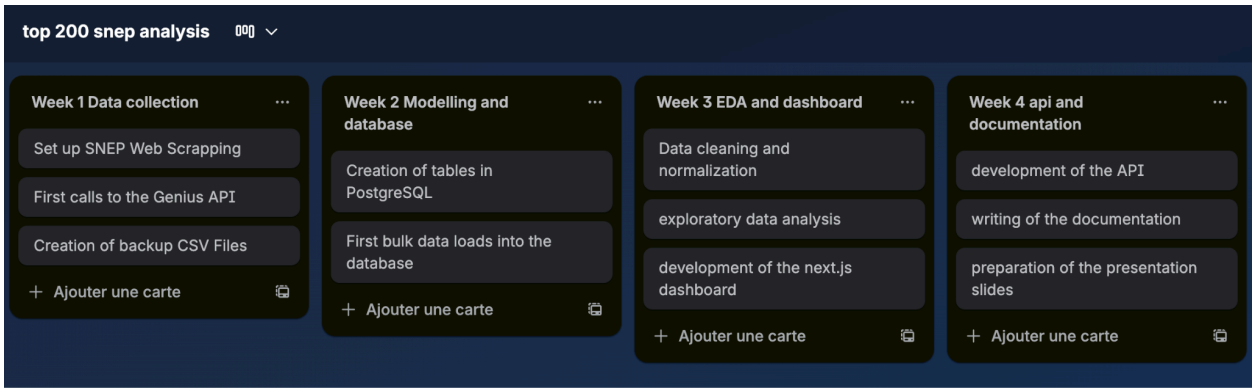
Functional objectives

- **Aggregate** and centralise SNEP Top 200 data from 2020 up to today.
- **Enrich** this dataset with additional information (notably via the Genius API), such as producers/beatmakers, songwriters and, when available, publishers.
- Provide a **web interface** that allows users to:
    - visualise, in an aggregated way, which artists, beatmakers and publishers are most present in the charts;
    - track their performance **by week and by year** (number of tracks, cumulative weeks in the charts, peak position, etc.);
    - search for an artist or beatmaker and quickly display their “profile” in the charts since 2020.
- Set up a **REST API** exposing this information for potential external uses (other applications, dashboards, additional analyses).

Personal and educational objectives

- Design a tool that addresses a real need: making SNEP data more accessible, readable and actionable, while reintegrating **composers/beatmakers** into the analysis.
- Put into practice the skills acquired in **data engineering** (collection, cleaning, modelling, APIs, visualisation).
- Build a project that reflects my personal interest in music statistics and in monitoring the French ecosystem (artists, beatmakers, publishers).
- **Contribute to giving more visibility to beatmakers**, who often remain invisible to the general public even though they are at the heart of contemporary music creation.

2. Project planning



3. Data Collection and Storage

3.1 Main Data Source: SNEP Top 200

For the main data source, **I used the official website of the SNEP** (Syndicat National de l'Édition Phonographique), which is the French music industry body that publishes the official singles charts. I scraped the weekly Top 200 in order to retrieve, for each week, the ranking, the track title, the main artist name and the publisher/label.

The time window starts in 2020 and goes up to today. **I chose this period for two reasons:**

- **the music industry has changed dramatically since COVID** (explosion of streaming, new consumption habits),
- **going too far back** in time would add historical noise that is **not directly relevant to my objective,** which is to analyse the current market value of beatmakers over roughly the last five years.

(The code shown here is only an excerpt, the full functions are too long to include entirely in the report.)

```python
class SNEPScraper:
    def __init__(self, delay_between_requests=1.5):
        """
        Initializes the SNEP scraper

        Args:
            delay_between_requests: Delay in seconds between each request
        """
        self.base_url = "https://snepmusique.com/les-tops/le-top-de-la-semaine/top-albums/"
        self.delay = delay_between_requests
        self.session = requests.Session()
        # Disable SSL verification to avoid local certificate errors
        self.session.verify = False
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/91.0.4472.124 Safari/537.36'
        })

        # Create data folder if it doesn't exist
        self.data_dir = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), 'data')
        if not os.path.exists(self.data_dir):
            os.makedirs(self.data_dir)
            logger.info(f"Folder '{self.data_dir}' created")

        # Cache initialization
        self.cache_file = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
'snep_scrap_cache.json')
        self.cache = self.load_cache()
```

3.2 Intermediate Storage: CSV Layer with pandas

In a first step, **all scraped data were stored as CSV files using pandas.** This provided a simple and transparent intermediate layer: I could inspect the raw data, redo the cleaning pipeline if needed, and keep a backup of each extraction step.

3.3 External Enrichment: Genius API

To identify the composers and producers behind each track, **I then cross-joined the SNEP data with the Genius API.** For each chart entry, a **Python script queries Genius using the artist and track title**, retrieves the song page, and extracts the list of credited producers/beatmakers. These credits are then merged back into my CSVs so that every line in the dataset contains, in addition to the SNEP information, the associated composers.

```python
# Configuration
PROJECT_ROOT = Path(__file__).resolve().parent.parent
DATA_DIR = PROJECT_ROOT / "data"
CACHE_FILE = PROJECT_ROOT / "song_cache_v2.json"

# Load environment variables
if load_dotenv:
    # Try to load from viz_dashboard/.env.local
    env_path = PROJECT_ROOT / 'viz_dashboard' / '.env.local'
    if env_path.exists():
        load_dotenv(env_path)
    else:
        # Fallback to root .env if it exists
        load_dotenv(PROJECT_ROOT / '.env')

# Use an environment variable for the token (security)
ACCESS_TOKEN = os.getenv("GENIUS_ACCESS_TOKEN")
if not ACCESS_TOKEN:
    logging.warning("GENIUS_ACCESS_TOKEN is not defined in environment
variables.")
BASE_URL = "https://api.genius.com"

# Logging configuration
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler(PROJECT_ROOT / 'update_data.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)
```

3.4 Data Cleaning and Normalisation in Python

Data cleaning and normalisation were also performed in Python. **One of the main issues with the SNEP website is how artists and featurings are written:** featured artists can appear in the same string as the main artist, inside parentheses in the title, and with many variants of “feat” (feat, ft, ft., featuring, &, commas, etc.).

To handle this, **I implemented a small parsing pipeline based on regular expressions:**

- **a function to standardise separators** (feat/ft/featuring, &, commas) **into a single temporary separator and split them into up to four explicit columns** (artiste, artiste_2, artiste_3, artiste_4),
- **a dedicated handler for the letter “X” as a separator** (e.g. *Artist A X Artist B*), which only treats “X” as a delimiter when it really looks like two artist names,
- **a function to clean track titles by removing parentheses and**, when a (feat …) is detected in the title, extracting the featured artists into a separate list,
- **a merge function that combines main and featured artists without duplicates**, filling the available artiste_* columns in a consistent way.

```
def handle_x_separator(text):
    """
    Smartly handles X as an artist separator
    """
    # Pattern to detect an X surrounded by spaces between words that look like names
    # We look for: [Word(s)] X [Word(s)] where words start with a capital letter
    x_pattern = r'\b([A-Z][A-Za-z\s]+?)\s+X\s+([A-Z][A-Za-z\s]+?)\b'

    def replace_x(match):
        artist1 = match.group(1).strip()
        artist2 = match.group(2).strip()

        # Additional checks to ensure they are artist names
        # Avoid replacing if words are too short or contain suspicious characters
        if (len(artist1) >= 2 and len(artist2) >= 2 and
            not re.search(r'\d{3,}', artist1 + artist2) and  # Avoid long numbers
            not re.search(r'\b(THE|AND|OF|FOR|WITH|IN|ON|AT)\b', artist1 + " " + artist2,
re.IGNORECASE)):
            return f"{artist1}|SEPARATOR|{artist2}"
        else:
            # Return original text if it doesn't look like artist names
            return match.group(0)

    return re.sub(x_pattern, replace_x, text)
```

This pipeline ensures that each artist involved in a track is stored in a structured form, rather than in a single free-text field. It is crucial for later computing accurate statistics per artist and per beatmaker, and it results in well-structured CSV files containing all the information required for the project (charts, artists, publishers, beatmakers).

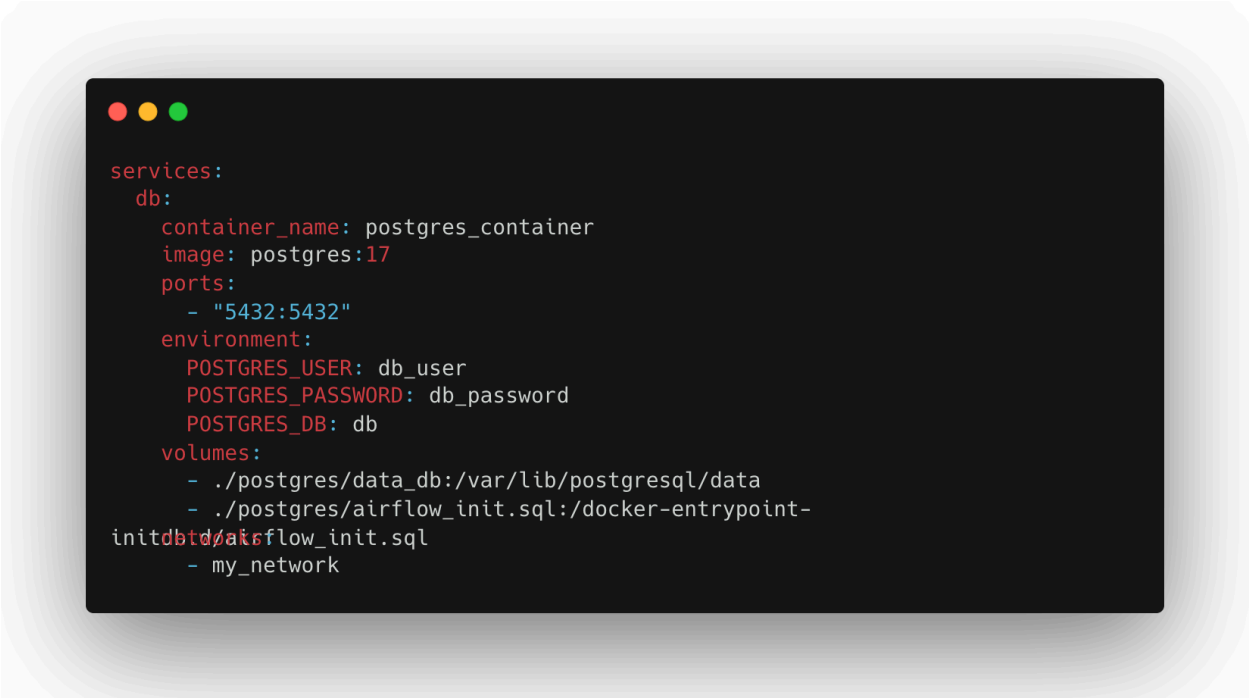
3.5 Relational Storage: PostgreSQL Schema

For the web application, staying on flat files was not an option. CSVs are fine for exploration, but they are not designed for:

- concurrent access from multiple users,
- complex filtering, joins and aggregations needed by the dashboard and the API,
- indexing and performance at scale as the dataset grows week after week,
- guaranteeing data integrity through constraints (foreign keys, unique keys, etc.).

I therefore designed a schema in PostgreSQL, deployed via Docker, and migrated the cleaned data into this relational database. In the current version, the schema contains one main table per year (top_single_snep_2020, top_single_snep_2021, etc.) within a db schema, plus dimension tables for artists, beatmakers and publishers used by the application.

PostgreSQL is a natural fit for a Next.js application because it can be queried efficiently from the backend (via an ORM or SQL client), supports transactions, and scales much better than reading CSVs on every request. It also makes it easier to expose the same data through the Flask API and to maintain a single, consistent source of truth.



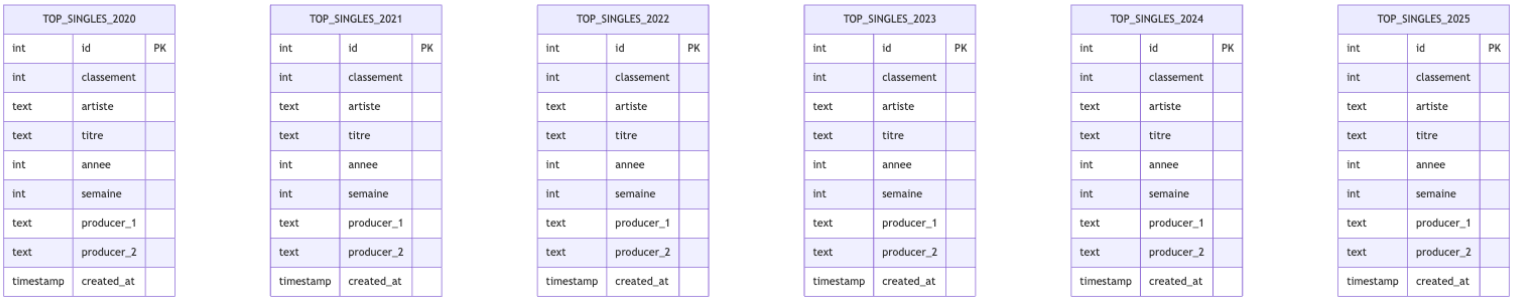
```
services:
  db:
    container_name: postgres_container
    image: postgres:17
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: db_user
      POSTGRES_PASSWORD: db_password
      POSTGRES_DB: db
    volumes:
      - ./postgres/data_db:/var/lib/postgresql/data
      - ./postgres/airflow_init.sql:/docker-entrypoint-
initdb.d/airflow_init.sql
    networks:
      - my_network
```

For the schema of my database, I chose to create one table per year for several reasons:

- **Query performance:** Each table contains around 10,400 rows (200 × 52 weeks). Queries on a specific year are instantaneous, without having to filter over 60,000+ rows.
- **Simplified maintenance:** You can easily drop or reload a full year (TRUNCATE top_singles_2024) without impacting any other data.
- **Data isolation:** An issue on 2025 data (the current, potentially unstable year) will never affect validated historical data.
- **Horizontal scalability:** Adding 2026, 2027, etc. simply means creating a new table. No schema migration is required.
- **Granular backups:** You can back up or restore a specific year independently from the others.
- **Optimised indexes:** Each table has its own indexes, which remain smaller and faster to scan.
- **Natural year-based analysis:** Queries like “Top artists in 2024” don’t need a WHERE year = 2024 clause; they directly target the relevant table.
- **Drawback:** Cross-year queries require UNION ALL, but this is acceptable since such analyses are less frequent and the total volume remains relatively small (~60k rows).

Entity Relationship Diagram

> **Note**: One identical table per year (2020-2025), all sharing the same structure.



Tables Overview

> **Architecture**: 6 independent tables, one per year of data.

Table	Year	Description	Records Pattern
`top_singles_2020`	2020	Weekly Top 200 charts	~200 rows × 52 weeks
`top_singles_2021`	2021	Weekly Top 200 charts	~200 rows × 52 weeks
`top_singles_2022`	2022	Weekly Top 200 charts	~200 rows × 52 weeks
`top_singles_2023`	2023	Weekly Top 200 charts	~200 rows × 52 weeks
`top_singles_2024`	2024	Weekly Top 200 charts	~200 rows × 52 weeks
`top_singles_2025`	2025	Weekly Top 200 charts	~200 rows × current week

3.6 Automated Updates of the Dataset

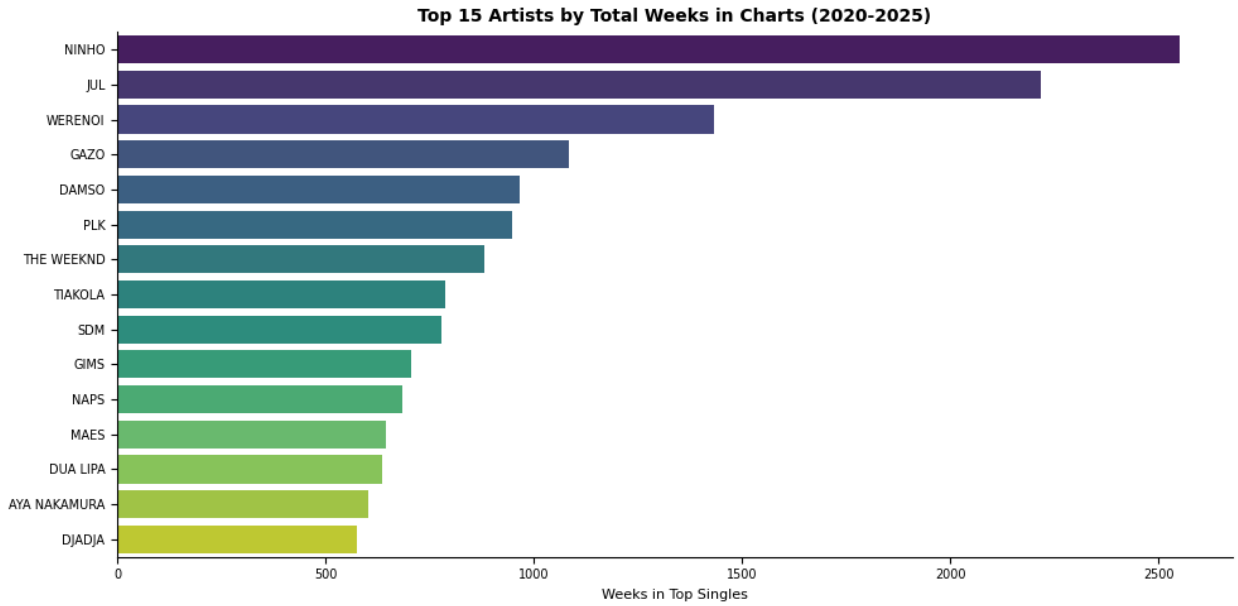
Finally, **I scheduled a daily Python job** (running every day at 11 a.m.) **with Airflow** that checks whether new chart data is available on the SNEP website. When new weeks appear, the script scrapes them, cleans them, and inserts them into the PostgreSQL database.

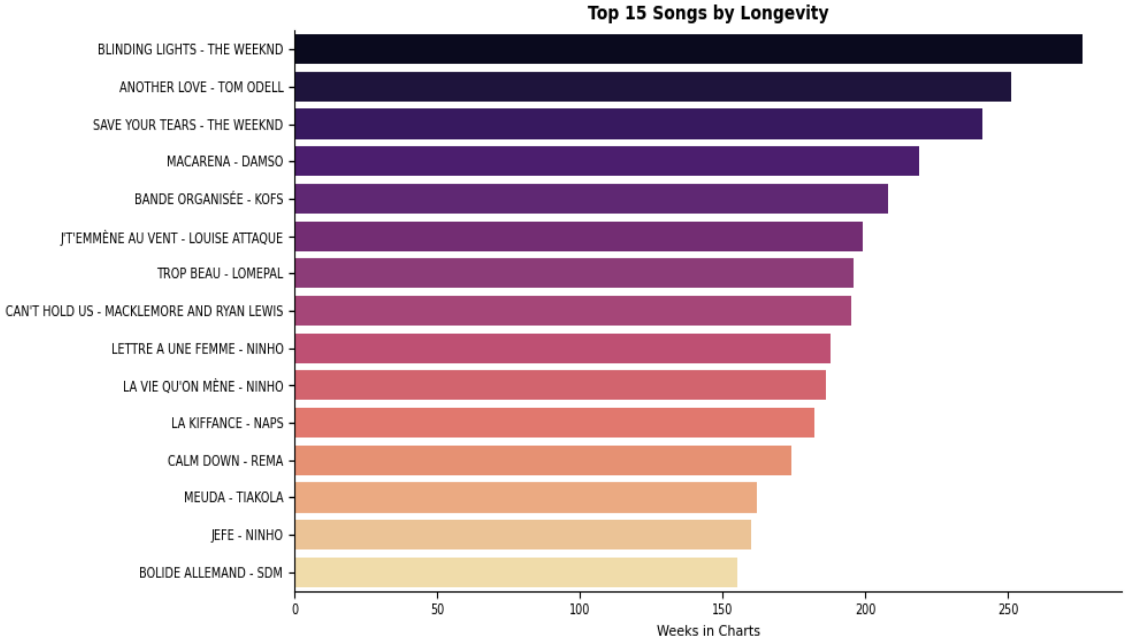
This automation ensures that the web app stays up to date without manual intervention and that headhunters and managers always see the latest information in the dashboard.

```python
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta
import os

default_args = {
    'owner': 'airflow',
    'description': 'Orchestrator for scraping and inserting SNEP data and Genius
API';start_date': datetime(2025, 11, 23),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    'catchup': False
}

with DAG(
    dag_id='snep_update_weekly',
    default_args=default_args,
    schedule_interval='0 11 * * *', # Every day at 11:00 AM
    catchup=False
) as dag:

    update_task = BashOperator(
        task_id='run_update_script',
        bash_command='cd /opt/airflow/project/scripts && python update.py',
        env={
            'TARGET_YEAR': '2025',
            'DB_HOST': os.getenv('DB_HOST', 'db'),
            'GENIUS_ACCESS_TOKEN': os.getenv('GENIUS_ACCESS_TOKEN')
        }
    )

    update_task
```

4. Exploratory data analysis

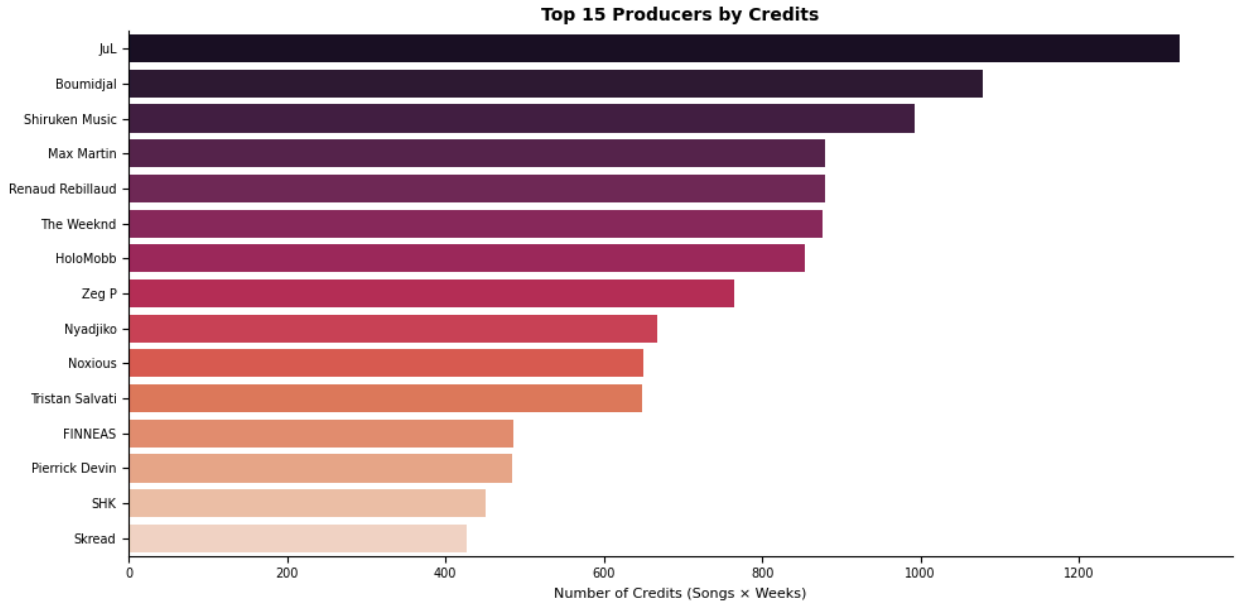
This chart shows the **top 15 artists in France by total number of weeks in the Top 200**. We can see that this ranking is largely dominated by French artists, mainly rappers. Only two foreign artists (The Weeknd and Dua Lipa) appear in this French Top 15.



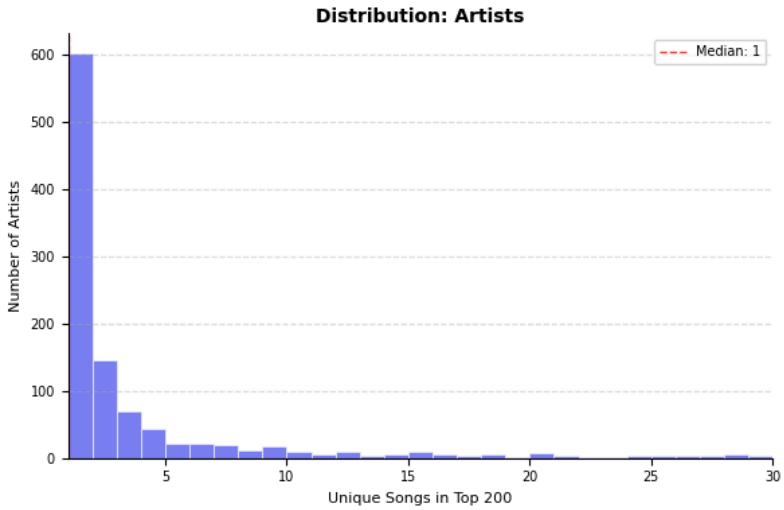


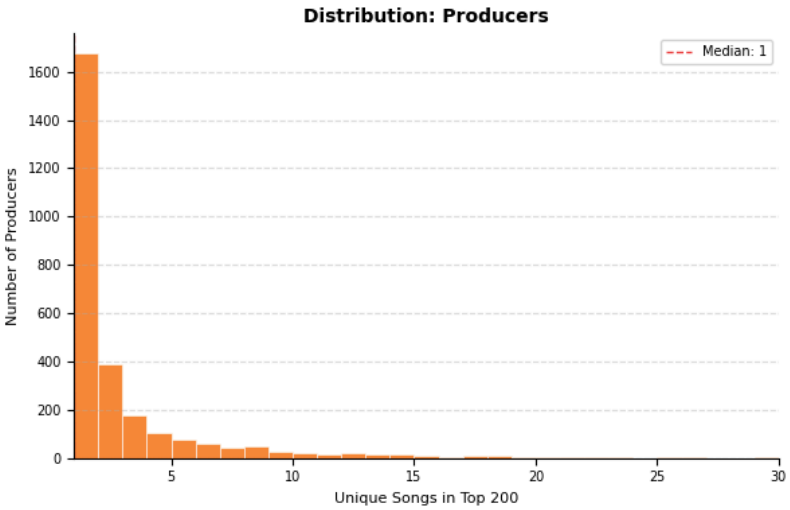
When we look at **single longevity** in the Top 200, the dominance of French artists is slightly less pronounced. Out of the 15 longest-lasting tracks, 10 are by French artists, but the top three longest-running singles are The Weeknd and Tom Odell.

The **top 15 producers** is very clearly dominated by **Jul, Boumidjal** and **Shiruken Music**.



We can also observe from these charts that the vast majority of artists **never have more than one single** entering the Top 200 over the last five years.





5. SQL Queries

Some example queries used to analyse the data directly in my databases:

1. **Data quality check (Artist cleaning)**
   This query checks whether any “feat” variations that were not correctly cleaned still appear in the artist column across all years.

```sql
SELECT source_year, artiste, titre
FROM (
    SELECT '2020' as source_year, artiste, titre FROM top_singles_2020
    UNION ALL SELECT '2021', artiste, titre FROM top_singles_2021
    UNION ALL SELECT '2022', artiste, titre FROM top_singles_2022
    UNION ALL SELECT '2023', artiste, titre FROM top_singles_2023
    UNION ALL SELECT '2024', artiste, titre FROM top_singles_2024
    UNION ALL SELECT '2025', artiste, titre FROM top_singles_2025
) as all_data
WHERE artiste ILIKE '% feat %'
   OR artiste ILIKE '% ft %'
   OR artiste ILIKE '% ft. %'
   OR artiste ILIKE '% feat. %'
   OR artiste ILIKE '%&%';
```

2. **Track with the greatest longevity (Record number of weeks)**
   Counts the total number of appearances of each (Title + Artist) pair across all years.

```sql
SELECT titre, artiste, COUNT(*) as semaines_presence
FROM (
    SELECT titre, artiste FROM top_singles_2020
    UNION ALL SELECT titre, artiste FROM top_singles_2021
    UNION ALL SELECT titre, artiste FROM top_singles_2022
    UNION ALL SELECT titre, artiste FROM top_singles_2023
    UNION ALL SELECT titre, artiste FROM top_singles_2024
    UNION ALL SELECT titre, artiste FROM top_singles_2025
) as all_data
GROUP BY titre, artiste
ORDER BY semaines_presence DESC
LIMIT 1;
```

3. **Most present artist (All tracks combined)**
   Sums all weeks in the Top 200 for all tracks associated with a given artist.

```sql
SELECT artiste, COUNT(*) as total_semaines_charts
FROM (
    SELECT artiste FROM top_singles_2020
    UNION ALL SELECT artiste FROM top_singles_2021
    UNION ALL SELECT artiste FROM top_singles_2022
    UNION ALL SELECT artiste FROM top_singles_2023
    UNION ALL SELECT artiste FROM top_singles_2024
    UNION ALL SELECT artiste FROM top_singles_2025
) as all_data
GROUP BY artiste
ORDER BY total_semaines_charts DESC
LIMIT 1;
```

4. **Most popular beatmaker in 2025**
   Looks only at the top_singles_2025 table and combines the producer_1 and producer_2 columns.

```sql
SELECT producer, COUNT(*) as semaines_presence
FROM (
    SELECT producer_1 as producer FROM top_singles_2025 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL
    SELECT producer_2 as producer FROM top_singles_2025 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
) as producers_2025
GROUP BY producer
ORDER BY semaines_presence DESC
LIMIT 1;
```

5. **Most present beatmaker since 2020 (full history)**
   Combines the first and second producer columns across all yearly tables to find the beatmaker with the highest cumulative presence.

```
WITH all_producers AS (
    -- 2020
    SELECT producer_1 as producer FROM top_singles_2020 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2020 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2021
    UNION ALL SELECT producer_1 FROM top_singles_2021 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2021 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2022
    UNION ALL SELECT producer_1 FROM top_singles_2022 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2022 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2023
    UNION ALL SELECT producer_1 FROM top_singles_2023 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2023 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2024
    UNION ALL SELECT producer_1 FROM top_singles_2024 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2024 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2025
    UNION ALL SELECT producer_1 FROM top_singles_2025 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2025 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
)
SELECT producer, COUNT(*) as total_credits
FROM all_WITH all_producers AS (
    -- 2020
    SELECT producer_1 as producer FROM top_singles_2020 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2020 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2021
    UNION ALL SELECT producer_1 FROM top_singles_2021 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2021 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2022
    UNION ALL SELECT producer_1 FROM top_singles_2022 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2022 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2023
    UNION ALL SELECT producer_1 FROM top_singles_2023 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2023 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2024
    UNION ALL SELECT producer_1 FROM top_singles_2024 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2024 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
    -- 2025
    UNION ALL SELECT producer_1 FROM top_singles_2025 WHERE producer_1 IS NOT NULL AND
producer_1 != ''
    UNION ALL SELECT producer_2 FROM top_singles_2025 WHERE producer_2 IS NOT NULL AND
producer_2 != ''
)
SELECT producer, COUNT(*) as total_credits
FROM all_
```

6. Exposing the Stored Data via a Flask API

To make the dataset usable by other applications and not only by my own dashboard, I built a small REST API in Flask, called **SNEP Analytics API**. The goal is to offer a simple, documented way to query historical SNEP Top 200 data (2020–present) for both artists and producers/beatmakers.

6.1 Architecture and Launch

The API is implemented with **Flask** and uses:

- flask and flask-cors for the web server and CORS handling,
- psycopg2-binary to connect to the PostgreSQL database,
- python-dotenv to load credentials from environment variables,
- configuration and dependencies listed in requirements.txt.

To start the API, the user must:

1. Navigate to the project root.
2. Install the dependencies:
3. Launch the Flask app

By default, the API is then available at:
http://localhost:5001

6.2 Main Endpoints and How to Query Them

The API exposes a small number of focused endpoints.

1. Search for an artist or producer

URL: /api/artist/<name>
Method: GET

This endpoint returns:

- basic information on the entity (artist or producer),
- a list of tracks found in the charts,
- statistics such as number of appearances and total weeks in the Top 200.

It accepts an optional query parameter:

- type:
  - artist (default),
  - producer (to search by beatmaker / producer name).

Examples

- Producer (note the URL encoding of spaces as %20):

```
curl "http://localhost:5001/api/artist/Maximum%20Beats?type=producer"
```

Output:


```
{
  "artist": "Maximum Beats",
  "total_songs": 18,
  "songs": [
    { "titre": "SPIDER", "best_rank": 1, "weeks_in_top": 80 },
    { "titre": "NINAO", "best_rank": 1, "weeks_in_top": 39 },
    { "titre": "CARRÉ OK", "best_rank": 19, "weeks_in_top": 33 },
    { "titre": "PIANO", "best_rank": 2, "weeks_in_top": 29 },
    { "titre": "APPELLE TA COPINE", "best_rank": 6, "weeks_in_top": 26 },
    { "titre": "AIR FORCE BLANCHE", "best_rank": 1, "weeks_in_top": 22 },
    { "titre": "TOUCHÉ", "best_rank": 18, "weeks_in_top": 22 },
    { "titre": "DIANA", "best_rank": 10, "weeks_in_top": 19 },
    { "titre": "PARISIENNE", "best_rank": 1, "weeks_in_top": 16 },
    { "titre": "TU ME RENDS BÊTE", "best_rank": 3, "weeks_in_top": 14 },
    { "titre": "APRÈS-VOUS MADAME", "best_rank": 74, "weeks_in_top": 14 },
    { "titre": "BABY", "best_rank": 15, "weeks_in_top": 13 },
    { "titre": "SENTIMENTAL", "best_rank": 60, "weeks_in_top": 6 },
    { "titre": "CONTACT", "best_rank": 71, "weeks_in_top": 5 },
    { "titre": "VENT DU NORD", "best_rank": 38, "weeks_in_top": 4 },
    { "titre": "GARANTIE", "best_rank": 32, "weeks_in_top": 3 },
    { "titre": "BLINDÉ COMME UN TANK", "best_rank": 118, "weeks_in_top": 1 },
    { "titre": "TANA", "best_rank": 148, "weeks_in_top": 1 }
  ]
}
```

From a user point of view, the steps are:

1. **Start the Flask API** (as shown above).
2. **Build the URL**
    - base endpoint: http://localhost:5001/api/artist/<NAME>
    - add ?type=producer if the search target is a producer/beatmaker,
    - replace spaces in the name with %20.
1. **Execute the request**
    - either from a terminal using curl,
    - or directly in a browser by pasting the URL.

The response is returned in **JSON format**, ready to be consumed by other services or visualisation tools.

2. Privacy information (GDPR)

URL: /api/privacy
Method: GET

This endpoint returns a human-readable description of how data is used in the project and a reminder of user rights (transparency, contact details of the data controller, etc.).

3. Data export (portability)

URL: /api/gdpr/export/<name>
Method: GET

This endpoint generates a complete export, in JSON, of all data associated with a given name (artist or producer). The export is structured and machine-readable so that it can be reused or transferred.

7. GDPR Note and Privacy Considerations

Even though the project only uses **public chart data** (no personal or sensitive data about listeners), I chose to align the API with the main principles of the **General Data Protection Regulation (GDPR)**.

The design addresses several key aspects:

1. **Transparency and right to be informed**
    - The /api/privacy endpoint explains what kind of data is processed (public SNEP chart data enriched with public credits), for what purpose (statistical analysis and visualisation), and who is responsible for the project.
1. **Right of access**
    - Any user can query the /api/artist/<name> endpoint to see what information is stored about a given artist or producer (within the scope of the public charts dataset).
1. **Right to data portability**
    - The /api/gdpr/export/<name> endpoint provides a complete export of all records related to that person in a structured JSON format, which can be reused or transferred to another system.
1. **Data minimisation**
    - The API only returns fields strictly necessary for music analysis (title, ranking, number of weeks, artist/producer names, etc.).
    - No sensitive information (private life, contact details, accounts, IP addresses, etc.) is collected or exposed.
1. **Right to erasure and rectification**
    - As stated in the privacy endpoint, any request for deletion or correction must be sent to the administrator of the project. The API itself is **read-only**; modifications are made directly at the database level by the data owner (acting as DPO for this project).
1. **Security**
    - The API only performs **read operations** on the database; there are no write endpoints exposed.
    - Database credentials and other secrets are managed via environment variables (.env) and are never hard-coded in the repository.

Overall, even though the legal risk is limited (public data only), this design shows how a small analytics API can still respect GDPR principles in terms of transparency, access, portability and minimisation.

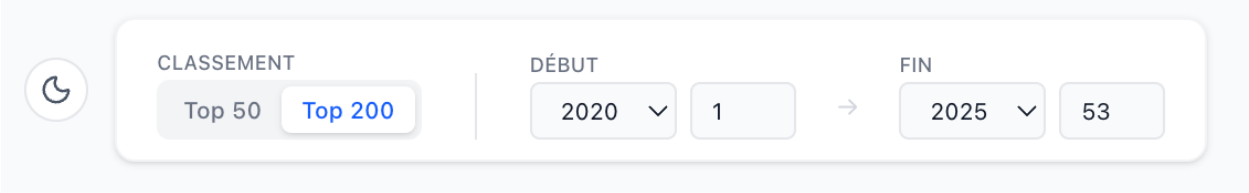
8. How the application works

The application is built with **Next.js** for the frontend and uses a **Python + PostgreSQL** stack on the backend for data access and processing.

Filters and analysis period

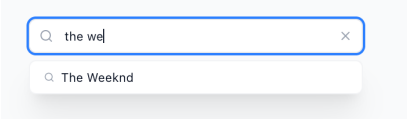
At the top right of the interface, a filter panel lets the user:

- choose the **time period** to analyse (for example: a specific year, the last 12 months, or the full range since 2020),
- select whether to work on the **Top 50** or the **Top 200**.

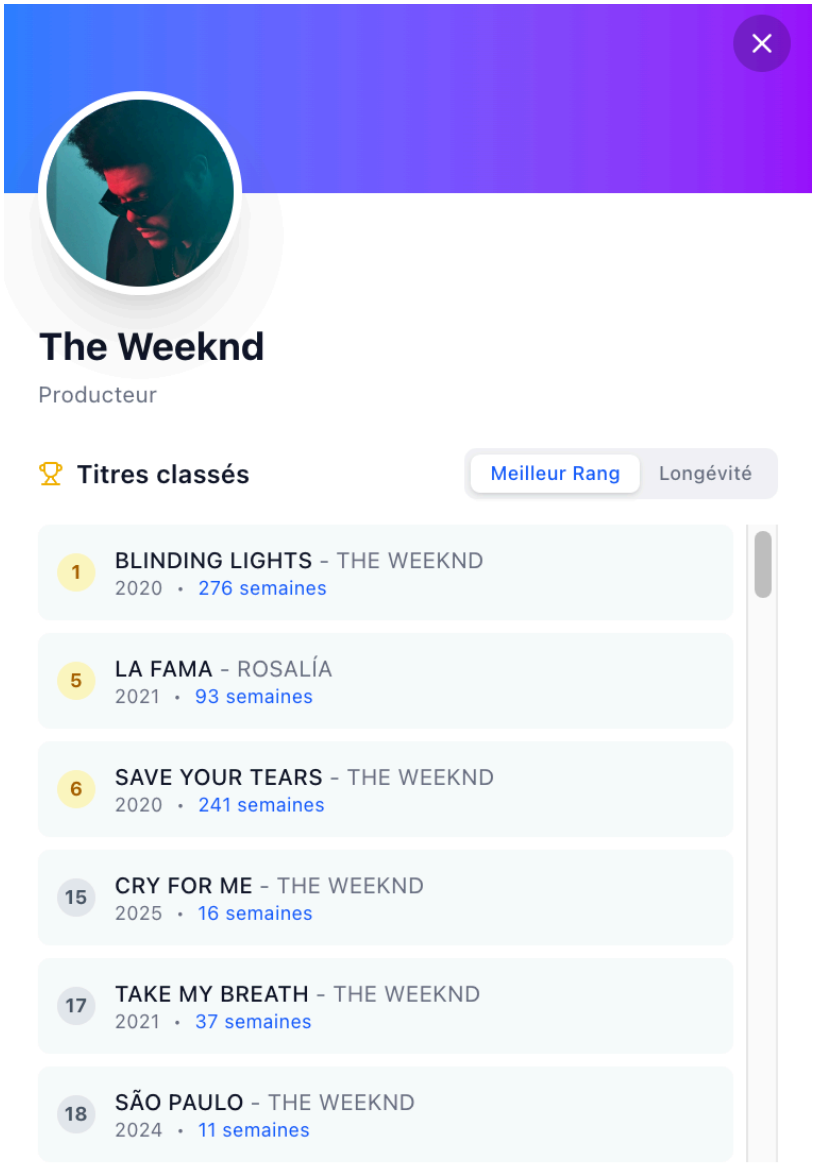


All the charts and rankings on the page are updated automatically when these filters change.

Search functionality



A **search bar** allows the user to look up any artist who has appeared in the Top 200 over the last five years.



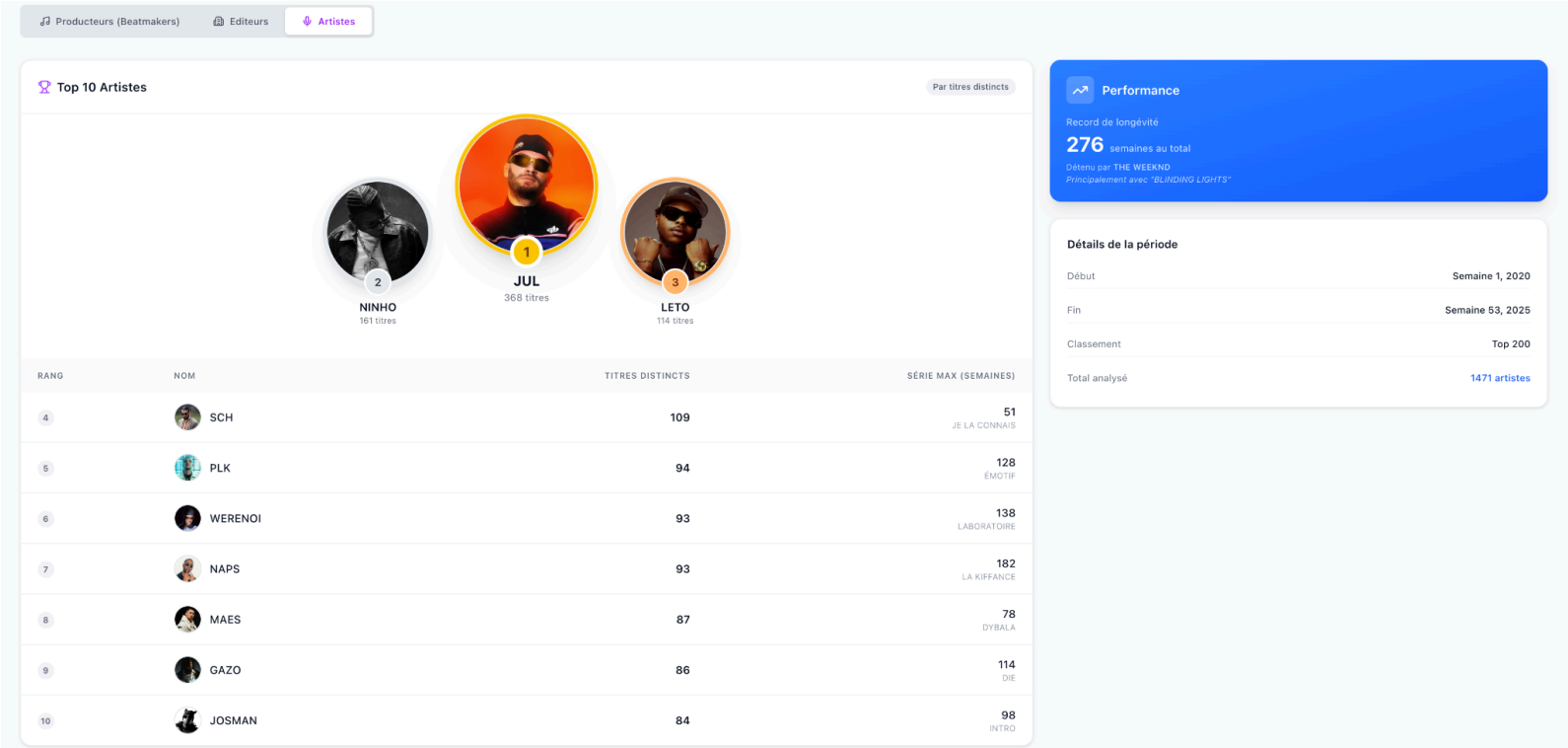
Main dashboard (Top 10 view)

The central part of the page displays the **Top 10** for the selected dimension:

- **artists**,
- **beatmakers/producers**,
- or **publishers**.

This Top 10 is computed over the currently selected period, based on the number of appearances in the Top 200.

Alongside this ranking, a **KPI card** continuously highlights the **single with the highest number of weeks in the Top 200** for the chosen period. This gives an immediate indication of the most persistent hit in the dataset.



Link to the web app: (available until the end of December):
http://172.233.243.159:3000/

Github:
https://github.com/Camil444/top-single-snep

Scrapping:

- Snep:
  https://snepmusique.com/les-tops/le-top-de-la-semaine/top-albums/?categorie=Top%20Singles&semaine=47&annee=2025

- Genius AP:
  https://docs.genius.com/