Music Metadata Service

1. Overview

1.1 Purpose

The **Music Metadata Service** is a microservice responsible for managing and serving metadata about music tracks, artists, and their relationships. It provides APIs for CRUD operations, paginated queries, and featured artist functionality, serving as the backbone for a music streaming platform (e.g., Spotify-like systems).

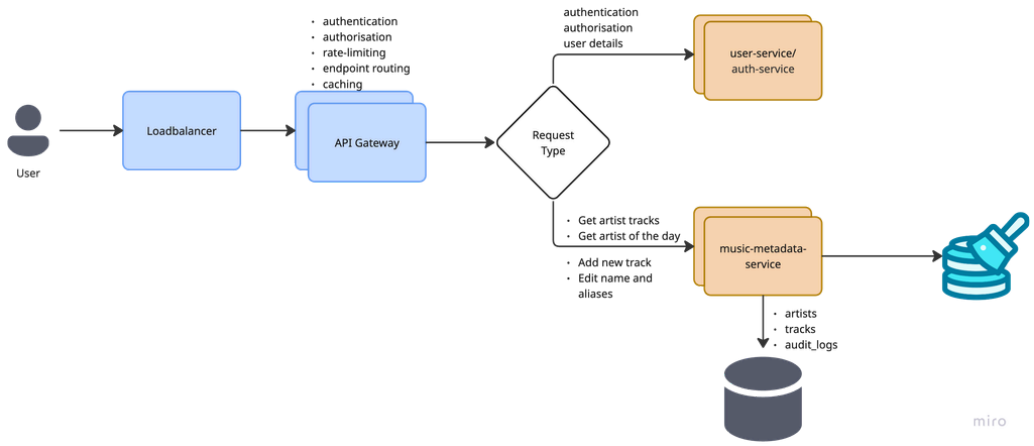
1.2 Scope

- Store and retrieve **track/artist metadata** (title, genre, duration, aliases).
- Support **pagination/filtering** for large datasets.
- Rotate a daily **"Artist of the Day"** fairly across the catalog.
- Cache for faster lookup

1.3 Out of Scope

- Audio file storage/streaming.
- User account management
- User authentication and authorisation(RBAC)
- Infrastructure

2. System Architecture



2.2 Components

Component	Description
ArtistsAPI	• Manages artist metadata (name, aliases). • Track-related operations (paginated queries)
Database	H2 with tables for `tracks`, `artists`, and `artist_aliases`. `prod: use Postgresql`
Scheduler	Rotates the "Artist of the Day" after 24h `quartz`

	`prod: use distributed event-store`
Cache	`Caffeine` caching for high-read endpoints (e.g., "Artist of the Day", list of `artist` tracks). `prod: use Redis`

3. API Specifications

The design follows the api-first approach by defining and working backwards from an open-api specification.

3.1 Endpoints

Update Artist Name

Endpoint	Method	Description
`/v1/artists/{id}/name`	GET	Update artist name and aliases

```
GET: /v1/artists/{id}/name
{
  name: ""
  duration: ""
}
200 OK
```

Add Tracks to Artist

Endpoint	Method	Description	Auth
`/v1/artists/{id}/tracks`	PUT	add tracks to artist	Admin

```
PATCH: /v1/admin/artists/{id}
Request:
{
  name: ""
  aliases:[]
}
Response:
204 No Content
```

Artist Tracks

Endpoint	Method	Description
`/v1/artists/{id}/tracks?page={page}& size={size}`	GET	Fetch tracks (paginated, filterable)

```
GET: /api/tracks?artistId={id}
{
  trackId : ""
  name: ""
  duration: ""
}
```

```
200 OK
```

Artist of the day

Endpoint	Method	Description
`/v1/artist/artist-of-the-day`	GET	Get today's featured artist

```
GET: /api/featured/artist-of-the-day
{
  artistId: ""
  name: ""
  aliases:[
  ]
}
```

4. Data Models

4.1 Database Schema

Table: `artists`

Field	Type	Description	Constraints
`id`	BIGINT	**Internal PK** (auto-increment)	`PRIMARY KEY, AUTO_INCREMENT`
`artist_id`	UUID	**Public-facing unique ID**	`UNIQUE, NOT NULL, DEFAULT uuid_generate_v4()`
`name`	VARCHAR(255)	Current artist name	`NOT NULL`
`created_at`	TIMESTAMP	Record creation time	`DEFAULT CURRENT_TIMESTAMP`
`updated_at`	TIMESTAMP	Last update time	`ON UPDATE CURRENT_TIMESTAMP`

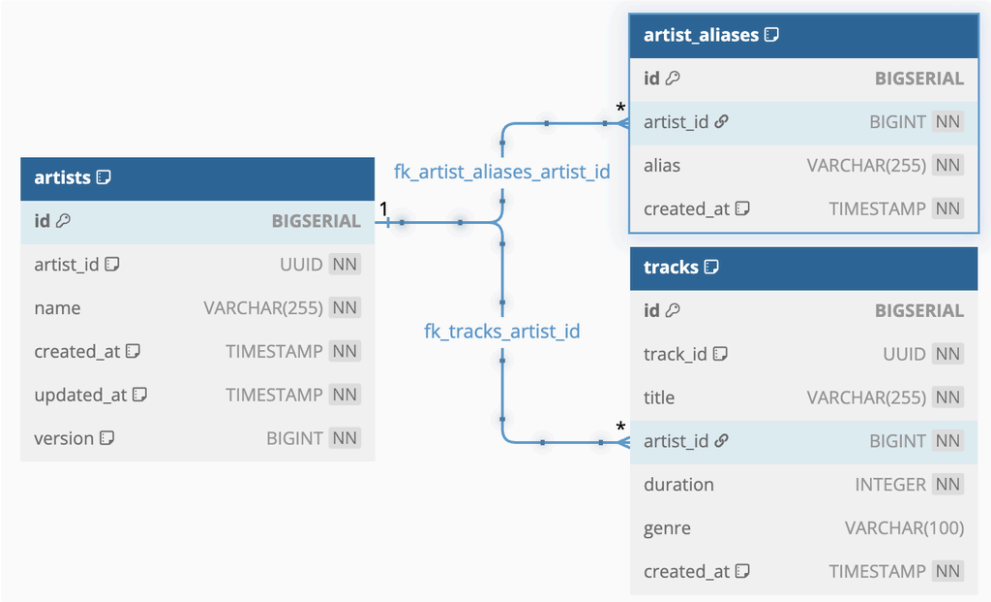
Table: `tracks`

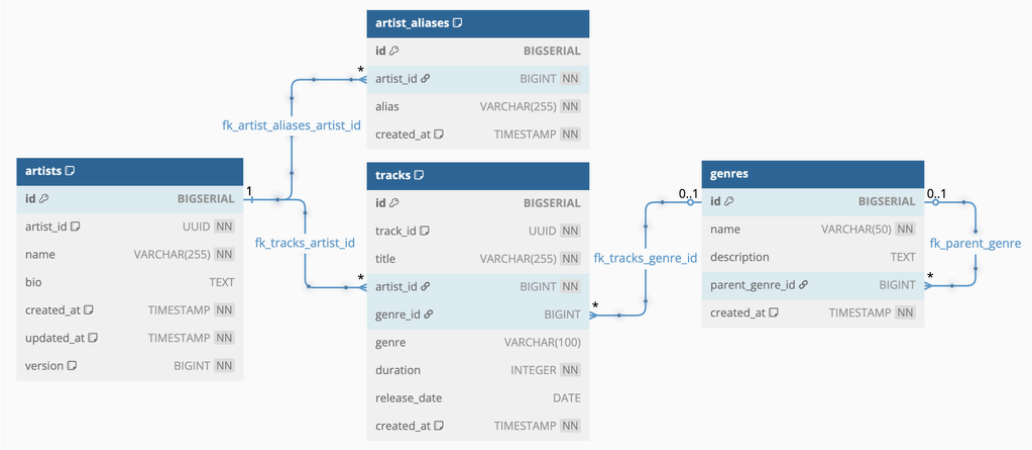
Field	Type	Description	Constraints
`id`	BIGINT	**Internal PK** (auto-increment)	`PRIMARY KEY, AUTO_INCREMENT`
`track_id`	UUID	**Public-facing unique ID**	`UNIQUE, NOT NULL, DEFAULT uuid_generate_v4()`
`title`	VARCHAR(255)	Track title	`NOT NULL`
`artist_id`	BIGINT	**FK to** `artists.id` (internal)	`NOT NULL`
`duration`	INT	Duration in seconds	`NOT NULL`
`genre`	VARCHAR(100)	Track genre	
`created_at`	TIMESTAMP	Record creation time	`DEFAULT CURRENT_TIMESTAMP`

Table: `artist_aliases`

Field	Type	Description	Constraints
`id`	BIGINT	**Internal PK** (auto-increment)	`PRIMARY KEY, AUTO_INCREMENT`
`artist_id`	BIGINT	**FK to** `artists.id` (internal)	`NOT NULL`
`alias`	VARCHAR(255)	Historical/alternate name	`NOT NULL`

4.1 ER Diagram





5. Business Logic

5.1 Key Features

Artist of the Day

- **Algorithm**: Rotates artists in a fixed cycle (e.g., ordered by `artist.id` or `artist.created_at` and `artist.featuredAt`).
- **Con Job:**

    - InMemory : Use `spring quartz`
    - Distributed Cache:
        - Dedicated EventStore Service
        - Jobrunr : uses database synchronisation
        - Infrastructure Event Trigger e.g AWS EventBridge to trigger a scheduled event
- **Caching**: caches the daily artist to reduce DB load.
    - InMemory Cache: Use `caffeine`
    - Distributed Cache: Use `Redis`
- **Fairness**: Resets cycle after all artists are featured.

Pagination

- **Default**: 20 items/page, customizable via `?page` and `?size`.
- **Sorting**: Optional `?sort=title,asc`.

---

6. Security

- **Authentication**: JWT tokens (OAuth2).
- **Authorization**:
    - Admin endpoints: Require `role:admin`.
    - Public endpoints: Read-only.
- **Data Validation**: Input sanitization (e.g., no SQL injection).

---

7. Deployment

- **Infrastructure**: Kubernetes (EKS/GKE) with auto-scaling.
- **CI/CD**: GitHub Actions (build → test → deploy).
- **Monitoring**: Prometheus (metrics) + ELK (logs).
