BOATYARD
EVERYBODY ON BOARD

Ruby on Rails Developer - Coding Challenge
Music Service API

PROJECT DESCRIPTION

You are creating a simple API as part of a music streaming service. The API is composed of 3 main entities (`Albums`, `Artists` and `Songs`) and must implement the following workflow:

1. The process starts by creating single albums in **individual requests**. Albums will be containers for songs.
2. Songs are created in **individual requests** with embedded **albums** and **artists** related to each song.
3. Finally, queries like the following could be performed:
    a. List albums with their songs and artists
    b. List songs with their albums and artists
    c. List artists with their songs and albums
    d. List artist balances

ENTITIES STRUCTURE

ALBUMS

`name`	`year`	`album_art`	`total_duration`	`cached_songs`
Album name. String, required	Album release year. Integer, required	URL to an external image. String, optional	Duration of all the album songs. Integer, required, default 0	For performance purposes. Serialized hash, required, default {}

SONGS

`name`	`duration`	`genre`	`streams`
Song name. String, required	Duration in seconds. Integer, required	Song genre. String, required.	Total number of song streams. Integer, required, default 0

`genre` attribute must be one of: Alternative Rock, Blues, Classical, Country, Electronic, Funk, Heavy Metal, Hip-Hop, Jazz, Pop, Reggae, Soul, Rock

ARTISTS

`name`	`biography`
Artist name. String, required	Short artist biography. String, required.

ARTIST BALANCES

`artist_id`	`balance`
Reference to artist. Integer, required	Balance in **cents**. Float, required

OPERATIONS BY ENDPOINT

ALBUMS

CREATE ALBUM
Albums are created with at least the required attributes described in the Entities Structure section.

UPDATE ALBUM
Alongside required album data, it must be possible to send in the same request a combination of:

- A set of existing *song ids* to associate with the album
- A set of existing *song ids* to remove from the album

Important

The **artists** of an **album** must be computed from the **songs** the album contains. If new songs are added or removed from the album, album artists must also be updated.
A similar rule goes for`album.total_duration`attribute, it must be consistent with the duration of all the album songs.

Example:
You create the album: *The Marshall Mathers.* Then you create the song *Stan* whose duration is 404 seconds and its artists are *Eminem* and *Dido*, and attach the song to the album. Then the **album artists** are *Eminem* and *Dido* and the album total duration is 404 seconds.

SHOW ALBUM
Display full details of a single album, including all its **songs** and **artists**.

DELETE ALBUM
If after deleting the album, the associated songs are no longer related to other albums or artists, songs must be deleted.

SONGS

CREATE SONG
Songs are created with at least the required attributes described in the Entities Structure section and, for simplicity, songs will be the entry point to create artists through nested data.

Alongside required song data, it must be possible to send in **the same request** a combination of:

- A set of new *artist attributes* to create and associate with the new song
- A set of existing *artist ids* to associate with the new song
- A set of existing *album ids* to associate with the new song

Every new song created must:

1) Increment `album.total_duration` attribute in all associated albums, by the song's duration in seconds.

2) Update `album.cached_songs` attribute in all associated albums. This attribute must be a hash with the song ID as a **key** and a Hash with the following keys as a **value**:`name, duration, genre, artists`(comma separated value with all artists in the song)

Example:

You create 2 albums: *The Marshall Mathers* and *Curtain Call: The Hits.* Then you create the song *Stan* whose artists are *Eminem* and *Dido*, and specify that the song must be associated with the 2 albums you created in a previous request. Assuming that the song *Stan* was assigned the ID 1587, the generated `cached_songs` should look like:

```
Album.find_by(name: 'The Marshall Mathers').cached_songs
=> {
  1587 => { name: 'Stan', duration: 404, genre: 'Hip-Hop', artists: 'Eminem, Dido' }
}

Album.find_by(name: 'Curtain Call: The Hits').cached_songs
=> {
  1587 => { name: 'Stan', duration: 404, genre: 'Hip-Hop', artists: 'Eminem, Dido' }
}
```

UPDATE SONG
Alongside song data, it must be possible to send in the same request a combination of:

- A set of existing *album ids* to associate with the song
- A set of existing *album ids* to remove from the song
- A set of new *artist attributes* to create and associate with the song
- A set of existing *artist ids* to associate with the song
- A set of existing *artist ids* to remove from the song

DELETE SONG
A song cannot be deleted if it is associated with albums or artists, so all its relations must be removed first.

SHOW SONG
Display full details of a single song, including all its **artists** and the **albums** where it is present.

STREAM SONG
This endpoint will only render a JSON message like “`Streaming #{song.name}`” but internally it has to:

- Increment by 1 `song.streams` attribute
- Create a record in the `artist_balances` entity for **each artist** associated with the song, with the following rules:
    - If the song has up to 1000 streams, every new stream should add up a third of a penny (0.3) to the artist’s balance
    - If the song has more than 1000 streams, every new stream should add half of a penny (0.5) to the artist’s balance
    - There must be only 1 record per artist in the `artist_balances` entity

ARTISTS

UPDATE ARTIST
For simplicity sake, only base attributes will be accepted (`name` and `biography`). If artists must be removed from songs or albums, the operation will be performed through nested attributes in a song request, as explained in the UPDATE SONG endpoint above.

DELETE ARTIST
An artist cannot be deleted if it is associated with albums or songs, so all its relations must be removed first.

SHOW ARTIST

Display full details of a single artist, including all its **albums** and **songs**.

ARTIST BALANCE

Important: This endpoint is only visible if the authenticated user is an admin.

It must display the balance of a single artist in USD (not cents), based on the data stored in `artist_balances`

GENERAL CONSIDERATIONS

- The API must be RESTful, although GraphQL is also welcomed, and created from scratch using the Ruby on Rails framework.
- Ruby 3.0 is preferred but not required.
- The data should be stored in a DB of your choice, although Postgres is preferred.
- The code must be Rubocop compliant.
- The code must be published to GitHub to allow communication with your reviewer.
- All main features must be covered with RSpec tests.
- Implementation of *design patterns* is always preferred over *model callbacks*, but this is not a requirement for this challenge.
- The application must be deployed to be tested, and remember: *an API is as good as its documentation.*
- It is desirable to have at least a few records of seeded data to facilitate testing.
- It is desirable to be provided with Postman collections to facilitate testing.
- Make it as real as possible, e.g. considering authentication, data validation, DB indexes where needed, security of sensitive data, clean and organized code, etc.
- Keep it simple and ask all questions you need

TIME FRAME

The suggested time frame for delivery is 8 hs.

Have fun and use your creativity!