Shazam

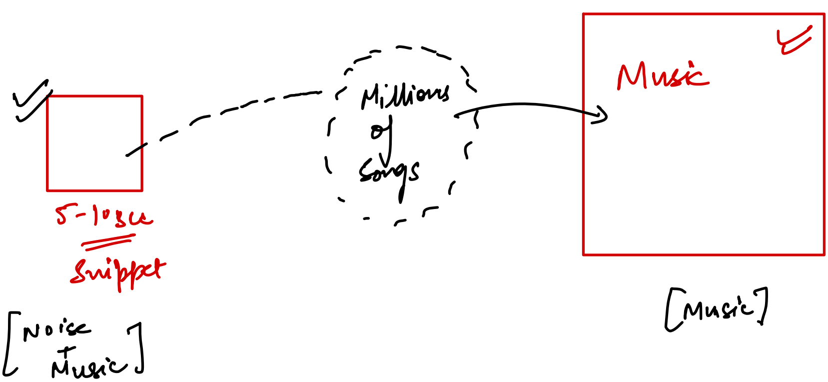
[+ system (Algorithm)]

+ system design [Basic]

https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

Core idea : Audio fingerprinting

→ unique fingerprint



Functional Requirements

1. Audio input → 5-10 sec audio (device microphone)
2. Song identification
3. Metadata → Artist, Album, lyrics, Release year
4. No-match
5. → Maintain history of User

Non-functional Requirements

1. Accuracy → High, noise + music
2. Latency → Identification process → fast
3. Scalability →
4. Availability (Uptime)

⋮

Data

1. Audio data → ______
   - → Snippet → (WAV, MP3)
   - [noisy, varying loudness, some parts]
   - Multiple songs can have some part common

2. Fingerprint data → ( )-
   - ↳ Sequence of hashes (specific audio at specific time)

→ User generated | Reference Songs.

Minor distortion
but sensitive enough to
distinguish diff songs.

③ Metadata
+

Process

Step 1 User records a raw audio

Audio needs cleaning

+ → Noise removal

Core Algo of Shazam is quite robust

https://www.cameronmacleod.com/blog/how-does-shazam-work

+ → Normalize
(Adjust the volume to standard level)

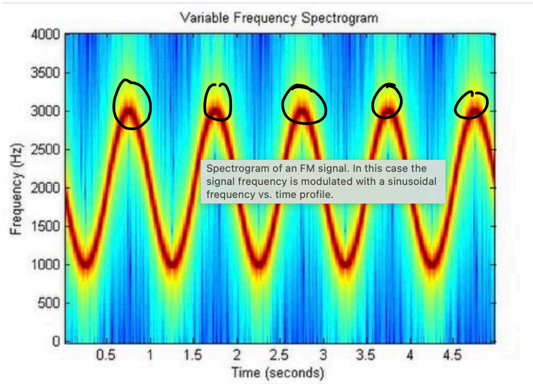
Reference Songs
↳

Step 2 Fourier Transform

+ → Deconstruct Noise

[Small chunks of audio } How loud]

Spectrogram ✓



Step 3 → Fingerprinting

[→ Landmarks / anchor points]

- → Identifying peaks
- → Create hashes

Peak 1 → F_1 T_1

Peak 2 → F_2 T_2

Delta → $T_2 - T_1$

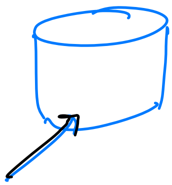
Save Absolute time T_1

Why pairs? → If one peak is slightly off, nearby peak should be able to help us in recognization.

Fingerprints

Song A fingerprint = [(hash1, time 1),
(hash 2, time 2),
(hash3, time 3)]

Step 4 Building database of fingerprints.



→ Song finger...

⇒ [hash, time-offsets] pair.

[indexed] → hash

finding songs quickly on the basis of hash

Database entry

hash1 → (Song 1, time offset)

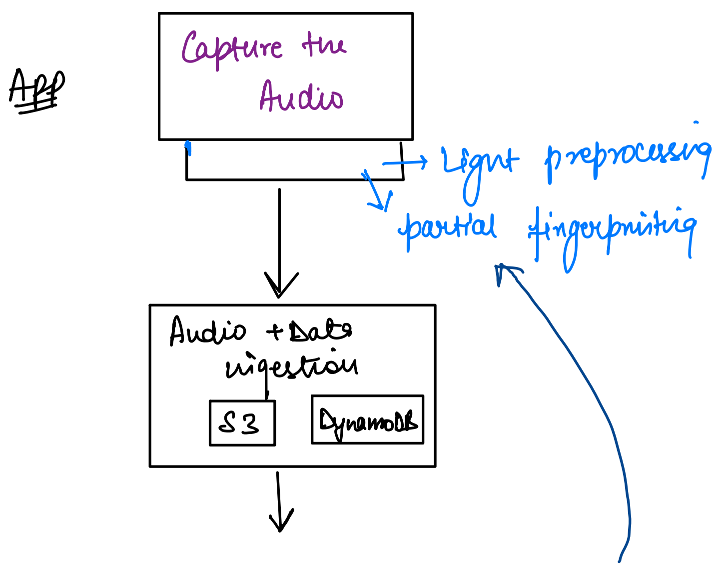
Step 5 Matching process

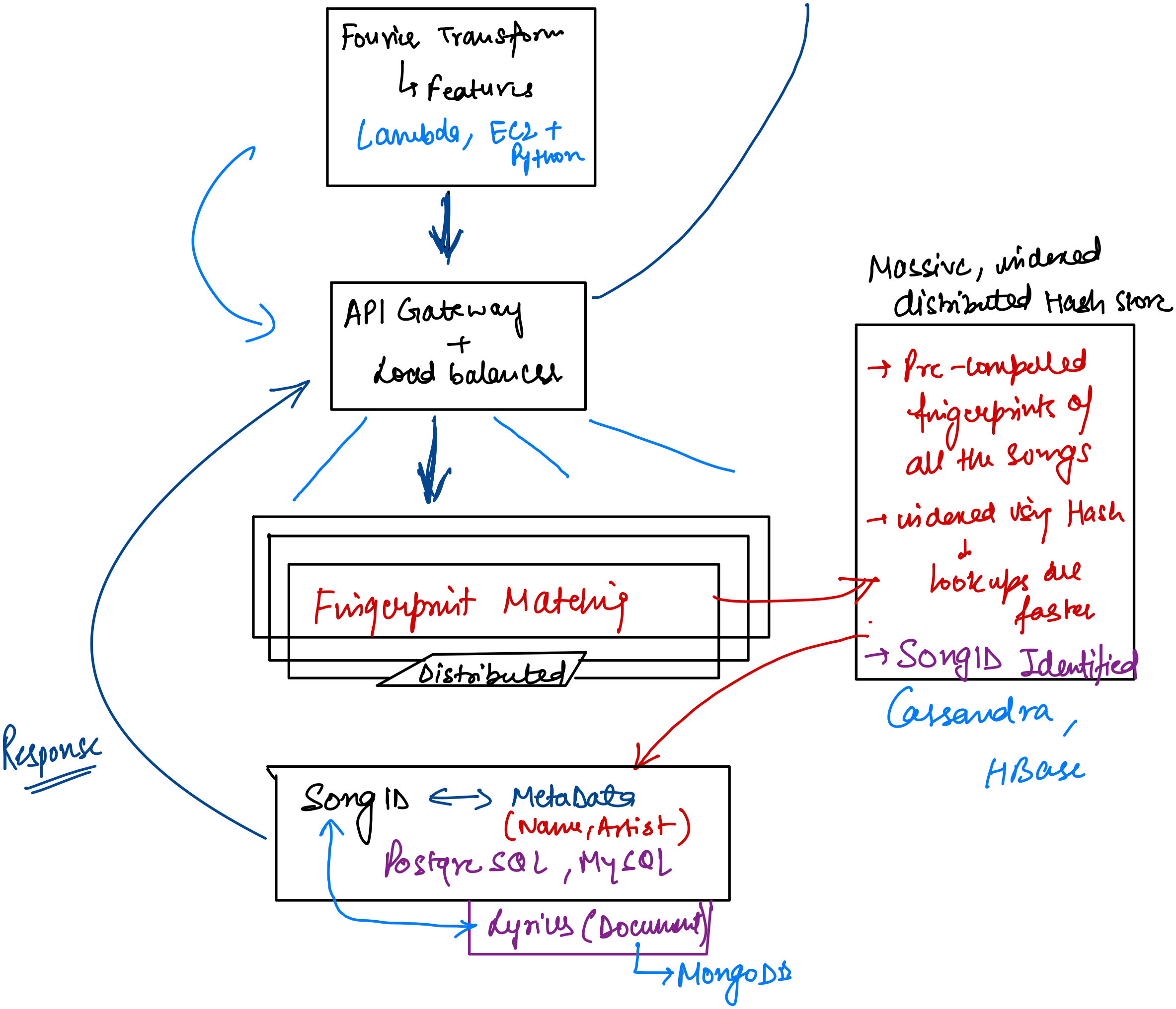
Recording matched 2 songs

- Let's say `snippet_hash_X` (which occurred at `time_X_in_snippet` in your recording) matches `hash_X_from_SongA` (which occurs at `time_offset_A1` in Song A) and also `hash_X_from_SongB` (which occurs at `time_offset_B1` in Song B).
- And `snippet_hash_Y` (at `time_Y_in_snippet`) matches `hash_Y_from_SongA` (at `time_offset_A2` in Song A) and `hash_Y_from_SongC` (at `time_offset_C1` in Song C).
- The system now looks for *consistent time differences*.
  - For each potential `(SongID, snippet_hash, time_in_snippet, time_in_song)` match, it calculates a "delta": `delta = time_in_song - time_in_snippet`.
  - If the snippet is indeed from Song A, then the `delta` calculated for `snippet_hash_X` matching a hash in Song A should be very similar to the `delta` calculated for `snippet_hash_Y` matching another hash in Song A. This `delta` essentially represents the offset of your recording within the original song.
- **Voting (Histogram Analysis):** The system creates a histogram (a bar chart) for each candidate song.
  - The x-axis of this histogram represents these calculated `delta` values (time offsets).
  - The y-axis represents how many matching hash pairs from the snippet and the candidate song produce that particular `delta`.
  - If many hash pairs from your snippet align with hash pairs in a specific song (say, Song A) and they all point to roughly the same `delta` (meaning they are consistently offset in time), then Song A will have a very tall peak in its histogram at that `delta`.
  - This peak signifies a strong match. The song with the most "votes" (highest peak in the histogram) is declared the winner.

Snippet Hash is User recorded song snippet

Architecture





Concept Used

- → Google Meet → AI Notetaker (User A, B, C, D) tags
  ↳ identify all the people in the meeting
- → Neosapien User identification through User's voice
- → Youtube plagiarism