UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

Relazione

WHATSTHEHIT

ESPLORA LA MUSICA NEL TEMPO

COSTRUZIONE BANCA DATI E WEB-API DI INTERROGAZIONE

Candidato: *Stefano Tambellini*

Relatore: *Andrea Marchetti*

Correlatore: *Mirko Tavosanis*

Anno Accademico 2017-2018

Sommario

Introduzione

WhatsTheHit è lo sviluppo di un'idea, nata durante un progetto d'esame collaborativo, in cui io e il mio gruppo ci eravamo posti l'obiettivo di analizzare come si è evoluta la musica nell'ultimo secolo.
Come facciamo a cercare quelle canzoni che hanno reso importante un'annata nel passato? I colossi informatici che siamo abituati a usare, che di fatto sono grossi canali di informazioni testuali e multimediali, mancano evidentemente di un'organizzazione basata sul tempo storico, attuale, di queste informazioni. Piattaforme come Youtube e Spotify catalogano le canzoni sulla base della data di caricamento sul sistema, ma non la vera data di pubblicazione.
Nessun sito ha tuttora come unico scopo quello di ordinare le canzoni in ordine di fama mondiale e al contempo permetterne l'ascolto immediato, il tutto tramite un'interfaccia facile e veloce da usare! L'alternativa è andare manualmente a cercare quali sono state le classifiche annuali su vari siti e spesso e volentieri occorre copiare e incollare titolo e artista in uno dei siti sopra citati.
La nostra soluzione a questo problema è WhatsTheHit, una web app che ha come unico obiettivo quello di riorganizzare e riprodurre in modo immediato le hit più famose dal 1900 ad oggi. Con questo strumento l'utente è in grado di compiere tutte le ricerche che vuole a partire dall'anno o dal nome di una canzone, per poi velocemente ricevere informazioni sull'artista e la possibilità di ascoltare la traccia o l'album in questione.
L'idea risultava già da subito avere un gran potenziale. Volevamo metterci alla prova e creare un sito in maniera professionale che avesse queste caratteristiche e, data la mole di lavoro che ci aspettava, abbiamo deciso di suddividerci il lavoro. Il mio compito è stato quello di sviluppare il lato back-end, che include sia la programmazione del server sia la gestione del database. In questo elaborato tratterò appunto di questa parte: di come ho progettato e programmato il lato server di WhatsTheHit.
Dal punto di vista pratico è stato un esercizio molto complesso di project management, organizzazione per lavoro condiviso e, ovviamente, di programmazione. Ho dovuto imparare da zero molti dei programmi, linguaggi e strumenti che ho utilizzato oltre a una gran parte di concetti informatici avanzati che fino ad ora non avevo mai approfondito. Dal punto di vista tecnico è stato invece molto interessante creare un ambiente di lavoro ad-hoc per questo progetto, sfruttando le ultime tecnologie in campo di programmazione back-end e collegando il tutto manualmente senza l'utilizzo di nessuno stack di programmi preimpostato.

Gli obiettivi del mio lavoro sono stati:

1. la progettazione e creazione di un database relazionale che contenga unicamente i dati delle canzoni, album e artisti che andranno utilizzati della facciata
2. la programmazione effettiva del server che fornirà la web-api di interrogazione del database, oltre al front-end stesso del sito
3. una documentazione che serva d'aiuto tecnico e sintattico alla compilazione effettiva delle richieste di dati alla web-api

A ognuno di questi punti ho dedicato un capitolo nell'elaborato.

CAPITOLO 1

CREAZIONE DEL DATABASE

Il primo passo del mio lavoro è stato quello di raccogliere i dati che mi interessavano per poi renderli utilizzabili dal server. L'obiettivo di questa fase è quello di **racimolare i dati che ci servono e inserirli all'interno di un struttura dati relazionale appositamente progettata**.

Fonte dei dati utilizzati

Per raccogliere i dati, inizialmente ci siamo concentrati su fonti di dati nazionali, a partire dall'Italia con ISTAT e SIAE. Era interessante un approccio di questo tipo perché ci avrebbe dato modo di considerare anche l'informazione della provenienza di una canzone.

Purtroppo, la scarsezza di risultati incontrata non si adattava bene alle necessità del progetto. Inversamente, gestori giganti di dati musicali come Discogs o MusicBrainz fornivano una mole così grande di dati che risultava molto difficile da utilizzare e che, molto spesso, nemmeno forniva un modello dati di tipo relazionale.

Fortunatamente, `tsort.info` si è rivelato un buon compromesso. Il sito fornisce liberamente, sottoforma di file CSV, tutte le più famose canzoni dal 1900 e tutti i più famosi album dal 1946 in poi. La tabella totale include più di 600.000 record, ai quali viene associato:

- Un valore relativo "`score`" per ogni record, che viene calcolato dai fornitori dei dati sulla base di più informazioni tra cui:
    - le posizioni nelle classifiche nazionali di riviste tra America, Europa e Asia
    - i dati di vendita di un record
    - quanto il titolo della canzone/album comparisse su articoli e testi
- L'anno di maggior fama della pubblicazione, che diventa un dato più importante dell'anno effettivo di uscita nel momento in cui si vuole fare un'analisi di tipo socioculturale

Trattamento dei dati

I vari file forniti dalla fonte erano molto lontani dal poter essere direttamente utilizzati: molte colonne risultavano prevalentemente vuote e mal formattate.

I dati sono stati quindi ripuliti utilizzando *OpenRefine*, un programma open-source che permette di compiere velocemente modifiche a grosse moli di dati in formato relazionale.
Il database così ripulito era utilizzabile ma presentava i seguenti problemi:

- Il formato della struttura era a singola tabella, il che creava problemi di gestione e valori `null`
- Ridondanza dei dati nella colonna artista, che erano ripetuti per ogni canzone
- L'aggiunta e la modifica dei dati era aggravata da questa struttura

Per ovviare a questi problemi, ho deciso di normalizzare il database, creando da zero una nuova architettura che più si adattava alle esigenze dell'applicazione.

Requisiti

Prima di cominciare la progettazione occorre definire quali saranno i requisiti della nuova base di dati. La nuova struttura dovrà, oltre a risolvere i problemi identificati in precedenza, avere nuove caratteristiche che permettano un uso più logico ed efficiente.

DIVISIONE DELLE DIVERSE ENTITÀ

L'architettura a tabella unica, oltre a rendere inutile l'utilizzo di un database relazionale, alla lunga avrebbe creato grandi problemi di manutenzione, ampliamento ed efficienza. Nel momento in cui si volesse andare a modificare la struttura dati, ad esempio aggiungendo una nuova colonna attributo, sarebbe necessario aggiungere alle righe non interessate da questo cambiamento dei valori nulli, che sono una grande fonte di confusione in un sistema relazionale.
È stato quindi essenziale definire un modello dati in grado di gestire indipendentemente ogni diversa entità. Le diverse entità sono principalmente 3: artista, album e canzone.

AGGIUNTA DI RELAZIONI N-N

Per risolvere la ridondanza della colonna artista occorreva creare una relazione uno a molti tra l'artista e la canzone/album. Volevo inoltre permettere di considerare il *featuring*, cioè la produzione contemporanea di una canzone da parte di due o più artisti.
Ho quindi stabilito che la base di dati dovesse avere una relazione molti a molti tra artista e i vari prodotti.

Progettazione del nuovo database

C'è adesso il bisogno di fare un piano generale sul funzionamento e l'architettura del prodotto che dovrò creare. Sulla base dei requisiti definiti, progetto il database rispettando tutti i vari vincoli e cercando il più possibile di risolvere eventuali problematiche sin da subito.

PROGETTAZIONE CONCETTUALE

Lo scopo di questa prima fase è quello di formalizzare la struttura dati seguendo i requisiti definiti in maniera totalmente indipendente dalla rappresentazione nei DBMS. Mi approccerò alla progettazione concettuale utilizzando, data la mole relativamente piccola di entità, una strategia "mista", nella quale definisco uno schema minimale che vado piano piano ad ampliare senza un ordine preciso.
Si vuole realizzare una base di dati per una applicazione web che fornisca una lista di canzoni sulla base delle preferenze di selezione.
Individuiamo e descriviamo le 3 entità chiave della nostra rappresentazione correlate dagli attributi:

- **Canzone**: Occorre definire il titolo, artista che la esegue, l'anno di produzione e il punteggio che ne determina la posizione sulla classifica
- **Album**: Stessi identici elementi della canzone
- **Artista**: Basta semplicemente definire il nome, non c'è bisogno di dati anagrafici

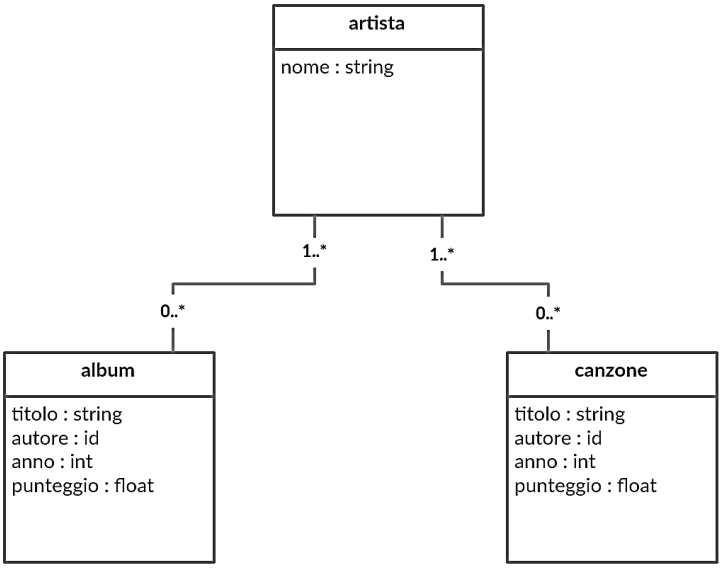
Individuiamo inoltre le associazioni tra queste entità:

- **Artista – Canzone**: L'artista produce una canzone, che gli viene accreditata
- **Artista – Album**: L'artista produce un album, che gli viene accreditato

Ho inoltre creato uno script, incluso nel progetto nella cartella `/script/testing_assocazioni`, nel quale ho cercato di capire quanto effettivamente fosse possibile creare l'associazione album-canzone. Interrogando l'API di MusicBrainz in questo script ho rilevato che nel database:

- Non c'è abbastanza correlazione tra gli album più venduti e famosi e le canzoni più vendute e famose
- Non è possibile trovare un album a partire soltanto dalle informazioni della canzone per mancanza di dati

Questo è lo schema concettuale risultante alla fine di questa fase di progettazione:



PROGETTAZIONE LOGICA

In questa fase tradurrò lo schema concettuale sulla base del modello di rappresentazione dei dati che ho intenzione di adottare, ovvero un database relazionale.

A partire dallo schema concettuale vado passo-passo a trasformare ogni entità e relazione in una definizione logica. Aggiungo un identificatore univoco id che uso come chiave primaria.

A questo punto lo schema logico si presenta così:

Canzone(id, titolo, anno, punteggio)
Album(id, titolo, anno, punteggio)
Artista(id, nome)

Si definiscono inoltre le due associazioni che servono a sostituire l'attributo artista nelle entità canzone e album definendo una relazione molti a molti:

ArtistaCanzone(idartista, idcanzone)
ArtistaAlbum(idartista, idalbum)

Aggiungo inoltre i vincoli di integrità referenziale:

- **ArtistaCanzone**.idcanzone fa riferimento a **Canzone**.id
- **ArtisaCanzone**.idartista fa riferimento a **Artista**.id
- **ArtistaAlbum**.idalbum fa riferimento a **Album**.id
- **ArtistaAlbum**.idartista fa riferimento a **Artista**.id

PROGETTAZIONE FISICA

In questa fase tradurrò lo schema logico in uno schema fisico, sulla base del DMBS che userò. Scelgo infatti di utilizzare PostgreSQL in quanto:

- Software libero e open-source
- Community, progetti e strumenti appositi anch'essi open-source
- Maggiori prestazioni e dimensioni ridotte del database rispetto alle alternative

Per l'assegnazione dei nomi alle tabelle e alle colonne mi sono attenuto alle seguenti convenzioni:

- Usare esclusivamente caratteri minuscoli
- Evitare simboli speciali e numeri
- Evitare spazi, dividendo le parole solo con trattino basso "_"
- Evitare le abbreviazioni

Inoltre, per l'id ho utilizzato un UUID[1], generato automaticamente da PostgreSQL.

Il risultato finale è il seguente schema fisico:

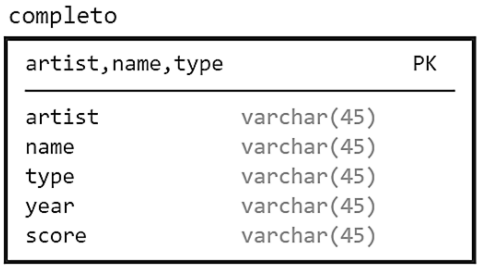
```
CREATE DATABASE whatsthehit

CREATE TABLE album
(
  id uuid NOT NULL,
  titolo text NOT NULL,
  anno numeric,
  punteggio numeric
);

ALTER TABLE album
    ADD CONSTRAINT chiave_primaria_id_album PRIMARY KEY (id);

CREATE TABLE artista
(
  id uuid NOT NULL,
  nome text NOT NULL
);

ALTER TABLE artista
```

[1] Acronimo per una sigla inglese che sta per "identificativo univoco universale". È una stringa di 32 caratteri esadecimali casuali, standardizzata nell'RFC 4122

```
    ADD CONSTRAINT "unicità_nome_artista" UNIQUE (nome),
    CONSTRAINT "chiave_primaria_id_artista" PRIMARY KEY (id);

CREATE TABLE canzone
(
  id uuid NOT NULL,
  titolo text,
  anno numeric,
  punteggio numeric
);

ALTER TABLE canzone
    ADD CONSTRAINT "chiave_primaria_id_canzone" PRIMARY KEY (id);

CREATE TABLE artisti_italia
(
  posizione integer NOT NULL,
  titolo text NOT NULL,
  artista text NOT NULL,
  anno integer NOT NULL
);

CREATE TABLE associazione_artista_album
(
  artista_id uuid NOT NULL,
  album_id uuid NOT NULL
);

ALTER TABLE associazione_artista_album
    ADD CONSTRAINT "chiave_primaria_associazione_artista_album"
PRIMARY KEY (artista_id, album_id),
    CONSTRAINT "chiave_esterna_associazione_artista_album_album_id"
FOREIGN KEY (album_id) REFERENCES album(id),
    CONSTRAINT "chiave_esterna_associazione_artista_album_artista_id"
FOREIGN KEY (artista_id) REFERENCES artista(id);

CREATE TABLE associazione_artista_canzone
(
  artista_id uuid NOT NULL,
  canzone_id uuid NOT NULL
);
```

```
ALTER TABLE associazione_artista_canzone
    ADD CONSTRAINT "chiave_primaria_associazioni_artista_canzone"
PRIMARY KEY (artista_id, canzone_id),
    CONSTRAINT "chiave_esterna_associazione_artista_canzone_artista_id"
FOREIGN KEY (artista_id) REFERENCES artista(id),
    CONSTRAINT "chiave_esterna_associazione_artista_canzone_canzone_id"
FOREIGN KEY (canzone_id) REFERENCES canzone(id);
```

Importazione dei dati



Le operazioni di adattamento dei dati iniziali alla nuova architettura si sono rivelate molto più lunghe e laboriose del previsto. Speravo infatti di riuscire a utilizzare qualche tool che facilitasse questa transizione, ma al contrario ho trovato solo soluzioni a pagamento o che non fornivano i risultati desiderati. A questo punto avevo più modi di agire:

- Programmaticamente, sviluppando un codice che leggeva la tabella originale e andava a riempire le tabelle risultanti
- Sviluppare un codice che sfruttava istruzioni SQL per creare le tre tabelle
- Manualmente, andando a adattare il vecchio database a quello nuovo tramite un'interfaccia grafica

Ho scelto l'ultima opzione perché mi sembrava la più semplice e logica; mi ha permesso di andare a curare la completezza e l'accuratezza del database, ma ha richiesto molto tempo nell'eseguire correttamente ogni passaggio.

Per questo procedimento mi sono avvalso dell'interfaccia grafica ufficiale di PostgreSQL: *pgAdmin*, con la quale sono andato passo-passo a modificare il database iniziale in formato a unica tabella nella struttura dati che ho precedentemente progettato.

CREAZIONE DEGLI ID

La tabella iniziale potrebbe essere considerata un join tra tutte le tabelle. Per normalizzarlo senza perdere l'informazione dell'associazione tra artista e album/canzone decido di assegnare due id a ogni riga utilizzando la funzione `uuid_generate_v1()`, integrata utilizzando la libreria standard di PostgreSQL `uuid-ossp`.

completo

artist,name,type		PK
id_artista	uuid	
id_prodotti	uuid	
artist	varchar(45)	
name	varchar(45)	
type	varchar(45)	
year	varchar(45)	
score	varchar(45)	

Queste colonne sarebbero diventate sia l'identificativo primario di ogni record di artisti e album, e di conseguenza sono di fatto, se messi in relazione, la nuova tabella associazione.

ESTRAPOLAZIONE DELLE TABELLE

A questo punto seleziono dalla tabella in maniera da dividere le entità artista da quella dei prodotti(album e canzone).

È stato molto importante fare attenzione a non perdere dati in questa fase. Sono stati richiesti più e più tentativi e test per arrivare a un risultato finale pulito e corretto da tutti i punti di vista. Sono stati riscontrati piccoli problemi che hanno rallentato lo svolgimento, ma riguardavano peculiarità del DBMS e dell'interfaccia, che altrimenti non avrebbero interferito.

Per ripulire infine la tabella artista dai valori ripetuti prima creo una tabella degli artisti con ID unici:

canzone

id_canzone	uuid	PK
id_artista	uuid	PK
titolo	varchar(45)	
anno	varchar(45)	
punteggio	varchar(45)	

album

id_album	uuid	PK
id_artista	uuid	PK
titolo	varchar(45)	
anno	integer	
punteggio	float	

artista

id	uuid	PK
nome	varchar(45)	

```
CREATE TABLE artisti_pulito_test AS
SELECT DISTINCT ON (nome) t1.id1, nome
FROM artisti_test AS t1
JOIN associazioni_artisti_test AS t2 ON t1.id1=t2.id1
ORDER By nome
```

Successivamente creo la tabella finale di associazioni con ID corretti sostituiti:

```
CREATE TABLE associazioni_artisti AS
SELECT art_p.id1 id1, aap.id2 id2
FROM ((associazioni_artisti aa join artisti art on aa.id1=art.id1)
JOIN artisti_pulito art_p on art.nome=art_p.nome)
JOIN associazioni_artisti aap on aa.id1=aap.id1
```

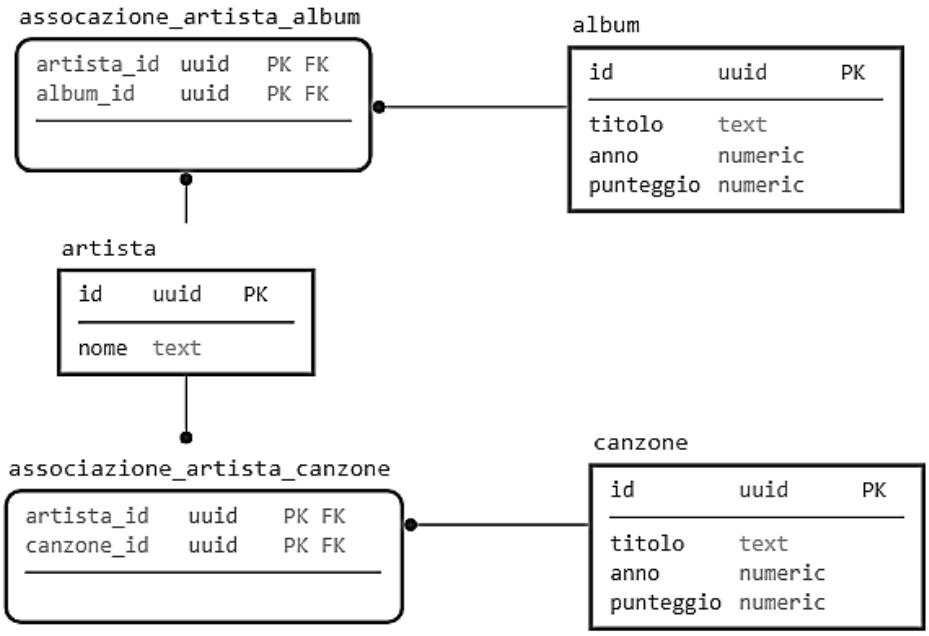
Che vado poi a dividere per album e per canzone semplicemente facendo una query di controllo sull'attributo `type`. Queste tabelle, una volta ripulite e definiti i vari vincoli, diventeranno le tabelle finali del database.

DEFINIZIONE DELLE ASSOCIAZIONI

Per creare le associazioni molti a molti seleziono nuovamente dalla tabella album e canzone le coppie di ID. Ogni coppia infatti è un occorrenza tra un artista e un prodotto. Vado quindi successivamente a rimuovere la colonna rispettiva all'ID dell'artista da album e canzone.

Per accertarsi che non esistano ID erronei creo inoltre vincoli di integrità referenziali dai valori della tabella associazioni ai rispettivi nelle tabelle esterne.

Dopo aver rinominato opportunamente il tutto, lo schema del database ha questa forma:



Aggiunta di record nazionali

La tabella finale si presenta completa dei dati già trovati inizialmente, che rappresentano però soltanto una media dei valori internazionali delle classifiche. In aggiunta a questi ho voluto mettere anche dei risultati che comprendessero soltanto

artisti_italia

posizione,titolo,artista,anno	PK
posizione	integer
titolo	text
artista	text
anno	integer

le classifiche italiane, sia a dimostrazione del funzionamento anche locale del meccanismo classifiche, sia ad ampliare il campo di ricerca.

Per questo mi sono appoggiato a `www.hitparadeitalia.it`, un sito che classifica le hit italiane sulla base di vari parametri di ascolto e classifiche. Il sito è organizzato in varie pagine HTML che listano le canzoni. In questo caso non si presenta un valore punteggio ma un numero posizione.

Il sito purtroppo non fornisce i suoi dati sottoforma di CSV, per ricavarli ho progettato uno script di scraping Node.JS. Ho utilizzato una libera chiamata `cheerio`, che mi consente di accedere in maniera più veloce alle marche dei vari file HTML che ricavo tramite richieste HTTP cicliche. Dopo varie pulizie e espressioni regolari, vado a formattare un file CSV `output.csv`. Vado a caricare questo file in una tabella apposita chiamata `artisti_italia`.

CAPITOLO 2

PROGETTAZIONE DEL SERVER

A questo punto i dati che il sito deve fornire sono impostati nella maniera desiderata, ovvero sottoforma di database relazione PostgreSQL. In questa fase **creerò il sistema server che fornisce sia questi dati sia l'eventuale sito statico che farà da interfaccia web**.
A questo scopo utilizzo Node.JS, l'implementazione server di Javascript, e per il routing farò ampio uso di una sua libreria: Express.

Scelta dell'ambiente di sviluppo

Node.JS è un ambiente server open-source che usa il linguaggio Javascript. Inoltre, sfrutta la programmazione asincrona per assegnare a ogni procedura un singolo processo, così da essere estremamente efficiente per la memoria e reattivo con i tempi di risposta. Occorrerà infatti che il server esegua velocemente query SQL, molteplici chiamate a REST API esterni e dovrà inoltre generare contenuto dinamico nelle pagine, oltre a offrire la parte statica del sito, perché l'ottica del progetto punta sul creare un prodotto di largo utilizzo.
Come editor principale utilizzo Visual Studio Code, un editor stabile targato Microsoft in continuo aggiornamento che, tra le sue tante funzionalità, permette di installare plugin per formattare automaticamente il codice e compilatori che restituiscono errori visuali.

Impostazione del progetto

L'intera struttura del codice è stata elaborata in maniera da favorire l'ampliamento e il mantenimento del codice. Varie accortezze sono state applicate al codice:

- Utilizzo di GIT per il *versioning* e gestione delle modifiche
- Pieno sfruttamento delle capacità di programmazione asincrona che Node.JS offre
- Gestione degli errori
- Regole di *naming convention* unificata nella definizione di variabili e funzioni
- Utilizzo di variabili d'ambiente per alterare il funzionamento e renderlo più adattabile
- Struttura delle cartelle divisa logicamente per componenti

Tecnologie principali utilizzate

Qui elenco le principali librerie e tecnologie che ho utilizzato per programmare il server di WhatsTheHit.

EXPRESS

Questa libreria permette, in maniera molto semplice e minimale, di aggiungere funzionalità altrimenti molto complesse da creare in normale Javascript. Alcune delle più utili sono state:

- Sintassi di routing minimalista
- Importazione dei middleware semplicissima
- Supporto per le sessioni
- Parsing del body delle richieste in entrata
- Templating di parametri
- Elegante gestione degli errori
- Costruito per utilizzare la meglio le capacità asincrone di Node.JS

PROMISE

La promise è un concetto piuttosto avanzato in programmazione, implementato nativamente in Node.JS. Un oggetto di tipo `Promise` in Javascript è un oggetto che rappresenta il risultato eventuale di una funzione.

In questa maniera, la struttura delle chiamate a funzioni asincrone diventa molto più elegante ed efficiente perché, invece che incastrare call-back una dietro l'altra, è possibile restituire un oggetto di tipo Promise il cui valore verrà considerato nel momento in cui la relativa funzione asincrona avrà ritornato qualche risultato.

YARN

Yarn è un gestore pacchetti alternativo sviluppato da Facebook. Ho scelto questo gestore perché, almeno all'inizio del progetto, questo forniva più sicurezza e controlli delle dipendenze ed era anche più veloce nell'utilizzo rispetto al suo concorrente ufficiale NPM.

Questo tipo di gestori in generale fornisce un sacco di vantaggi tra cui:

- Gestione automatizzata delle dipendenze, che facilita lo scambio e il salvataggio sul cloud
- Creazione di alias dei comandi da lanciare in console

KNEX.JS

Knex è una libreria open-source che agevola e velocizza la comunicazione col database. È stato essenziale il suo utilizzo per più ragioni:

- Composizione dinamica della query
- Controllo e sanitizzazione degli input
- Controllo della sessione tra server e database
- Conversione adeguata nei tipi richiesti

Interrogazione del server

Il programma non gestisce direttamente stringe di SQL, ma crea una interfaccia intermedia nella quale sia possibile, tramite un oggetto JSON adeguatamente formattato, riuscire a richiedere semplicemente le informazioni dal database.

```
{
  "from": "canzone",
  "select": "count",
  "where": {
    "anno": ["1980", "1992"]
  }
}
```

Il JSON viene inserito nel body, una sezione della richiesta HTTP al server.

La sintassi da utilizzare richiama quella già molto semplice dell'SQL, così da garantire più libertà da parte dell'utilizzatore di comporre le query nella maniera che più preferisce. In questa maniera infatti, non si limita l'accesso ai dati tramite funzioni a uso singolo più minime ma anzi si dà più libertà a chi vorrà interrogare il database di comporre e richiedere esclusivamente le informazioni di cui ha bisogno. Avere questo tipo di scelta, in questo caso, è molto più efficace che creare funzioni più semplici ma che non forniscono questo ampio raggio di utilizzo, perché si crea molta meno dipendenza tra ciò che il lato server propone rispetto a quello di cui il client ha bisogno. La facciata web sarà quindi più libera di modificare o aggiungere componenti o funzioni diversi senza necessariamente ogni volta andare a modificare il programma lato server.

La scelta di una API così aperta è dettata dalla caratteristica che volevo dare al progetto di indipendenza totale tra front-end e back-end. Sono al corrente che nel caso in cui vengano eseguite una serie di query onerose il funzionamento del server potrebbe risentirne. Faccio affidamento sul tipo di implementazione che il front-end ne farà, e al fatto che imposti delle contromisure per evitare che questo tipo di problemi accada.

Ricerca di lingua, genere e foto dell'artista

Nel server sono anche incluse tre funzioni di *enrichment* per trovare lingua, genere e immagine di un artista o complesso.

Le informazioni vengono indicate tramite una semplice stringa contenente il nome passato come parametro alla richiesta HTTP ai vari endpoint, rispettivamente `/lang`, `/genre` , `/img`. Il risultato sarà stringa che contiene la lingua o il genere (se disponibile in italiano altrimenti in inglese) oppure un URI che fa riferimento all'immagine.

Il funzionamento si basa sull'utilizzo dell'API di Wikidata, una versione di Wikipedia ma sottoforma di biblioteca dati, invece che di testi enciclopedici. Grazie la libreria Node.JS ufficiale, il server fa più chiamate per prima individuare l'identificativo dell'entità, per poi arrivare a selezionare il risultato tramite un codice ben preciso che rappresenta la caratteristica ricercata. Infine, il server prende il risultato, lo ripulisce, e lo restituisce sottoforma di stringa.

Sicurezza

Vorrei inoltre soffermarmi sui vari livelli di sicurezza che ho inserito. Nonostante la dimensione dell'applicazione fosse piuttosto ridotta e i dati che effettivamente poi andavo a inserire nel database erano molto pochi, ho voluto garantire che il funzionamento dell'applicazione e la sicurezza del database non avessero falle a livello software. I vari stratagemmi che ho utilizzato sono stati:

- Evitare di inserire credenziali in chiaro sul codice, ma aggiungendole esclusivamente tramite variabili di sistema
- HTTPS che garantisce la crittografia dei dati mandati e ricevuti
- CSRF che garantisce che solo il client ufficiale possa comunicare con il server
- Limite del numero di chiamate al server per minuto
- Liberia Helmet
- Disabilitato `x-powered-by`

Tutto questo garantisce sicurezza all'applicazione pure permettendo di avere un codice open-source, in quanto i vari certificati e controlli sono o impostati al di fuori del codice o utilizzano variabili impostate casualmente.

Distribuzione

Per il deployment effettivo del sito mi sono avvalso di Heroku, una soluzione PaaS che ha un piano gratuito per piccole applicazioni. È stato anche possibile automatizzare il procedimento di distribuzione in quanto il sito ha una funzione che permette di collegarsi direttamente alla repository su GitHub.

Per far sì che Heroku funzioni correttamente è stato necessario inserire un file chiamato `Procfile` nella cartella radice del progetto in cui indicavo la funzione da lanciare all'avvio.
Per permettere inoltre al server hostato sul cloud di accedere al database ho sfruttato un servizio di hosting PostgreSQL gratuito chiamato ElephantSQL.
In questo modo è possibile accedere liberamente all'ultima versione dell'API in qualsiasi momento all'indirizzo `https://whatsthehit.herokuapp.com/`.

Esempio d'utilizzo online

Per eseguire una semplice query SELECT i requisiti sono:

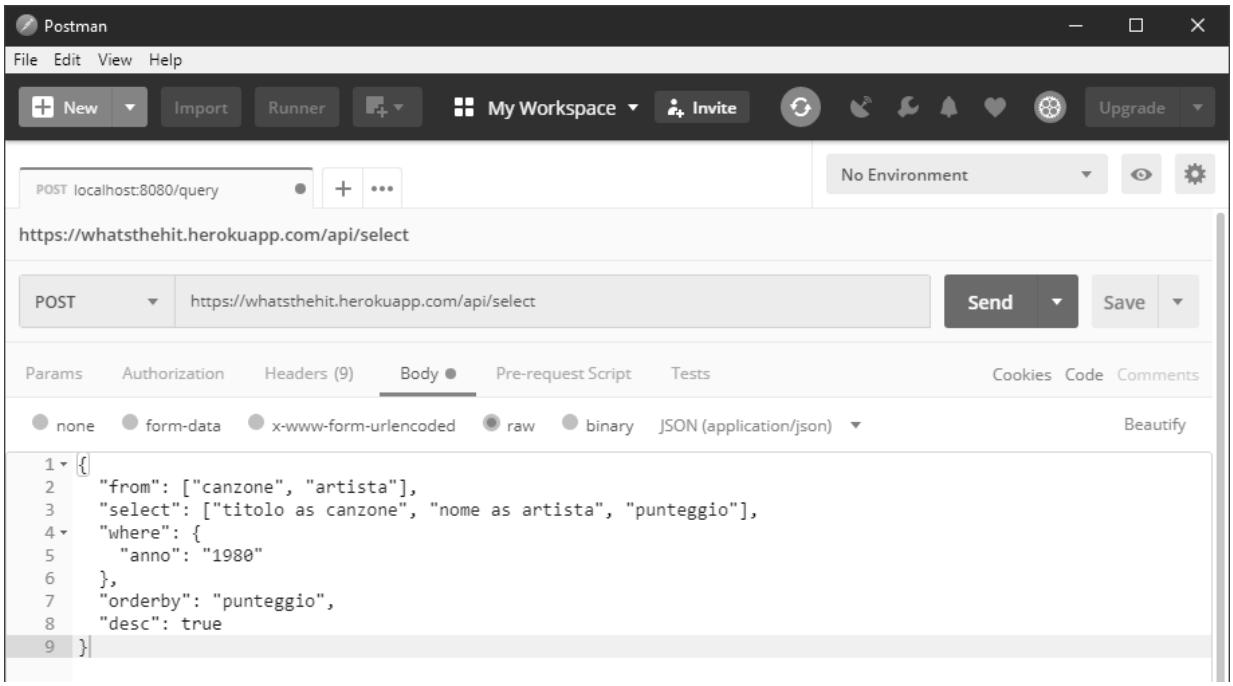
- Fare richiesta di tipo POST a indirizzo `https://whatsthehit.herokuapp.com/api/select`
- Impostare l'header `Content-Type` al valore `application/json`
- Aggiungere al body una stringa sottoforma di oggetto JSON formattata in maniera come illustrata nella documentazione online.

La più classica query da eseguire è quella che richiede di trovare la classifica annuale delle hit per un anno. Nel caso si vogliano cercare le canzoni ordinate per punteggio nel 1980 occorrerà compilare il body in questo modo:

```
{
  "from": ["canzone", "artista"],
  "select": ["titolo as canzone", "nome as artista", "punteggio"],
  "where": {
    "anno": 1980
  },
  "orderby": "punteggio",
  "desc": true,
  "limit": 30
}
```

È possibile interrogare il server WhatsTheHit usando Postman, l'interfaccia grafica che ho utilizzato durante lo sviluppo. Molto intuitiva, permette di comporre le richieste e aggiungere e rimuovere parametri in maniera molto veloce.

Impostandola in questa maniera e cliccando sul pulsante `Send`:



Avremmo di risposta un array di oggetti JSON ordinati per punteggio.

I risultati completi hanno questa forma:


```
[
    {
        "canzone": "Another Brick in the Wall (part 2)",
        "artista": "Pink Floyd",
        "punteggio": "32.020"
    },
    {
        "canzone": "A Woman in Love",
        "artista": "Barbra Streisand",
        "punteggio": "25.365"
    },
    {
        "canzone": "(Just Like) Starting Over",
        "artista": "John Lennon",
        "punteggio": "22.448"
    },
[…]
]
```


Ogni oggetto JSON rappresenta un record, selezionato secondo quanto indicato nella query, le cui informazioni sono stringhe.

CAPITOLO 3

DOCUMENTAZIONE DELLA WEB-API

Il server, più precisamente la parte della web-api, fornisce i dati del database sulla base di una particolare sezione della richiesta HTTP, detta *body*. Questo campo deve essere correttamente compilato per far sì che il database abbia sufficienti informazioni per compiere una query.

Nasce quindi la necessità di stilare una documentazione esaustiva che spieghi come vada formattato questo campo e come forgiare la richiesta. La finalità principale sarà quella di **agevolare la comprensione e l'utilizzo dell'API e di tutte le sue funzionalità, così da rendere l'accesso ai dati indipendente dalle varie astrazioni applicative**.

Profili utente

Gli utilizzatori della web-api sono soprattutto i programmatori del *front-end*, che avranno bisogno di rivolgersi continuamente alla documentazione. In essa infatti dovranno trovare informazioni tecniche non altrimenti fruibili sul funzionamento dell'interfaccia API, che di base è l'unico punto di accesso diretto al database.

Il funzionamento dell'interfaccia non è banale e non segue nessuno standard in particolare. L'unica fonte affidabile di informazioni su come utilizzare il servizio di interrogazione dati è quindi la documentazione.

Non si esclude però una piccola parte di utenza che, interessata ad approfondire o a racimolare velocemente più dati sottoforma di file, preferisca usare direttamente l'API invece che ricorrere al sito. Se infatti si disattiva il controllo CSRF è infatti possibile fare chiamate all'API liberamente, senza l'obbligo di passare per il sito web.

Competenze

Per la stesura della documentazione ho cercato (per quanto possibile) di non utilizzare linguaggio tecnico: è importante dare spazio agli argomenti trattati in modo ampio e chiaro a chiunque voglia cimentarsi nella materia, anche ai meno esperti, così da dar loro un'occasione per imparare. Il lavoro però non è di matrice didattica e non si rivolge a un pubblico ampio e generico, ma il target è una

ristretta cerchia per la quale ho individuato le seguenti competenze che divido in necessarie e facoltative:

COMPETENZE NECESSARIE

- Conoscenze di livello OSI applicativo e richieste HTTP e AJAX
- Competenze all'utilizzo di client HTTP o CLI per forgiare richieste HTTP

COMPETENZE FACOLTATIVE

- Conoscenza di sintassi Javascript e formattazione oggetti JSON
- Conoscenza minima del linguaggio SQL

Obiettivi

Lo scopo finale è quello di avere un prodotto web, liberamente fruibile e dettagliato, accostato al server che è già open-source. In questo modo:

- Fornisco un punto in cui ritrovare tutte le informazioni relative al progetto
- Creo una maniera facile di andare a leggere e navigare la documentazione dell'API
- Fornisco esempi pratici di utilizzo, oltre a una sezione di FAQ

Tutto questo è possibile grazie a una funzionalità di GitHub (il sito che ho scelto per ospitare il codice di questo progetto) che permette di aggiungere una wiki formattata in Markdown[2].

Contenuti

La stesura di una documentazione permette di:

1. Elencare le operazioni a disposizione e descriverne parametri richiesti e funzionamento, così da renderle facilmente utilizzabili e richiamabili dai programmatori dell'interfaccia.
2. Descrivere passo-passo come installare il server sulla propria macchina, in maniera da permettere agli sviluppatori di lavorare in locale.
3. Creare una sezione FAQ apposita

Il fatto di avere un codice open-source permette a chiunque di indicare i punti critici o gli errori più comuni, così da permettere l'ampliamento della documentazione sulla base dei problemi riscontrati.

[2] Linguaggio di annotazione testuale semplice e di veloce modifica. Sfrutta file di testo formattati usando caratteri speciali

Inoltre, rielaborare logicamente il tutto, mettendolo sottoforma di una documentazione, mi permette di razionalizzare meglio concetti e procedure: mettendomi nei panni di chi andrà a utilizzare il mio programma, ho maniera di capire meglio quali sono gli aspetti da migliorare, aggiungere o rimuovere.

Caratteristiche

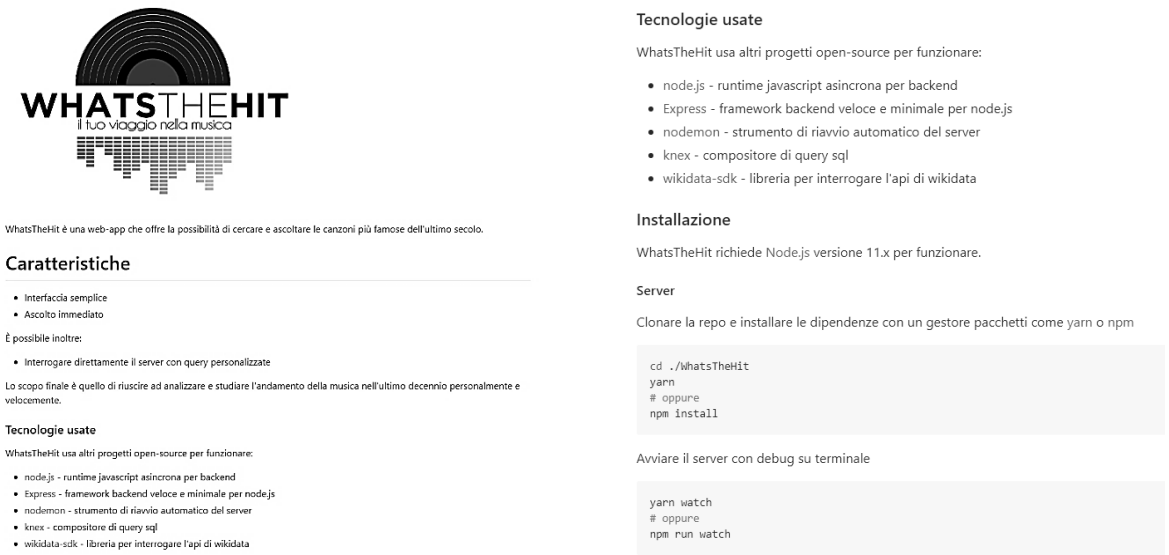
Dopo aver definito i profili e l'obiettivo, occorre chiarire quali saranno le caratteristiche che questo tipo di prodotto deve avere per funzionare. In particolare, definisco come fondamentali:

- Semplicità: una forma naturale e semplice è più che sufficiente per un prodotto di questo tipo, non occorre nessuno studio particolare dell'impaginazione se non quello che basta a rendere la documentazione fruibile e navigabile in maniera veloce
- Sinteticità: il non dilungarsi è una caratteristica fondamentale per una documentazione; chi consulta la documentazione cerca, spesso e volentieri, una soluzione rapida e veloce. Questo non significa fare economia di spiegazioni, ma è più una garanzia dell'essenzialità delle informazioni riportate
- Usabilità: il prodotto è conforme a tutti gli standard del Web2.0

Risultato

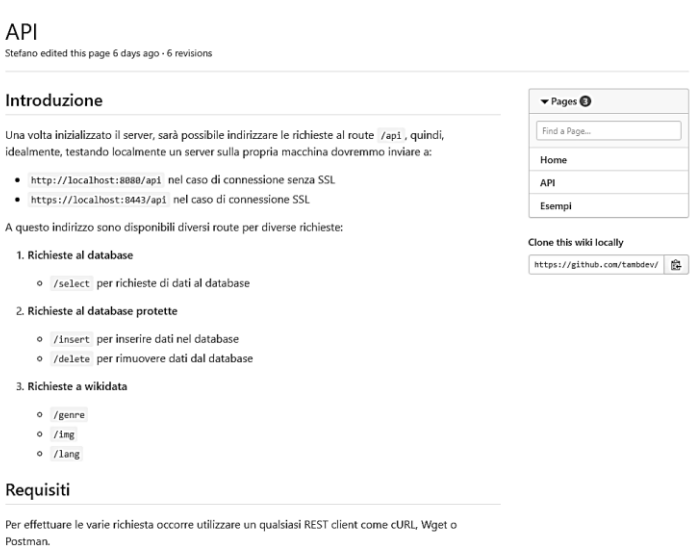
Il prodotto finale si presenta come da immagine: più pagine web responsive e leggibili, con collegamenti e controlli di navigazione ai vari capitoli e sezioni. Il tutto è liberamente visitabile alla pagina:

https://github.com/tambdev/WhatsTheHit/wiki



Esempio di consultazione

Consideriamo il caso di un qualsiasi utente che voglia collaborare o sviluppare un prodotto associato a questo progetto. La prima pagina che vede è la home della repository di GitHub dove, oltre a collegamenti al codice sorgente, potrà trovare sin da subito informazioni su come installare il prodotto in locale sulla sua macchina.



Sin da subito è chiaro quali sono le dipendenze requisite, librerie utilizzate e come muoversi all'interno della cartella.

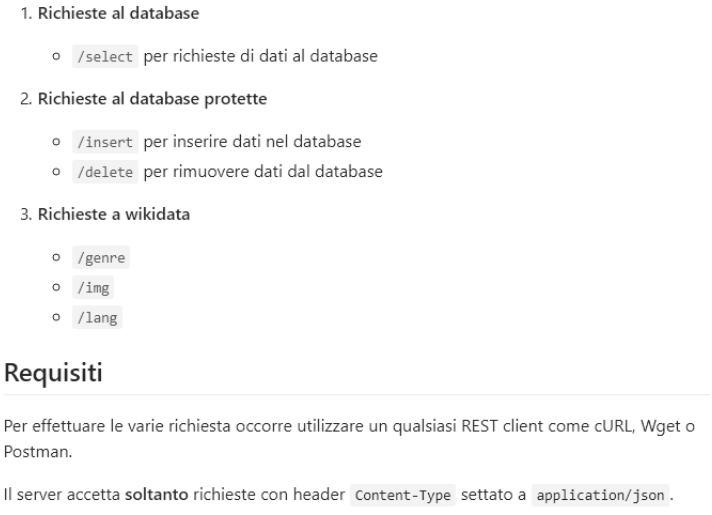
Una volta eseguite queste operazioni preliminari il tutto è pronto per essere eseguito in locale sulla propria macchina.

Per avere informazioni invece riguardo al funzionamento dell'API, l'utente può visitare la wiki apposita tramite un link posizionato in cima alla pagina oppure alla fine del procedimento di installazione. Se ci fosse bisogno di un esempio pratico e veloce l'utente può subito ritrovarsi nella sezione "Esempi", altrimenti per andare più nel dettaglio può visitare la sezione "API".

Se l'utente necessita di entrare più nel dettaglio delle funzione proposte, tramite link posizionati all'inizio della pagina sarà possibile raggiungere la parte che più desidera.

Ogni sezione relativa a ogni comando è affiancato da spiegazione dettagliata e un esempio di query pratico per spiegare il funzionamento e la formattazione.

Nel caso invece si voglia subito provare a comporre le richieste che, a mio avviso, sono le più utili o comunque le più utilizzate, sarà possibile copiare un pezzo di codice dalla sezione Esempi.



CAPITOLO FINALE

CONCLUSIONE

Spinto dalla mia voglia di imparare queste nuove tecnologie e dall'entusiasmo di poter finalmente scoprire cosa comporta far nascere un'idea di questo tipo, **ho creato la struttura base e il lato server di WhatsTheHit, una applicazione web il cui fine è quello di fare da canale di informazioni testuali e multimediali.**

Osservazioni

Con questo progetto ho cercato di compiere uno studio personale dello sviluppo web moderno applicato a un idea innovativa: organizzare in forma semplice e accessibile informazioni multimediali. A tal fine, ho condotto una ricerca dello stato dell'arte dei più importanti e usati strumenti del web-development. Ho individuato quali erano pro e contro di ogni linguaggio e libreria e ho cercato di addentrarmi in questa tecnologia il più a fondo possibile. Ho infine spiegato l'utilizzo del prodotto che avevo creato tramite una semplice documentazione che illustra e mostra tramite esempi come approcciarsi all'API di interrogazione del database.

Uno degli aspetti più interessanti è stato come, in futuro, un sito di questo tipo fosse effettivamente di interesse al pubblico. Spingendo su questa caratteristica ho notato più eccitazione e coinvolgimento da parte mia e dei miei collaboratori. C'era un grande interesse da parte nostra, giovani laureandi, nel dar vita a qualcosa di tangibile, da affiancare alle realtà informatiche che ci siamo abituati ad utilizzare ogni giorno ma delle quali, purtroppo, non ci rendiamo conto del lavoro immenso che c'è dietro.

Dal mio punto di vista, ho potuto finalmente toccare con mano un argomento che invidiavo ai professionali: lo sviluppo Node.JS avanzato, un linguaggio in continuo sviluppo, data l'enorme e attiva community. Questa esperienza mi ha inoltre permesso di sfruttare PostgreSQL, un nuovo DBMS, anch'esso con una comunità molto attiva e in veloce sviluppo, e processori HTML lato server come Pug. L'aspetto più soddisfacente è stato quello di creare, a partire da questi vari componenti, qualcosa di nuovo e originale utilizzando un ambiente di sviluppo personalizzato sulle mie esigenze e liberamente riproducibile, dato che tutte le dipendenze sono open-source.

Aspetti critici

Molte parti del mio lavoro si sono rivelate particolarmente difficili da gestire da solo. L'importazione dei dati nel database ha richiesto un sacco di tentativi e riprove per arrivare al risultato finale. Inizialmente pensavo di poterlo fare programmaticamente, ma non avendo a disposizione un'interfaccia di interrogazione del server a questo punto dello sviluppo, pareva più semplice andare manualmente a editare passo-passo il database piuttosto che creare e testare a tentativi uno script che maneggiasse questa mole di dati.

È probabilmente stata la fase in cui ho perso più tempo, perché per ogni singola operazione dovevo controllare di non aver perso dati o fatto errori di alcun tipo, ma in compenso questo processo mi ha dato la certezza che i dati importati fossero completamente curati e controllati.

I testing preliminari, con i quali volevo controllare quale piano effettivamente era il più efficiente da svolgere e quali possibilità avevo o non con il materiale in possesso, hanno anch'essi richiesto una grande mole di tempo, e non si sono mai rivelati conclusivi in tutto e per tutto. Col senno di poi, avrei dedicato più attenzione a questa parte, successiva al planning.

Possibili ampliamenti

Gli aspetti che ho affrontato però sono solo una piccola parte di tutta la conoscenza in materia. Possibili sviluppi futuri al progetto potrebbero riguardare:

- Il database che, per quanto ben gestito e pulito, presenta il problema della mancanza di dati o quantomeno della relazione tra dati. Un possibile ampliamento sarebbe un lavoro di scraping dei dati musicali da altri siti per cercare di crearsi una propria classifica personale, così da essere liberi di aggiornala e ampliarla nel tempo.
- Un altro possibile tipo di progetto che ci interessava era proprio l'analisi fisica della musica in quanto tale: un'analisi dei cambiamenti sonori nel tempo. Questa idea però si imbatte purtroppo nelle normative copyright e nei termini di utilizzo delle maggiori piattaforme di streaming. Entrambi questi aspetti meriterebbero ampliamenti.

Per qualsiasi contributo, il codice è di libero accesso all'indirizzo:

https://github.com/tambdev/WhatsTheHit

BIBLIOGRAFIA

- Atzeni, Paolo, Stefano Ceri, Piero Fraternali, Stefano Paraboschi, Riccardo Torlone. *Basi di dati. Modelli e linguaggi di interrogazione*. Milano, McGraw-Hill Education, 2013.