MusicDiscover

**Facoltà di Ingegneria dell’informazione, informatica e statistica**

**Dipartimento di Ingegneria informatica, automatica e gestionale  
Corso di laurea in Ingegneria Informatica e Automatica**

**Stefano Rughetti**

**Matricola 1757975**

Relatore

Andrea Vitaletti

A.A. 2019-2020

Indice

1. Introduzione 1
2. Scelta delle tecnologie per lo sviluppo 2
   1. Principali stack contemporanee 3
3. Scelta delle qualità tecnologiche 4
   1. React 5
   2. Nodejs 7
   3. Express 8
   4. MongoDB 8
4. Strumenti, librerie e tecnologie accessorie 9
   1. Github 9
   2. Json 9
   3. React-Bootstrap 10
   4. REST 11
   5. Nodemon 13
5. MusicDiscover 13
   1. Architettura dell’applicazione web 14
   2. API server layer 15
   3. Front-end server layer 16
      1. Pagina Homepage 16
      2. Pagina Bandcamp 18
      3. Pagina Lastfm 23
      4. Pagina Twitter 27
6. Bibliografia 30

1. Introduzione

La musica è uno dei media più usufruiti nella vita di tutti i giorni: la sentiamo in macchina per andare al lavoro, la sentiamo durante i momenti di relax, nei momenti di gioia e convivialità.

Molte volte però non siamo noi a scegliere ciò che sentiamo. Spesso canzoni ci vengono automaticamente proposte tramite playlist create da algoritmi inseriti in piattaforme come Youtube e Spotify.

Questi algoritmi scelgono soprattutto canzoni che sono di tendenza al momento, inoltre sono influenzati da campagne pubblicitarie.

Ciò che ho deciso di sviluppare è un'applicazione web che, in maniera veloce e intuitiva, potesse dare informazioni sugli artisti musicali, in modo da trovare personalmente nuovi artisti, rendendo così l’utente più partecipe nello sviluppo dei propri orizzonti musicali, perciò l’applicazione web è stata chiamata “MusicDiscover”.

La possibilità di svilupparla è nata grazie alla parte progettuale dell’esame “**Reti di Calcolatori**”, tenuto dai docenti Stefano Leonardi e Andrea Vitaletti e al corso di “**Linguaggi e Tecnologie per il Web**” tenuto dal docente Riccardo Rosati.

Durante questi corsi, ciò che aveva catturato il mio interesse era la grande quantità e complessità di servizi che possono essere distribuiti tramite le pagine web dinamiche.

Il funzionamento dell’applicazione web è intuitivo in quanto si basa su tre servizi messi a disposizione su tre diverse sezioni della pagina web.

Ogni sezione ha un campo d’input in cui inserire il nome di un artista. Una volta inserito il nome e avviata la ricerca, a seconda della sezione scelta, l’utente potrà:

* Sentire le varie canzoni dell’artista presenti nei diversi album;
* Ottenere le informazioni dell’artista come la biografia, le canzoni più popolari e gli artisti simili;
* I tweets più recenti dell’artista presi direttamente dal loro account Twitter.

Si assume che l’utente abbia almeno un artista musicale preferito tramite il quale si possa iniziare la ricerca.

Quest'applicazione web è pensata per un utente che vuole visualizzare in maniera semplice le informazioni, senza dover creare account in molteplici siti e sentire immediatamente la musica prodotta dall’artista.

L’ottenimento di queste informazioni è stato possibile grazie alle API messe a disposizione dai siti Twitter e Lastfm, la possibilità di sentire le canzoni è invece resa disponibile grazie alla piattaforma Bandcamp.

Il funzionamento dell'applicazione web dipende dalla connessione con queste tre piattaforme: Twitter, Lastfm e Bandcamp. Se una piattaforma dovesse andare offline i servizi ad esso associati non sarebbero più funzionanti.

Tuttavia, poiché questi servizi sono indipendenti fra di loro, se una delle tre piattaforme fosse in manutenzione, o per qualche motivo non fossero disponibili le informazioni, l’applicazione potrebbe comunque offrire i servizi connessi alle piattaforme rimanenti.

2. Scelta delle tecnologie per lo sviluppo

Nello sviluppo di un’applicazione web è fondamentale la scelta delle tecnologie da utilizzare, perché vanno a cambiare in maniera significativa in funzione del tipo di contenuto.

Una web application che ha una quantità bassa di contenuto statico è adatta ad essere sviluppata con un'architettura single-page application (SPA). Una SPA genera tutto il suo contenuto all'interno del browser e non ha mai bisogno di ricaricare completamente la pagina per cambiare il suo contenuto. Questo approccio riduce notevolmente la quantità di risorse necessarie dedicate al server perché la maggior parte dei processi vengono eseguiti nel client. Ma il client per eseguire questi processi ha bisogno del framework utilizzato per sviluppare una SPA (ad esempio Angular, Vue o React) e del codice fornito dal server e quindi il primo caricamento della pagina risulterà lento, ma le successive azioni risulteranno molto più veloci perché si richiede molto meno codice al server e si dovrà aggiornare solamente il contenuto di alcune porzioni della pagina invece che la pagina intera.

Invece una web application che ha un contenuto statico molto elevato è adatta per essere sviluppata in modo classico, cioè la pagina già renderizzata tramite un HTML template engine è inviata dal server e il client non ha bisogno di eseguire nessuna azione e non ha bisogno di scaricare un framework per visualizzare la pagina. Ciò riduce di molto la latenza nella visualizzazione ma riduce la fluidità successiva dovendo, ad ogni cambio di contenuto, ricaricare la pagina.

Deciso il tipo di architettura (oppure un misto delle architetture) si deve scegliere le tecnologie che andranno a costruire l'architettura. La combinazione delle tecnologie usate per costruire una web application si chiama “stack”, termine andatosi a popolarizzare tramite la LAMP stack, che è un acronimo per Linux, Apache, MySQL and PHP.

2.1 Principali stack contemporanee

La principale stack adottata per creare nuove web application in questo momento è la stack MEAN acronimo di MongoDB, Express, Angular e Nodejs, che quindi utilizza il seguente schema:

* **Front-end Framework:** Angular.js
* **Back-end Technologies:** Nodejs
* **Back-end Framework**: Express
* **Database Technologies:** MongoDB

**Questa stack ha molti vantaggi tra cui, essere open source e quindi avere una grande community alle spalle che ha documentato estensivamente le tecnologie componenti, avere una grande varietà di tool di sviluppo e di monitoring e l’utilità nello sviluppo che i framework portano rendendo la programmazione semplice e organizzata.**

**Un'altra stack che sta crescendo in popolarità è la stack MERN che ha le stesse componenti della stack MEAN tranne per il front-end framework che ha React al posto di Angular. Inoltre, React non è un vero e proprio framework come Angular ma solamente una libreria quindi limitando le funzionalità disponibili e dovendo adottare servizi di terze parti.**

**L’ultima stack che reputo di grande interesse è la stack MEVN che adotta Vue.js come framework front-end. Vue.js ha il vantaggio rispetto ad Angular e a React di essere light-weight in quanto offre solamente alcune funzionalità principali ma quest’ultime possono essere estese tramite terzi come React, di fatto prendendo solo lo stretto necessario allo sviluppo.**

**Queste stack hanno un grande vantaggio, Javascript è usato in ogni sua componente e quindi facilita di molto lo sviluppo e l’organizzazione, in più rende ogni programmatore capace di creare servizi sia per il back-end che per il front-end.**

**3. Scelta delle qualità tecnologiche**

**Per sviluppare il mio progetto ho deciso di utilizzare la stack MERN, i motivi sono i seguenti:**

**Il progetto in questione è stato sviluppato unicamente da una sola persona quindi React con Nodejs dà la possibilità di utilizzare Javascript in tutti i vari layer del progetto facilitando lo sviluppo dell’applicazione e dando la possibilità di non dover cambiare paradigma e strutture dati quando si lavora con il back-end e il front-end.**

**Inoltre, in un ambiente lavorativo con più persone, questo fatto rende possibile organizzare dinamicamente i programmatori nella fase di sviluppo di un progetto, per esempio se una parte del progetto dovesse essere completata gli stessi programmatori che lavoravano su quella parte potrebbero essere messi nello sviluppo di altre parti senza dover cambiare linguaggio di programmazione.**

**Un altro motivo che ha portato alla scelta della stack MERN per lo sviluppo dell’applicazione è che React, non essendo un vero e proprio framework ma solamente una libreria, è libero dai costrutti imposti da un MVC (in contrasto per esempio ad Angular).**

**Ciò comporta la possibilità di utilizzare React solamente quando è strettamente necessario e facilità l’organizzazione della struttura e dei files e alleggerisce la struttura del progetto. Un'altra tecnologia che trova nella leggerezza uno dei suoi punti di forza è Nodejs che ha di per sé solamente un gruppo limitato di funzioni ma ben strutturate e per ampliare queste funzioni si possono installare dei moduli sviluppati da terzi, come ad esempio express che è la base per la gestione del server. Ciò permette di installare solamente i moduli necessarie snellendo la mole dei file e la pesantezza del progetto.**

**In più la quantità dei moduli resi disponibili tramite npm è molto ampia grazie alla grande community che supporta Nodejs.**

**Inoltre, l’applicazione che ho sviluppato, essendo una SPA, deve essere veloce e responsiva agli input dell’utente e al cambiamento dei dati visualizzati nella pagina, in modo di rendere piacevole all’utente la navigazione. Ciò è stato ottenuto grazie all’ottimizzazione da parte di React riguardo alla cattura degli eventi riguardanti i componenti presenti nella pagina e riguardo alla gestione dei cambiamenti nei dati appartenenti ai componenti, tutto ciò è gestito autonomamente senza il bisogno di intervento da parte del programmatore.**

**Successivamente il linguaggio di mark-up JSX, utilizzato da React, ha subito catturato la mia attenzione per la sua facilità di utilizzo e la capacità di automatizzare la creazione di elementi DOM tramite l’utilizzo di costrutti interni a Javascript.**

**3.1 React**

**La tecnologia che differenzia questa stack dalle altre è la libreria React utilizzata per sviluppare il front-end. React è stata creata dal dipartimento pubblicitario di Facebook per ovviare al problema della complessità di un framework che utilizza un MVC (Model-view-controller) pattern, quindi rendendo la vista delle pagine dichiarativa invece che imperativa.**

**Avere una vista dichiarativa significa che il programmatore non deve gestire gli effetti che un cambiamento ha sul layout della pagina o sui dati. Quindi avere un front-end dichiarativo rende il comportamento della pagina consistente, prevedibile e facile da gestire e da capire. Per rendere veloce il rendering della pagina, a fronte delle varie manipolazioni che React compie, viene utilizzata la Virtual DOM technology che crea una struttura virtuale in memoria del layout della pagina. Quando c’è un cambiamento nello stato della pagina la struttura nuova della pagina viene comparata con quella vecchia, quindi React effettua i cambiamenti solo delle variazioni nella pagina.**

**Il principale costrutto di React sono i “componenti” che mantengono i loro propri stati e si auto-renderizzano nel layout della pagina. Ogni React tutto è strutturato tramite componenti e la capacità di dividere tutto in componenti rende la scrittura di codice e la progettazione della pagina più facile rispetto ad avere un blocco monolitico.**

**Questi componenti possono comunicare tra di loro attraverso la condivisione di proprietà di sola lettura e tramite funzioni call-back.**

**La maggior parte dei framework per le applicazioni si basano su templates per automatizzare il compito di creare la parte HTML della pagina o gli elementi del DOM. Il linguaggio per creare e utilizzare questi template è specifico dei framework e gli sviluppatori hanno l’obbligo di impararlo, ma questo non succede in React. React usa Javascript per creare HTML ed elementi DOM. Per esempio, si può utilizzare un ciclo for(…) oppure usare la funzione map() su un array per costruire una tabella.**

**Questo grazie ad un linguaggio intermedio che rappresenta il Virtual DOM, JSX, che è molto simile al linguaggio HTML. Da notare che JSX non è un linguaggio di programmazione; è un linguaggio di markup come HTML.**

**L’ultima particolarità di questa libreria è la sua isomorfia, cioè la capacità di eseguire la stessa parte di codice sia nel front-end che nel back-end. Questo fa sì che il server può creare una pagina se richiesto, questo fenomeno si chiama server-side rendering.**

**3.2 Node.JS**

**Node.js è stato creato prendendo l’engine V8 di Chrome che compila Javascript e facendolo eseguire indipendentemente, quindi di fatto è Javascript ma compilato al di fuori del browser di ricerca. Le più notabili compagnie che utilizzano Node.js sono Netflix, Uber e LinkedIn.**

**Una differenza fra Javascript utilizzato in un browser, quindi incapsulato all’interno di una pagina HTML, e Node.js è l’aggiunta di altri file Javascript che forniscono funzioni create da terzi. Node.js non ha una pagina HTML che racchiude tutti i file e li interconnette tra di loro ma ha bisogno di un costrutto separato il “require”. Require è il costrutto con cui Javascript aggiunge i moduli, che sono come delle librerie che racchiudono funzionalità aggiuntive. Quindi, utilizzando la struttura dei moduli si può sia importare codice di terzi per usufruirne oppure suddividere il proprio codice per rendere più chiara la visualizzazione e il debugging.**

**Ovviamente Node.js già racchiude alcuni moduli principali che mettono a servizio funzioni per gestire i file di sistema, il networking, input e output, etc...**

**Per gestire l’installazione, la rimozione e le varie dipendenze dei moduli Node.js ha di per se un package manager chiamato npm.**

**Node.js ha un modello di esecuzione del codice basato sugli eventi asincroni che non bloccano il flusso in input o in output, in contrasto con l’uso dei threads per avere il multi-tasking.**

**Questo modello fa affidamento alle funzioni di callback per avvisare quando un’attività in sospeso è stata completata. Quindi, da una parte se viene eseguita una funzione che apre un file, ad essa è associata una funzione di callback che una volta ricevuto il file ne gestisce il contenuto, dall’altra viene eseguito il codice successivo che non richiede il file per il suo funzionamento. Ciò fa sì che il programmatore non deve strutturare semafori o cose simili affinché il codice venga eseguito asincronamente e rende, inoltre, l’applicazione veloce.**

3.3 Express

Node.js è solamente un ambiente runtime che esegue Javascript e costruire un web server completo di tutti i servizi direttamente con Node.js non è un compito facile, soprattutto non è necessario. Express è un framework che semplifica la costruzione del codice.

Questo framework permetta la definizione di “routes”, le quali specificano cosa fare quando arriva ad un certo url una richiesta HTTP. Questi end-point si possono anche configurare tramite espressioni regolari in modo da rendere la “route” flessibile a molteplici richieste in modo facile. Express analizza autonomamente l’URL, gli headers e i parametri della richiesta e in base alla risposta determina il codice di risposta, i coockies etc. Inoltre, si possono costruire dei middleware, del codice che si inserisce nel processo di risposta o di richiesta, che realizzano delle funzionalità aggiuntive come l’autenticazione, il login etc.

3.4 MongoDB

MongoDB è un database NoSQL document-oriented, dove NoSQL sta a significare non relazionale, con uno schema flessibile e un linguaggio di query basato su JSON. Esso è basato su documenti che racchiudono tutte le informazioni riguardo un oggetto e i documenti a loro volta sono raggruppati in collezioni. Ogni documento in una collezione ha un identificativo unico tramite il quale ci si accede.

La memorizzazione di un oggetto in un database MongoDB non deve seguire uno schema preciso. Infatti, ogni documento in una collezione spesso ha dei campi differenti dagli altri e inoltre si possono aggiungere o rimuovere campi di un documento specifico senza dover modificare la collezione intera.

4. Strumenti, librerie e tecnologie accessorie

Di grande importanza nello sviluppo del progetto sono state anche le tecnologie esterne alla stack MERN, cioè tutto l’ecosistema di sviluppo che si occupa di monitorare, facilitare, velocizzare e memorizzare il progetto.

Di seguito cito le più importanti descrivendone il funzionamento e le caratteristiche.

4.1 Github

Github è il principale servizio di hosting di programmi nel web. È bastato su repositories cioè cartelle virtuali che contengono codice oppure progetti che sono un insieme di cartelle. Github ha molteplici funzionalità come la possibilità di clonare (copiare) il codice reso disponibile da altri per modificarlo e svilupparne una versione personalizzata, dividere il progetto in branch (rami) per sviluppare in modo autonomo parti di codice o per creare differenti versioni del programma. Inoltre, sono disponibili servizi per la manutenzione e lo sviluppo dei programmi come un issue tracker e un servizio di sicurezza che notifica la presenza di eventuali password o key esposte nel codice.

4.2 JSON

JSON, [acronimo](https://it.wikipedia.org/wiki/Acronimo) di JavaScript Object Notation, è un formato adatto all'interscambio di dati fra [applicazioni](https://it.wikipedia.org/wiki/Applicazione_(informatica)) [client/server](https://it.wikipedia.org/wiki/Sistema_client/server). Esso è basato su Javascript ed è usato come alternativa a XML/XSLT. Ha una struttura molto flessibile ed è capace di contenere ogni tipo di dato, anche un altro JSON. Un esempio è il seguente:

{

"name": "Mario",

"surname": "Rossi",

"active": true,

"favoriteNumber": 42,

"birthday": {

"day": 1,

"month": 1,

"year": 2000

},

"languages": [ "it", "en" ]

}

Come si può vedere l’oggetto JSON è formato da delle coppie nome/valore separate da il simbolo “:”, quindi la struttura dell’oggetto si può riassumere con la seguente immagine:

[[1]](#footnote-1)

4.3 React-Bootstrap

Bootstrap è una libreria specificatamente creata per rendere più semplice lo sviluppo del front-end fornendo in maniera open source vari stili per ogni elemento HTML come bottoni e form di input, inoltre facilita l’organizzazione del layout e mette a disposizione alcuni costrutti come la barra di navigazione e lo slideshow. React-Bootstrap è semplicemente Bootstrap riformattato per avere ogni componente come componente React senza dipendenze non necessarie come JQuery. Nella mia tesi ho fatto un uso estensivo di questa libreria usufruendo di vari componenti come il Carousel, la navbar, le form e il componente Jumbotron.

Organizzare il layout con questa libreria è molto semplice potendo divedere i vari componenti e incapsulandoli in un costrutto formato da righe e colonne. Ad esempio, per fornire un campo di input nella stessa riga di un bottone possiamo semplicemente avere questo codice:

Immagine che contiene screenshot

Descrizione generata automaticamente[[2]](#footnote-2)

E nello stesso modo per fare un semplice Carousel è possibile utilizzare questo semplice snippet di codice:

Immagine che contiene screenshot

Descrizione generata automaticamente

4.4 REST

La gestione della comunicazione tra il client e il server viene gestito secondo il modello architetturale REST (Representational State Transfer).

Questo modello architetturale è stato introdotto nel 2000 nella tesi di dottorato di Roy Fielding, uno dei principali autori delle specifiche dell'Hypertext Transfer Protocol (HTTP), e vennero rapidamente adottati dalla comunità di sviluppatori Internet.[[3]](#footnote-3)

REST è composto da vincoli quali:

* Avere un’architettura di tipo client-server nello strato di networking, cioè avere un server che mette a disposizione dei servizi al client. Questo fa si che ci sia una separazione dei compiti, in quanto il server si occupa della gestione dei dati e il client della visualizzazione e della User Interface;
* Essere “stateless” cioè ogni richiesta da parte del client deve avere tutte le informazioni richieste dal server per poter fruire del servizio, senza la necessità di accedere a dati memorizzati in precedenza o successivamente. Ciò beneficia molto l’architettura in quanto dà più visibilità al monitoraggio del sistema e una maggiore scalabilità non essendo necessario salvare dati aggiuntivi in memoria;

Immagine che contiene screenshot

Descrizione generata automaticamente

* Un’interfaccia uniforme tramite la quale la pagina web può comunicare con differenti tipologie di devices. Questo vincolo è ottenuto grazie ai browser di navigazione;
* Un sistema diviso in livelli nei quali suddividere i componenti dell’applicazione web come per esempio un livello che si occupa di gestire le chiamate verso le API esterne e un altro livello che si occupa di richiedere e gestire i dati da un database.

Inoltre, l’architettura REST si basa sulle risorse identificate in modo univoco da URL (Uniform Resource Locator) e sulle azioni che si possono compiere su queste risorse.

Queste azioni dono descritte dall’acronimo CRUD che significa create, retrive, update e delete.

Queste azioni sono mappate su dei verbi HTTP che sono GET, POST, PUT, DELETE, HEAD, OPTIONS.

4.5 Nodemon

Uno strumento che mi è stato di grande aiuto durante lo sviluppo del progetto è stato Nodemon.

Nodemon monitora i file all’interno di una cartella o di più cartelle contemporaneamente e quando nota un cambiamento nel codice riavvia automaticamente l’applicazione, eliminando il bisogno di avviare manualmente il server ad ogni modifica del codice.

L’installazione di questo strumento è fatta semplicemente con npm e la sua esecuzione può essere delegata ad uno script in modo da fare il deploy del progetto con solamente un comando.

5. MusicDiscover

MusicDiscover è il nome del progetto e, come si può intuire dal titolo, è una web application che ha lo scopo di fornire all’utente informazioni riguardo ai suoi artisti preferiti, al fine di scoprire artisti simili all’artista cercato oppure di avere informazioni riguardo canzoni e album dello stesso.

L’applicazione è strutturata in un server back-end e in un front-end diviso in quattro diverse sezioni, descriverò qui in seguito sia la parte back-end che front-end.

Tutto il codice dell’applicazione è disponibile su Github al seguente link:

<https://github.com/Silver2104/Progetto_tesi_musicdiscover>

5.1 Architettura dell’applicazione

Immagine che contiene screenshot

Descrizione generata automaticamente

L’architettura di questo progetto è basata sul paradigma client-server in cui il client fa una richiesta http al server ed esso in risposta manda il contenuto della pagina web.

Il server è composto da due layer per separare la logica per la gestione dei dati e la logica per la creazione e la renderizzazione delle pagine richieste dall’utente.

I server dialogano con l’architettura networking REST perché ogni richiesta/risposta contiene tutti i dati necessari e non ha bisogno di dati aggiuntivi derivanti da successive richieste/risposte, inoltre il dialogo tra i due server avviene solamente tramite URL senza apertura di socket dirette.

La tecnologia di base su cui si basano entrambi i server è Nodejs, successivamente su ogni server è installata la libreria/framework che specializza il server o nella gestione dell’API o nella creazione del front-end.

La creazione dell’API è fatta tramite il framework express che tramite la sua funzione di routing semplifica la gestione delle callback e degli eventi riguardanti i punti di accesso delle API.

Per la gestione del front-end si utilizza React che tramite server-side rendering minimizza la necessità del client di eseguire codice, inoltre tramite un altro framework react-bootstrap si è andata a sviluppare un’interfaccia grafica intuitiva e moderna.

Per mettere suddividere la logica delle pagine in sezioni separate si è fatto uso del modulo react-router-dom.

Lo scambio dei dati è fatto unicamente tramite JSON per la sua facilità di visualizzazione in debugging vista la similarità con le strutture dati Javascript.

5.2 API server Layer

Questo progetto, essendo basato sulla stack MERN, ha come runtime JavaScript Nodejs che è guidato da eventi asincroni e in aggiunta a Nodejs viene utilizzato il framework Express che, tramite le sue features, rende la creazione del back-end molto intuitiva e flessibile.

Per controllare il flusso delle richieste e ampliare le funzionalità del server Express utilizza i middleware. I middleware sono delle funzioni che possono accedere alla richiesta ricevuta e alla risposta da mandare al client, queste funzioni possono eseguire queste azioni:

* Eseguire qualsiasi codice.
* Apportare modifiche agli oggetti richiesta e risposta.
* Terminare il ciclo richiesta-risposta.
* Chiamare la successiva funzione middleware nello stack.

Queste funzioni possono essere concatenate tramite la funzione next() creando delle sotto-stack middleware.

Queste middleware devono essere montate su dei percorsi URI denominati routes.

Una route può essere composta da particelle per far combaciare molteplici richieste tutte verso la stessa route, similmente alla struttura delle espressioni regolari.

I principali middleware usati nell’applicazione MusicDiscover sono:

* app.use(express.json()) : che si occupa di adattare i dati ricevuti e inviati con la sintassi Json;
* app.use("/download", downloadPage);
* app.use("/lastfm", LastfmPage);
* app.use("/twitter", TwitterPage).

Gli ultimi tre middleware sono usati per divergere le richieste fatte dall’utente per i vari servizi messi a disposizione verso una sotto-stack middleware che si occupa di gestire le richieste e rispondere in maniera appropriata per ogni servizio.

Ogni sotto-stack è definita in un file alternativo rispetto al main file server.js per rendere il codice modulare e più facile da debuggare rispetto ad un codice monolitico in cui tutte le funzioni sono nello stesso file.

5.3 Front-end server layer

Qui di seguito saranno descritte le funzioni delle varie pagine disponibili nell’applicazione e verranno spiegati concetti fondamentali della libreria React attraverso esempi presi direttamente dal progetto.

5.3.1 Pagina Homepage

La prima pagina è l’Homepage la quale spiega il funzionamento delle altre sezioni e le loro limitazioni, in quanto non tutti gli artisti potrebbero avere la stessa copertura d’informazioni necessarie per il corretto funzionamento dei servizi disponibili.

Questa pagina, essendo molto semplice, è perfetta per introdurre il concetto delle classi in React. Come ho detto in precedenza, la struttura base in React è il “Component”, cioè una classe che gestisce autonomamente la propria renderizzazione e il suo stato delle informazioni. Una classe componente si dichiara nel seguente modo (linea 3 di codice):

Immagine che contiene screenshot, schermo

Descrizione generata automaticamente.

Si nota come la classe abbia solamente un metodo di nome “render”. Questo metodo è l’unico necessario per il corretto funzionamento del codice e specifica a React cosa deve essere renderizzato quando si carica nel browser il componente.

Il linguaggio che si utilizza nel metodo render è JSX e, come spiegato in precedenza, grazie ad esso utilizzare i tag HTML per comporre il layout della pagina e aggiungere elementi in maniera organizzata diventa facile perché ha gli stessi vincoli di HTML.

Per aggiungere script in Javascript con JSX, come ad esempio una chiamata a funzione per la creazione di elementi HTML o componenti React, basta racchiudere il codice tra parentesi graffe.

Lo spazio prima del metodo render è utilizzato per definire i metodi specifici della classe, come ad esempio il costruttore, oppure per definire variabili e strutture dati.

Per chiamare queste strutture dati, oppure per chiamare funzioni, è necessario anteporre al nome/metodo il costrutto “this” per far riconoscere a React che il metodo o la struttura fa parte della classe e non dell’ambiente esterno.

La classe Homepage per essere visualizzata propriamente nel browser deve prima essere caricata nel file main di React che generalmente è App.js e poi montata nel file index.html che è il diretto intermediario fra React e il browser.

Qui di seguito mostro il contenuto del file App.js per spigare come aggiungere il codice di un componente secondario nel file principale di React:

Immagine che contiene screenshot

Descrizione generata automaticamente

Per importare un componente nel metodo di rendering si utilizzano dei tag simili a quelli HTML, in cui il nome del tag è dettato dal nome della classe che comprende il codice che si vuole importare e renderizzare, per esempio in questo caso per importare il componente della barra di navigazione si utilizza il codice alla riga 15 del file App.js (<Navbar />).

Nel file App.js oltre ad importare la barra di navigazione è possibile notare che ho importato due altri componenti, il componente “Route” e il componente “Router” entrambi definiti nel modulo “react-router-dom”.

Questi due componenti hanno il compito di facilitare la programmazione della navigazione con gli URL da parte del client, infatti ad ogni route con path specificato corrisponde l’URL con lo stesso path.

5.3.2 Pagina Bandcamp

La seconda pagina della web application offre il servizio di visualizzare i vari album di un artista e la possibilità di fare il download delle canzoni dell’album.

Per cercare l’artista ho messo a disposizione una semplice form composta da un campo in cui è necessario mettere il nome dell’artista e un bottone per inviare la richiesta al server.

Successivamente il server ricevuta la richiesta prenderà i dati da bandcamp e li invierà al client, in seguito è possibile vedere nel browser il risultato della risposta del server

Immagine che contiene screenshot

Descrizione generata automaticamente

Per inviare la richiesta al server ho usato un modulo esterno chiamato Axios.

Axios è basato sulle promesse (argomento che spiegherò nel seguito) per effettuare chiamate http sia da lato browser sia da lato server, quindi ritornando al concetto fondamentale che la stack MERN è isomorfa, cioè le tecnologie sono intercambiabili tra front-end e back-end visto che il linguaggio di programmazione è lo stesso.

Ho usato Axios perché semplifica la creazione di richieste http settando autonomamente gli header e altre impostazioni.

Inoltre, Axios essendo basato sulle promesse è asincrono e usando la costruzione “then” e “catch” è possibile gestire la risposta tramite callback, sempre ricordando il paradigma ad eventi che è presente estensivamente anche in nodejs.

Di seguito è il codice che si occupa di inviare la richiesta dei dati al server:

Immagine che contiene telefono, cellulare, sedendo, portatile

Descrizione generata automaticamente

Le informazioni ricevute dal server, determinando la modifica della visualizzazione del componente, sono salvate nello “stato” del componente e non in una struttura dati aggiuntiva.

Avendo parlato di “stato” di un componente è necessario fare una digressione per spiegare questa struttura dati, fondamentale in React.

Questa struttura dati è specifica di ogni componente e viene inizializzata tramite il metodo costruttore della classe, all’interno è norma mettere dati i quali, tramite la gestione di eventi, determinano il cambiamento nella visualizzazione del componente nella pagina.

Infatti, React ogni volta che avviene un cambiamento nella struttura dati “state” inizializza la rirenderizzazione del componente.

L’accesso allo stato è fatto necessariamente tramite la particella “this.” per specificare che si sta parlando della struttura dati propria del componente, come anche quando si utilizza la struttura dati “props”.

La modifica dei dati all’interno dello stato non deve essere fatta chiamando lo stato direttamente, ma tramite il metodo specifico dello stato chiamato “setState” il quale ha come parametro le modifiche necessarie da apportare allo stato. Da notare che la modifica dello stato avviene in maniera asincrona e che React, per migliorare le prestazioni, potrebbe accorpare due o più chiamate a setState, quindi è bene gestire propriamente gli eventi che determinano un cambio dei parametri di stato di un componente.

Ritornando al codice del progetto, lo stato di BandCampPage contiene, oltre che alle informazioni ricevute dal server, anche la variabile booleana isShowBandcampDiv che gestisce la visualizzazione degli album nella pagina.

Infatti, essa serve per controllare l’evento di renderizzazione del componente in cui ci sono le informazioni ottenute dal server e se non ci fosse React cercherebbe di renderizzare dei componenti senza informazioni e ciò manderebbe in errore la renderizzazione del componente nel sito.

Questo controllo è fatto semplicemente mettendo in and (&&) il componente da renderizzare e la variabile booleana:

Immagine che contiene cibo

Descrizione generata automaticamente

Le informazioni per comporre gli album nel browser sono date dal client attraverso un API gestita del server da me sviluppato e l’end-point per usufruire di questa API è “/download” a cui deve essere associato un parametro, “download\_link”.

Una volta fatta la richiesta al server esso si occuperà di recuperare i dati dal sito Bandcamp tramite l’utilizzo delle espressioni regolari, che ho utilizzato per mancanza di API da parte di Bandcamp.

Per trovare un’alternativa alle espressioni regolari, ho provato ad utilizzare dei moduli per il web scrapping come Cheerio.js. Tuttavia, nonostante siano facili da utilizzare, ho preferito continuare ad utilizzare le espressioni regolari per la loro velocità di esecuzione e la loro flessibilità.

Il server recupera le informazioni in maniera sequenziale per mantenere la consistenza rispetto alla data di uscita degli album, ciò è fatto tramite chiamate di funzioni nidificate con il costrutto delle premesse.

In Javascript una promessa è un oggetto usato per eseguire funzioni asincrone che rendono l’esecuzione di azioni non bloccanti nel server e sono una struttura che si può schematizzare nel seguente modo:

Immagine che contiene screenshot

Descrizione generata automaticamente

All’inizio della cascata di promesse creo un oggetto “info” che è il contenitore delle informazioni che successivamente dovranno essere mandare e già contiene tutti i campi necessari al servizio.

Successivamente questo oggetto viene passato nella cascata di funzioni e ogni funzione è incaricata di fare una copia dell’oggetto, compilare le informazioni di un campo e passare l’oggetto alla funzione successiva.

In questo modo dividendo il recupero di informazioni tramite molteplici funzioni si semplifica di molto il debugging e il testing, potendo testare una funzione alla volta senza dover eseguire tutto il blocco.

Successivamente se il recupero dei dati è andato a buon fine il server manderà l’oggetto info al client, questo oggetto è in formato Json e contiene diverse informazioni, ma il dato più importante è l’array “albums” questo array è formato da una serie di oggetti che hanno come campi:

* Link\_album: che è il link a cui è possibile comprare l’album e usufruire dei servizi di Bandcamp;
* foto\_album: Link alla foto della copertina dell’album necessaria per visualizzare le immagini nel browser;
* Nome\_album: Il nome è esplicativo;
* Canzoni: È la lista delle canzoni che compongono l’album

Se si vuole testare questa API è sufficiente andare al seguente link:

“localhost:3001/download/?download\_link=https://noveller.bandcamp.com/”.

5.3.3 Pagina Lastfm

Successivamente, nella terza pagina, abbiamo la parte che si occupa di servire al cliente informazioni più dettagliate sull’autore, esse sono prese dal sito Lastfm e sono richieste tramite l’API dello stesso.

Le API utilizzate sono:

* /artist.getTopTrack (Limite posto a 5);
* /artist.getTopAlbums (Limite posto a 5);
* /artist.getInfo.

Per poter usufruire di questo servizio API è stato necessario creare un account developer in modo da poter autenticare l’applicazione con una chiave. Questa chiave è univoca per ogni applicazione registrata nell’account ed è necessario inserirla in ogni chiamata all’API per poter autenticare la provenienza della chiamata.

Inoltre, per prendere le foto dell’artista è stato necessario utilizzare un’espressione regolare a causa di una nuova policy sulle API da parte di Lastfm che fa in modo di rendere non disponibili le immagini tramite API.

Le informazioni riguardo l’artista che ho scelto di mostrare nella mia web application sono quindi le seguenti:

* Le foto dell’artista;
* La biografia, incentrata sull’aspetto musicale;
* Le canzoni più sentite in un lasso temporale di una settimana;
* Gli album più popolari;
* Gli artisti simili.

I dati riguardo canzoni, albums e artisti simili sono quelli che ritengo più importanti perché sono tramite questi dati che l’utente può scoprire altri artisti partendo da un artista di riferimento, oppure può scoprire gli album e le canzoni più popolare che potrebbe non aver ancora sentito.

Iniziando con le canzoni più popolari dell’artista l’API mette a disposizione solamente le canzoni più ascoltati in generale, non potendo specificare un lasso di tempo da poter analizzare, quindi nella mia applicazione ho potuto mettere solamente quest’ultimo gruppo di canzoni. Ma andando a vedere il traffico di rete, con la console di sviluppo di un browser, della pagina dell’artista su Lastfm si può notare come la pagina, per ottenere i brani più popolari in un lasso di tempo diverso da quello dell’API, faccia semplicemente una richiesta alla pagina dell’artista specificando alcuni campi aggiuntivi, per esempio “+partial/tracks?top\_tracks\_date\_preset=LAST\_180\_DAYS&ajax=1”, in questo modo si può accedere alle canzoni più popolari di un artista in un lasso di tempo di sei mesi. Per capire quali sono i brani più popolare il sito Lastfm utilizza un metodo basato sugli ascolti, che sono contati in maniera diversa secondo il device che si utilizza per ascoltare la musica, per esempio se la canzone viene ascoltata tramite browser bisogna ascoltare tutto il brano affinché venga contata come ascolto, invece se la si ascolta sul programma desktop c’è un tetto minimo di tempo per contare la canzone come ascoltata.

Riguardo agli albums la popolarità di questi ultimi, similmente alle canzoni, è basata sugli ascolti cumulativi delle varie canzoni che compongono l’album singolo. Ma, a differenza delle canzoni, nel caso degli album non è presente la possibilità di scegliere un periodo temporale nel quale analizzare la popolarità dell’album.

In ultima analisi, per capire se due o più artisti sono simili si usa una combinazione di due fattori:

1. Il primo fattore tramite il quale si può capire se due artisti sono simili è semplicemente fare un’analisi dei tag degli artisti (un tag è un genere di musica, per esempio “rock”). Un autore che avrà molteplici tag uguali o simili ad un altro autore avrà una forte probabilità di comporre musica simile;
2. Il secondo fattore, tramite il quale si può capire la similarità tra due autori, è basato su cosa ascolta l’utenza in un dato momento. Gli artisti più ascoltati da un utente in un lasso di tempo breve, meno di un mese, sono con forte probabilità correlati tra di loro. Questo fatto è esperienza comunque e si basa sul fatto che le persone siano di base inclini ad ascoltare in un lasso di tempo breve musica dello stesso genere o simile.

Ovviamente questi due fattori sono combinati per avere un’accurata analisi degli artisti simili ad un dato artista.

Per visualizzare i vari dati ottenuti dal sito sul front-end della mia applicazione ho deciso di suddividere questi dati in tre componenti. I componenti sono stati scelti in base al tipo di contenuto che devono contenere, infatti ho utilizzato tre diversi elementi per rappresentare tre diversi tipi di dato.

Come per le altre pagine, per sviluppare in maniera semplice e veloce il front-end ho utilizzato la libreria React-Bootstrap.

Il primo è un Carousel semplice per contenere in maniera compatta le foto prese dall’API.

Il Carousel è dinamico in quanto si adatta al numero di immagini che sono state fornite dall’API. Per fare ciò ho utilizzato il linguaggio JSX, nello specifico ho utilizzato il metodo map sull’array dei link delle immagini fornito dall’API. Così facendo React crea automaticamente per ogni link contenuto nell’array un “Carousel.Item” che all’interno contiene il tag HTML “img” che ha come parametro “src” il link dell’immagine.

Il secondo componente è un’area di testo che si occupa di visualizzare la biografia dell’artista.

Il terzo componente è formato da un pannello “Card” in cui nell’header c’è la possibilità di selezionare se si vogliono vedere le canzoni o gli album più popolari e il body della Card è formato dall’elenco delle canzoni/albums. Ad ogni canzone o album è associato un bottone che se cliccato porta alla pagina di Youtube in cui si è fatta automaticamente la ricerca della canzone/album in questione.

L’ultimo componente è anch’esso un componente Card in cui visualizzo gli artisti simili nell’ordine proposto dall’API e anche in questo caso ad ogni artista è associato un bottone che porta alla ricerca dell’artista su Youtube.

Per prendere i dati dal sito Lastfm il client fa una chiamata prima all’API del mio server che a sua volta fa delle chiamate all’API di Lastfm.

Quindi il server si occupa di compattare i vari dati che sono forniti dal sito di Lastfm in un unico oggetto JSON che contiene tutte le informazioni necessarie per visualizzare in maniera corretta la pagina front-end.

Per fare queste varie chiamate ho utilizzato una funzione asincrona “asyncRunner” che comprende le varie chiamate all’API. Questa funzione asyncrona fa in modo di chiamare parallelamente più funzioni grazie al costrutto “await” che si applica solamente a funzioni promises. Quando ogni promises è stata soddisfatta la funzione asyncRunner ritorna i vari dati che il server commuta in un unico oggetto JSON che sarà passato al client. Quindi utilizzando async-await si ottiene una riduzione del tempo necessario per ottenere i vari dati dall’API di Lastfm. In seguito, si può vedere la funzione in codice:

Immagine che contiene testo, tavolo, sedendo, portatile

Descrizione generata automaticamente

5.3.4 Pagina Twitter

L’ultima pagina di questa web application offre un servizio di interfacciamento con l’account di Twitter dell’artista scelto.

Tramite la barra di ricerca è possibile inserire il nome dell’account di Twitter dell’artista per avere gli ultimi aggiornamenti riguardo Tweets e Retweets.

Per poter avere le informazioni riguardo agli ultimi aggiornamenti dell’artista su Twitter si è utilizzata l’API di quest’ultimo.

L’API che si è scelto di utilizzare è stata sviluppata specificatamente per il web e comprende un insieme di strumenti per rappresentare nella pagina di un’applicazione il contenuto di Twitter.

Nello specifico il progetto fa uso dello strumento “Embedded Tweets” che permette di prendere singoli Tweet con tutte le informazioni riguardo quest’ultimo, come likes, commenti, orario di pubblicazione e media connessi.

La visualizzazione di queste informazioni è gestita automaticamente da Twitter tramite il codice HTML (snippet HTML) che è generato dall’API e dalla libreria Javascript nativa di Twitter appositamente creata per gestire questo tipo di contenuti.

Tramite oEmbed, che sarebbe lo strumento per creare questi snippet di codice HTML e Javascript, è possibile personalizzare i parametri del componente Tweet creato, ad esempio è possibile utilizzare il tema chiaro o il tema scuro per far combaciare il Tweet con lo stile della pagina, è possibile nascondere i media allegati al Tweet per velocizzarne il caricamento ecc…

Per utilizzare questo strumento di embedding dei tweets messo a disposizione da Twitter non è necessaria una chiave di autenticazione da parte dell’applicazione, basta utilizzare lo strumento online. Ma, per automatizzare e rendere dinamica la creazione di questi tweets embedded rispetto all’autore cercato ho utilizzato un ulteriore API Twitter nella mia applicazione:

* https://api.twitter.com/1.1/statuses/user\_timeline.json?screen\_name=${account\_name}&count=9.

Il metodo usato per richiedere i dati, come si può vedere dall’URL sopra riposto, è “user\_timeline” che, specificando il nome dell’utente con il campo “account\_name” e tramite la limitazione imposta dal parametro “count=9”, prende la timeline dell’utente con una profondità di nove tweets inoltre, “.json” specifica il tipo di formato da utilizzare per inviare i dati. Il bearer token utilizzato per autenticare l’applicazione è messo nell’header della richiesta tramite il parametro “Authorization”.

L’oggetto che sarà inviato al server conterrà molteplici campi ma per utilizzare lo strumento di embedding dei tweets è necessario solamente l’id del tweet, quindi il server invece di mandare al client tutto l’oggetto Json, rendendo più pesante l’elaborazione lato client, estrapolerà i vari id dei tweet e con quelli comporrà un altro oggetto Json e solo quest’ultimo sarà inviato al client.

L’API messa a disposizione del server prevede due URI accessibili solamente tramite metodo POST:

* /twitter
* /twitter/getTimeline

La prima ha la funzione di mandare al client solamente gli id dei tweets.

Invece la seconda terminazione manda l’oggetto Json che è stato richiesto all’API di Twitter, questa seconda terminazione è stata aggiunta per il principio di espanzione delle funzionalità del server, in quanto se diversi servizi richiedessero lo stesso oggetto Json, invece di creare codice ridondante per prendere l’oggetto tramite chiamata all’API di Twitter si utilizza direttamente l’API del server.

In un primo stato l’applicazione prevedeva l’utilizzo manuale dello strumento di embedding dei tweets rendendo il codice più lungo e complesso da debuggare, ma applicando il principio del riutilizzo del codice si è andato da un utilizzare un modulo creato dalla community di npm, chiamato “react-twitter-embed”.

Questo modulo è stato creato appositamente per React e rende la creazione dei componenti Tweet molto più semplice. Quindi utilizzando la seguente funzione:



Si è resa automatica la creazione e la visualizzazione dei Tweet nella pagina del browser. Inoltre il componente non ha lo scopo solamente di visualizzare le informazioni del Tweet ma sono incorporate alcuni metodi per gestire l’interazione con il tweet tramite il proprio account Twitter, infatti andando a cliccare sul pulsante dei like e dei commenti l’utente sarà rindirizzato alla pagina dell’artista e immettendo i propri dati per fare l’accesso sarà possibile mettere like automaticamente al post.

Bibliografia

* SUBRAMANIAN, Vasan. Pro MERN stack: full stack web app development with Mongo, Express, React, and Node. Apress, 2017. Capitolo 3 in generale
* FREEMAN, Adam. Pro React 16. Apress, 2019. Capitolo 3.1
* HOLMES, Simon. Getting mean with mongo, express, angular, and node. Manning Publications Co., 2015. Capitolo 2
* DOGLIO, Fernando; DOGLIO; CORRIGAN. REST API Development with Node. js. Apress, 2018. Capitolo 4.4

1. https://www.json.org/json-it.html [↑](#footnote-ref-1)
2. https://react-bootstrap.github.io/components/forms/ [↑](#footnote-ref-2)
3. https://it.wikipedia.org/wiki/Representational\_State\_Transfer [↑](#footnote-ref-3)