Relazione Progetto di Progettazione di Applicazioni Web e Mobili

Autore: Luca Bianchi

Applicazione

Il progetto da me sviluppato riguarda un'applicazione per la gestione delle Note.
All'interno di questa applicazione è quindi possibile andare ad operare le classiche operazioni effettuabili sulle note, perciò è possibile creare una nota, così come è possibile modificarla o anche eliminarla.

Autenticazione

Nell'andare a fare le varie operazioni sulle note è necessario però prima Autenticarsi.
Questo vuol dire che prima vi sarà una fase di Registrazione dove vengono richieste: Email, Password e Username.
Una volta registrati nell'applicazione si potrà effettuare il Login sempre fornendo Email e Password (utilizzate nella fase di registrazione).

L'autenticazione dell'utente sarà ovviamente necessaria per poter risalire alle Note di quel determinato utente.

Più precisamente, per quanto riguarda "come" avviene il processo di Autenticazione si utilizzano i JWT (JSON Web Token).
Perché piuttosto che utilizzare le sessioni, per un'applicazione come questa (dove il frontend è una SPA) è più comodo utilizzare i Token.
Perciò al termine dell'autenticazione, viene restituito al client un Token (firmato ovviamente) dove nella parte del Claim sono inserite come "informazioni personalizzate" Email e Username.
(Principalmente il dato utile è l'email, in quanto è attraverso di essa che vengono reperite le note relative all'utente nel Database)

Per quanto riguarda invece cosa accade lato frontend, vale a dire l'applicazione Angular, una volta che viene ricevuto il JWT (se l'utente si è autenticato correttamente) esso viene salvato, per poi essere utilizzato per mezzo di un Interceptor. Questo perché piuttosto di reinserire il Token in ogni singola richiesta (a rotte protette) da inviare verso il backend, è molto più comodo per non ripetere codice utilizzare un qualcosa che prima di inoltrare la richiesta verso il backend, intercetta la richiesta e vi aggiunge tra gli Header della richiesta il Token in questione.

Logs

Date le varie operazioni che possono essere effettuate sulle note, viene anche resa disponibile una sezione in cui è possibile controllare tutte le operazioni di ADD, UPDATE e DELETE che sono state effettuate.

Tuttavia vi sono solo delle rotte per poter leggere i Log, senza però andare a crearli o modificarli, questo perché sono in realtà creati in modo del tutto autonomo dal backend una volta effettuate delle operazioni sulle note.

Tecnologia

Le tecnologie utilizzate in questa applicazione sono svariate.
Partendo prima di tutto con il backend è stato realizzato con il framework NestJS, utilizzando alcuni dei paradigmi messi a disposizione dal framework.
All'interno del file package.json contente le dipendenze possiamo notare alcune dipendenze ad altri pacchetti appartenenti al repository "npm", ad esempio:

- mongoose: per poter gestire il database ed effettuare tutte le query in modo facile e veloce.
- bcrypt: per poter memorizzare le password nel database (sfruttando sia il Secret che il Salt).
- class-validator e class-transformer: per poter gestire gli "Schema" all'interno del database.
- passport e passport-jwt: per implementare in modo molto più semplice e standard l'autenticazione per mezzo di JWT.
- sanitize-html: per poter sanificare il testo proveniente dal frontend (data la presenza di molti form).

Per quanto riguarda il frontend, è stato sviluppato in Angular.
Le principali dipendenze utilizzate in questo caso sono quelle relative al framework CSS "Angular Material", cioè una libreria già pronta con del CSS per poter semplificare la parte relativa alla visualizzazione delle componenti.
Ovviamente il frontend comunica con il backend effettuando chiamate HTTP verso lo stesso.

Infine come database è stato utilizzato MongoDB (vale a dire un database NoSQL).
E come già detto, viene per questo utilizzato nel backend "mongoose".
Inoltre nel backend tutti i file chiamati <nome>.schema.ts sono appunto tutti i file Typescript utilizzati per descrivere gli schemi delle tabelle nel database, in quanto anche se si tratta di un database basato sui "documenti" e non ha una struttura fissa per quanto riguarda i documenti inseriti nelle Collection, è comunque necessario specificare una loro struttura.

Problematiche

Nel backend di questa applicazione vengono inoltre gestite delle particolari problematiche, cioè: XSS, NoSQL Injection e CORS.

La problematica più critica in un progetto del genere è XSS (Cross-Site Scripting) in quanto vi sono molti form che possono essere afflitti da questa problematica.

Essa viene gestita per mezzo della libreria “sanitize-html” (cioè un pacchetto in https://www.npmjs.com/), che va infatti a ripulire il testo ricevuto nel backend da possibili tag HTML.

Successivamente, nonostante la problematica SQL Injection non è presente, in quanto le query non vengono eseguite in MongoDB così come in ogni database relazionale, è comunque presente un altro tipo di problematica particolare per cui se passato in input un particolare testo è comunque possibile iniettare dei comportamenti particolari nel testo passato nei form.

Infine, per risolvere la problematica relativa al CORS (Cross-Origin Resource Sharing), dato che a meno che non siamo in fase di produzione (in cui il backend andrà a fornire come contenuti statici i file prodotti dal comando “ng build”, cioè il comando per fare una compilazione di Release per un progetto Angular), abbiamo due Web Server diversi (rispettivamente quello di NestJS gira dietro la porta 3000, mentre quello di Angular dietro la porta 4200), e scaricando le rispettive risorse da entrambi, quando l’applicazione Angular farà una chiamata al backend, avendo due origini diverse verrà generato CORS.
Tuttavia piuttosto di risolvere la problematica abilitando il CORS lato backend (che resta comunque un'operazione pericolosa se poi ci si dimentica di disattivarlo), la soluzione migliore è utilizzare un Proxy lato frontend. E dalla configurazione del file “proxy.conf.json”, è possibile notare che tutte le chiamate effettuate verso “localhost:4200/api” vengono reindirizzate verso “localhost:3000/api”, così facendo non genererà più CORS.
Questo meccanismo però come già detto è solo utilizzato in fase di sviluppo (infatti nel file angular.json, il proxy viene inserito solamente sotto il comando “ng serve”).

API

Per quanto riguarda le rotte con cui è possibile interrogare il backend e i relativi metodi HTTP utilizzabili, esse sono:

- POST .../api/auth/login : per il Login
- POST .../api/auth/register : per la Registrazione
- GET .../api/notes : per leggere tutte le note
- GET .../api/notes/:id : per richiedere la nota con il dato ID
- POST .../api/notes : per creare una nuova nota
- DELETE .../api/notes/:id : per eliminare la nota con il dato ID
- PUT .../api/notes/:id : per modificare la nota con il dato ID
- GET .../api/logs : per leggere tutti i Log
- GET .../api/logs/:id : per richiedere il Log con il dato ID

(Ovviamente i puntini (...) dovrebbero essere sostituiti con il dominio del backend, che nel caso sia eseguito in locale sarà “localhost:3000/”)
Ovviamente buona parte di esse risulteranno protette, perciò per non ricevere l’errore “401 Unauthorized” sarà necessario fornire un JWT “valido”.

NOTA: Notare anche come nelle rotte non viene ovviamente mai passato lo UserID o l’Email dell’utente direttamente nell’URL per risalire alle note di quel determinato utente, in quanto come già detto sarà possibile risalire alla Email direttamente dal contenuto del JWT.